中山大学硕士学位论文

一种回答集逻辑程序改进分割计算方法及程序化简的研究

Research on Program Splitting of ASP Program and Its Application in Program Simplification

学位申请人：霍子伟

指 导 教 师：万海 高级讲师

专 业 名 称：软件工程

答辩委员会主席（签名）：

答辩委员会委员（签名）：

年　　月　　日

学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究作出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期：　　　年　月　日

学位论文使用授权声明

本人完全了解中山大学有关保留、使用学位论文的规定，即：学校有权保留学位论文并向国家主管部门或其指定机构送交论文的电子版和纸质版，有权将学位论文用于非赢利目的的少量复制并允许论文进入学校图书馆、院系资料室被查阅，有权将学位论文的内容编入有关数据库进行检索，可以采用复印、缩印或其他方法保存学位论文。

学位论文作者签名：

导师签名：

日期：　　　年　月　日

摘 要

人工智能是计算机科学中的一个重要分支，其中的知识表示与推理则是一门通过理解智能和认知本质，以最终通过计算机实现并体现人类智能为目的的学科。人工智能创始人之一的John McCarthy早在上世纪50年代末就开始推动知识表示的发展，他第一次提出了使用符号推理形式来刻画知识的认知和推理。

早年的知识表示是通过单调逻辑进行推理的，但很快人们就意识到人类常识的认知和推理是非单调的，所以非单调逻辑在产生至今都是知识表示的研究热点。本文研究的回答集编程（Answer Set Programming, ASP）则是非单调逻辑的重要内容。随着回答集编程问题的多年发展，其理论和求解器都已比较成熟。然而经过证明，回答集编程中判断正规逻辑程序和析取逻辑程序是否存在稳定模型的计算复杂性分别是**NP**-$complete$和Π_2^P-$complete$，所以回答集编程的主要发展瓶颈是求解器的效率问题。

在求解器提速的发展过程中，Lifschitz 和Turner在1994年时提出了分割集（splitting set）和程序分割（program splitting）的概念，并从理论上证明了ASP程序可以通过分割集被分割为bottom和top两部分，并通过这两部分的回答集计算原程序的回答集。分割集和程序分割的提出，为求解ASP回答集的提速发展带来了新思路。在后续的时间里，分割集和程序分割得到了不断的推广。然而，Lifschitz和Turner当初定义的分割集是基于强条件的，在实际程序中往往会出现一个ASP程序的分割集就只有空集和程序全部原子构成的集合，而这样的分割集对于分割程序是没有任何意义的。

基于这样的应用背景之下，本文对Lifschitz 和Turner提出的分割集和程序分割算法进行了深入的探讨和研究，对原来的分割集和程序分割方法进行了扩展，并把分割的概念引入到程序化简当中。本文所获得的主要成果具体如下：

首先，把原来Lifschitz 和Turner定义的强条件下的分割集扩展为可以是任意原子集，同时定义了新的程序分割方法。在分割集可以为任意原子集的情况下，程序分割的适用性可以大大地得到扩展。此外，本文通过分析得到了新分割集对新程序分割的性能影响，找出了主要性能瓶颈所在。同时，通过实验数据验证了使用新分割集和新程序分割方法计算一个ASP程序的回答集对比直接求解要快，提速效果大致维持在2到3倍。

其次，在分析新分割集对新程序分割的性能影响之时，得到结论：如果分割集中原子都被原程序的每个回答集所满足，那么使用这样的分割集来分割程序可

以大大降低原程序回答集的计算复杂性。基于这个发现，本文把分割集的应用拓展到了程序化简当中，即通过程序结论去化简ASP程序。这里的程序结论就是一个能被程序所有回答集所满足的文字集，本文只取正文字，以达到原子集的效果。程序化简的目的依旧是为了让回答集程序求解提速。

本文提出的新分割集与新程序分割给回答集程序的提速带来了实质性的效果，并由此推广分割集的概念到程序化简等实际应用当中，让回答集程序的求解效率有了长足的提升和新思路。

关键词：非单调逻辑，回答集编程，分割集，程序分割，程序化简

Abstract

Artificial intelligence is a very important subject in computer science. The knowledge representation and reasoning in artificial intelligence aims at completing and reflecting human intelligence in computer through the understanding of intelligence and cognitive nature. John McCarthy, one of the founders of artificial intelligence, has begun to promote the development of knowledge representation in the early 1950s. He first proposed the use of symbolic reasoning form to depict the cognitive and reasoning of knowledge.

Knowledge representation is through monotonic logic for reasoning at its early age. However, people came to realize that human cognition of common sense and reasoning is non-monotonic soon. In this case, non-monotonic logic became the mainstream tool of knowledge representation from then on. In this thesis, the answer set programming (ASP) which we studied is an import content of non-monotonic logic. With many years development of answer set programming problems, its theory and solver has been relatively mature. Nevertheless, the computational complexity of normal logic program (NLP) and disjunctive logic program (DLP) in ASP program under stab le model semantic are $\mathbf{NP}$-$complete$ and Π_2^P-$complete$ respectively after strict proven. Therefore, the main development bottleneck problem in answer set program is solver efficiency.

In the process of ASP solver speeding up development, Lifschitz and Turner proposed the concept of splitting set and program splitting in 1994. Moreover, they provided a method to divide a logic program into two parts which was named as bottom and top, and showed that the task of computing the answer sets of the program can be converted into the tasks of computing the answer sets of these parts. The concept of splitting set and program splitting brought new ideas to speed up solving answer sets of ASP program. In the subsequent period, splitting set and program splitting have been promoted and recognized continuously. However, the notion of splitting set which proposed by Lifschitz and Turner was based on strong conditions. Because of this, the empty set and the set of all atoms are the only two splitting sets for many ASP programs in the actual situation. These two splitting sets made no sense to speed up the answer sets

solving in ASP programs, because these splitting sets cannot divide programs by the splitting methods.

Based on this background, in this thesis, I carried on the deep discussion and research about Lifschitz and Turner' s splitting set and program splitting theorem and extend them. Along this train of thought, I introduce the concept of splitting set to program simplification. The main achievements obtained in this thesis are shown as following details.

Firstly, I extend Lifschitz and Turner' s splitting set and program splitting theorem to allow the program to be split by an arbitrary set of atoms and introduce a new program splitting method. As the splitting set can be an arbitrary set of atoms, the applicability of program splitting can be extended greatly. Besides, this thesis figure out the main performance bottlenecks through analyzing properties of the new splitting method. At the same time, the data coming from experiment in this thesis support the fact that using arbitrary set of atoms as splitting set and the new splitting method to divide ASP program and solve its answer sets is quicker than solve directly. More precisely, it is two to three times faster than the original according the experiment.

Secondly, during analyzing how the new splitting set effect on the performance of new splitting method, I found out that if atoms in the splitting set are satisfied by every answer set of the program, it could release the computational complexity of answer set solving. According to this, this thesis extends the usage of splitting set to program simplification. In the same words, we can use consequence of the ASP programs to simplify themselves. The consequence of an ASP program is a set of literals that are satisfied by every answer set of the program. I took only positive literals to make them as atoms. The purpose of program simplification is still to speed up solving answer sets of ASP program.

This thesis proposes the new splitting set and new splitting method to make contribution to speed up solving answer sets of ASP program which brings a substantial progress, and extends the concept of splitting set to practical application such as program simplification. All these make important improvement to solving ASP program.

Key Words: non-monotonic logic, answer set programming, splitting set,

program splitting, program simplification

目　录

第 1 章　引言

本章将以人工智能的发展过程为主线，层层递进地来介绍本文的研究背景和当前国内外在回答集编程方面的研究成果和发展前景。并描述本文主要的研究问题和工作成果及其对回答集编程领域的发展意义，最后简明扼要地给出本文后续章节的安排结构。

1.1　研究背景与现状

自计算机诞生以来，人们便致力于让计算机拥有智能，希望计算机能够模拟人类的大脑去思考和推理。如何让计算机认知和理解知识，并使用规则进行推理以求解问题是人工智能领域中最为困难但具有重要意义的一类问题，而这类问题被称为知识表示与推理（Knowledge Representation and Reasoning）[1]。经过多年来的研究和发展，知识表示这个领域已经得到了长足的发展，其最主要的描述和求解工具则是逻辑程序[2]。在发展过程中，更加符合人类常识推理模式的非单调逻辑替代了单调逻辑称为了主流的工具。

1955年，人工智能奠基人之一的John McCarthy发起了达特茅斯会议，正式在会议上提出了“人工智能”这个概念，并在1959年发明了LISP语言，这是第一个广泛流行于人工智能研究工作中的高级语言。LISP最大的特点是它所计算和处理的是符号表达式而不是数。这为逻辑程序的诞生和发展提供了基础。1951年，Aflred Horn 提出了霍恩(Horn)子句，这是带有最多一个肯定文字的析取范式。霍恩子句是逻辑程序的重要构成基础[3]。上世纪70年代，Kowalski率先提出了逻辑可以作为程序设计语言的基本思想。不久逻辑程序设计正式诞生[4]。逻辑程序只需要设置求解问题需要符合的规则和加入问题所有的前置事实即可求解问题，而非传统的高级语言那样使用“顺序、控制、循环”等步骤来解决问题。逻辑程序就是：事实+ 规则= 结果。基于Kowalski提出的逻辑程序思想，法国的Colmerauer在1979年研发出世界上第一种用于逻辑程序设计的语言－－PROLOG（PROgram in LOGic）[5]。LSIP和PROLOG对知识表示的

发展起了十分深远的影响，这两种语言事实上为计算机科学中编程语言发展提供了新的方向：即与当下流行的高级语言不同，并不是基于过程控制去演算一个问题的解，而是通过程序员从逻辑出发，刻画出一个问题是什么即可，至于怎么实现则由系统完成，然后问题便可以被求解。很遗憾，这个目标至今也没有被实现[2]。但可以知道逻辑程序是一种更为贴合人类日常语言模式的编码方式。同时逻辑程序可以帮助我们在编程上从编写“怎么做”到“做什么”的转变[6]。

然而，早期的经典逻辑中，以命题逻辑和一阶逻辑为主，都是通过已知的事实推出结论，并不会因为已知事实的增加而使得之前的结论丧失其有效性，因此是单调的(monotonic)，知道的事实越多，结论就越多。但人类在正常认知的过程中，当前的认知并不一定是事实或真理，所以一旦得到新的修正认知，之前的结论就存在被否定的可能性和需要，由于新认知的加入不一定会让结论增多，所以这样的逻辑称为非单调逻辑(non-monotonic logic)[7]。就如这样一个例子：一个人相信树的叶子在冬天都会掉光，他看到的树也都是这样子；当他看到柏树的树叶在冬天没有掉光时，他不会认为柏树不是树，而是会否定以前的结论：并非所有树的叶子都会在冬天掉光。所以说非单调逻辑才更符合人类的日常认知模式。

1978年，Keith Clark提出了失败即否定（Negation as Failure）。此理论补全了逻辑程序中的否定问题的表达困难[8]。但彼时尚未能马上得到失败即否定的模型语义。Gelfond和Lifschitz在1988年提出稳定模型语义（stable model semantics）后，填补了非单调逻辑领域对失败即否定的解释[9]。基于Gelfond和Lifschitz的稳定模型语义，及其后的发展，在上世纪90年代前后，形成了一种新的逻辑程序设计方法，即回答集编程（Answer Set Programming，ASP）[10]。

1.2 国内外研究现状

ASP在发展过程中不断地被扩展，并且拥有了一系列高效的求解器。主要分为两大类。一类是基于DPLL算法的：DLV、smodels、clasp及其扩展claspD、clingo及其扩展iclingo；另一类是基于SAT求解方法的：ASSAT和cmodels。ASP领

域在过去20年里可谓发展迅速，求解器的速度有了很大的提升。由于求解器的提速，ASP得以被广泛应用在实际项目中。2001年，NASA决定在航天决策系统中使用ASP[11]；2003年时，Eiter和Lenoe把ASP应用在了行为决策中[12]；Soininen和Niemela则尝试了在产品配置上运用ASP，并取得了不错效果[13]。而国内也有不少关于ASP的应用尝试。2010年，华南理工的李鑫为了克服E-R模型不具有自动推理能力的缺陷，提出了一种利用ASP来表示E-R模型的方法[14]；2012年，华南师范大学的赖河蒗在Banks选举问题上使用了ASP取代启发式算法进行求解，得到了良好的结果[15]；中科大的吉建民老师及其团队长久以来都把ASP技术运用到机器人控制上[2]。

虽然ASP已经发展相对成熟，但近几年来，国内外对ASP的研究重心依旧围绕着ASP求解器的提速问题。Lifschitz和Turner在1994年提出了分割集（splitting set）的概念，以及相应的程序分割（program splitting）方法，即通过分割集把原逻辑程序分割为bottom和top两部分，并证明了原程序的回答集可以通过该两部分的回答集求解得到[16]。分割集的思想很快被应用到ASP领域的多个方向，同时，分割集和程序分割能为ASP求解器的提速带来新思路。

1.3 本文的工作及意义

Lifschitz和Turner提出的分割集和程序分割方法为ASP的理论扩展和ASP求解器提速都带来了帮助。然而Lifschitz和Turner所定义的分割集对于很多ASP程序而言只有空集和全部原子构造成的集合（全集）两种情况。分割集为空集或全集都无法进行程序分割，这样便会导致分割集变得毫无意义。本文基于这个缺陷，对Lifschitz和Turner的理论展开了研究，并对其进行了扩展和应用，具体的工作如下：

1. 首先对ASP领域的基础知识进行梳理总结。首先详细介绍说明ASP中的重要理论概念，如：失败即否定、稳定模型语义、极小模型等。同时，详细分析了Lifschitz和Turner的分割理论，并剖析其分割集的缺陷所在。

2. 在深入研究Lifschitz和Turner的理论后，对原来的分割集进行扩展，具体是提出了新的分割集，新分割集可以为关于原程序的任意原子集。

3. 此外，本文提出了一种可以基于分割集为任意原子集的新程序分割方法。这种新的程序分割方法把bottom和top从之前的简单互补关系进行了扩展。在计算top部分时会引入新原子辅助，但不会影响最终结果。

4. 同时，本文对新的程序分割方法进行了计算复杂性的分析，指出整个分割和求解过程中的主要耗时部分，并给出如何设计分割集可以帮助求解提速的结论，同时通过实验支持了使用新程序分割方法求解原程序的回答集可以带来显著的提速效果。

5. 在分析新的程序分割的计算复杂性时，得出了如果分割集为程序结论的话，可以大大降低复杂性。由此为思路，提出了通过程序结论去进行程序化简，为程序化简问题引入分割集的概念，并得到理想的效果。

分割集思想和理论自提出以来就被认为是研究回答集语义的一个良好工具[17]。Gebser在2008年以分割集理论作为基础实现了增量式ASP求解器－－iclingo[18]。此外，Oikarinen和Janhunen把分割集的思想拓展到了带嵌套表达式的逻辑程序中[19]，Ferraris则在稳定模型语义下的任意一阶逻辑中引入了分割集的使用[20]。分割集的思想对ASP领域有着重要的影响，所以本文将分割集扩展到任意原子集，并提出相应的新程序分割方法，让原来分割集的局限性得意打破，同时本文把新的分割集思想延伸到实际应用中－－程序化简。对ASP求解提速有莫大的意义。

1.4 本文的安排结构

本文首先介绍了人工智能的发展过程及回答集编程产生的背景，同时说明了回答集编程当前的发展情况，然后引出分割集和程序分割的概念，详细分析Lifschitz和Turner的分割集的局限性后，提出自己的新分割集和新程序分割方法，然后将其应用到程序化简中。此外，本文进行了两个实验。第一个为对比使

用程序分割方法求解原程序回答集和直接求解的效率，结果为使用程序分割的效率更好；第二个为在程序化简中引入分割集的概念，并比较求解化简后的程序跟原程序的效率对比，结果为化简后的求解更快。具体的章节安排如下：

第一章给出了本文的研究背景和现状，简要地阐述了从人工智能到回答集编程这个领域诞生的整个过程，指出了当前回答集编程的发展情况和发展方向——求解器提速。同时，指出了本文的主要工作和工作内容对回答集编程发展的意义。

第二章介绍了回答集编程的基础知识，主要讲及失败即否定和稳定模型语义，以及回答集编程自身的语义。此外，还会引入说明回答集编程的特性，如：环与环公式，分割集和程序分割的基本概念。

第三章详细地讲述本文提出的新分割集和新程序分割的概念和思想。同时会对程序化简过程的计算复杂性进行剖析，指出其中的主要耗时点，并给出提升的办法。

第四章展现了新分割集和新程序分割在回答集编程中的应用，并给出了这些应用的具体细节。

第五章根据第四章的应用设计相关实验，并通过实验数据分析新分割集和新程序分割的使用情况。

第六章是本文的收尾部分，给出了对全文的总结和对分割集思想的后续工作进行展望，列举出一些具体的可尝试方向。

第 2 章 预备知识

本章给出了本文将涉及的基础知识，先从命题逻辑进行介绍，引入失败即否定，并定义ASP中的普通规则，及规则中各个部分涉及的命名符号。然后给出“满足”的概念。完成基础介绍后，进入回答集语义，即如何定义ASP程序的回答集。然后列出一些ASP程序的主要性质，主要是环与环公式，及本文的核心内容分割集与程序分割。

2.1 命题逻辑

命题（**proposition**）是非真即假的陈述句。我们一般使用字母代表一个命题。命题逻辑由命题公式和一套证明规则所组成，其中命题公式和证明规则就是命题逻辑的运算本体和运算规则[21]。

定义 2.1： 命题逻辑的符号包括[22]：

- 命题符号：A，B，C，...，统称为$\aleph$
- 真值符号：$true$，$false$；
- 连接词：$\wedge$，$\vee$，$\neg$，$\rightarrow$，$\equiv$。

在ASP逻辑程序中，我们也把命题符号称为一个原子。

定义 2.2 (文字)： 一个命题符号或者一个命题符号的否定。其中一个命题符号为正文字，一个命题符号的否定为负文字。

命题的真假性被称为其真值（truth value），真值只有真（true）和假（false）两个。每个命题只能为真或者为假，不能既是真又是假，或者既不是真又不是假。通常地，在求解逻辑程序中，我们所得到的结果就是指能够被确定真值为真的命题集合。

在定义2.1中的连接词的作用是把若干个文字连接成命题公式。每个连接词都有自己的真值表，可以概括如下[22]:

1. 非（$\neg$），真假性与其操作的命题相反，当A为真时，$\neg A$为假，当A为假时，$\neg A$为真；

2. 合取（$\wedge$），$A \wedge B$为真，当且仅当命题A和B同时为真时；

3. 析取（$\vee$），$A \vee B$为真，当且仅当命题A和B中至少有一个为真；

4. 蕴涵（$\rightarrow$），$A \rightarrow B$为假，当且仅当命题A为真且B为假。

命题公式由一个文字或者多个文字通过连接词组成。命题公式是命题逻辑的推理基础。在有需要的情况下，命题公式可以通过对合律、德?摩根定律、结合律和分配律等运算性质得到逻辑等价的范式，本文涉及的范式有合取范式和析取范式。

定义 2.3 (范式)： 合取范式（$ConjunctiveNormalForm, CNF$）：一系列析取式的合取形式；析取范式（$DisjunctiveNormalForm, DNF$）：一系列合取式的析取形式；其中析取式（合取式）为若干文字只通过连接词$\vee$（）$\wedge$)进行连接。

2.2 逻辑程序

逻辑程序是知识表示的基础。早期研究人工智能的学者都详细可以找到一种以符号刻画知识和推理过程的方法以达到模拟人类大脑推理的过程[2]。而事实上，逻辑程序便是这样的一种手段。逻辑程序的形式有很多。而在ASP逻辑程序中，主要用及的便是引入失败即否定和经典否定的命题公式。

ASP程序一般分为两部分：事实集（Facts）和ASP逻辑程序。要求解一个ASP程序的回答集，需要结合使用事实集和ASP逻辑程序。

ASP程序的求解过程是：

1. 使用例化工具（常用的有gringo和lparse）通过事实集例化ASP逻辑程序；

2. 对例化后的逻辑程序，调用求解器进行求解。

从实际效果来看，例化后的逻辑程序就是不含变量的命题公式集合。本文中只考虑完全例化后的长度有限的逻辑程序。本文所探讨的ASP逻辑程序由有限个规则（rule）所组成，其中规则的形式如下：

$$a_1 \vee a_2 \vee ... \vee a_k \ \leftarrow \ a_{k+1}, ..., a_m, \ not \ a_{m+1}, ..., \ not \ a_n. \tag{2.1}$$

其中$n \geq m \geq k \geq 1$，a_i为原子（atom），not代表失败即否定。在k，m，n取不同范围的值时，形式(2.1)所表示的规则有不同的意义，具体如下：

1. $k = 0$时，规则被称为限制（Constraint）；
2. $k = 1$时，规则被称为正规规则（Normal Rule）；
3. $n = m = 0$时，规则被称为事实（Fact）；
4. $n = m$时，即规则中没有带not的原子时，规则称其为正规则（Positive Rule）。

例 2.1： 下面为四种形式的规则：

$$\leftarrow \ p, \ q. \tag{2.2}$$

$$r \ \leftarrow \ s, \ not \ t. \tag{2.3}$$

$$s. \tag{2.4}$$

$$p \ \leftarrow \ r, \ t. \tag{2.5}$$

$$s \ \vee \ p \ \leftarrow \ r, \ t, \ not \ p. \tag{2.6}$$

其中的(2.2)为限制，(2.3)为正规规则，(2.4)为事实，(2.5)为正规则，(2.6)为一般规则。

定义 2.4 (正规逻辑程序（$NormalLogicProgram, NLP$）)： 由有限条正规规则组成的逻辑程序，其中可以包含有限个事实及限制。

定义 2.5 (析取逻辑程序（$DisjunctiveLogicProgram, DLP$）)： 由有限条形如形式(1)的规则组成的逻辑程序，其中可以包含有限个事实及限制。

对于形式(2.1)中的规则，我们还可以将其等价于以下形式：

$$head(r) \leftarrow body(r) \tag{2.7}$$

其中r代表形式(2.1)中的整条规则，$head(r)$称为规则r的头部，$body(r)$称为规则r的体部。具体有$head(r)$为$a_1 \vee a_2 \vee ... \vee a_k$，$body(r)$为$a_{k+1}$, ... , a_m, $not\ a_{m+1}$, ... , $not\ a_n$。其中ASP程序的头部中的原子连接关系只为析取，体部中的原子连接关系只为合取。更进一步的划分有$body(r) = body^+(r) \wedge body^-(r)$，其中$body^+(r)$为$a_{k+1}$, ... , a_m，即不带not的，$body^-(r)$为$not\ a_{m+1}$, ... , $not\ a_n$，即带有not的。在某些情景下上述定义的命题公式可以被作为原子集（a set of atoms）进行讨论。

从集合意义出发，引入以下常用的集合概念：

定义 2.6： 规则中的集合，以形式(2.1)中的规则作为例子：

- $head(r) = \{a_1, a_2, ..., a_k\}$；
- $body(r) = \{a_{k+1}, ..., a_m, ..., \neg\ a_{m+1}, ..., \neg\ a_n\}$，一个文字集，把体部中的$not$替换为$\neg$而得到；
- $body^+(r) = \{a_{k+1}, ..., a_m\}$，体部正文字的原子组成的原子集；
- $body^-(r) = \{a_{m+1}, ..., a_n\}$，体部负文字的原子组成的原子集；
- $Atoms(r) = head(r) \cup body^+(r) \cup body^-(r)$，即规则r中的原子全集。

给定一个逻辑程序P，P就是一个规则集合。这里定义Atoms(P)即逻辑程序P中所有出现过的原子。

定义 2.7： 逻辑程序P的原子集为：

$$Atoms(P) = \bigcup_{r \in P} Atoms(r) \tag{2.8}$$

例 2.2： 把例子2.1中的规则集合看作是一个逻辑程序P，则有：

对于(2.6)，$head(r) = \{s, q\}$，$body(r) = \{r, t, p\}$，$body^+(r) = \{r, t\}$，$body^-(r) = \{p\}$，$Atoms(r) = \{s, q, r, t, p\}$。

同时，$Atoms(P) = \{s, q, r, t, p\}$。

在了解了逻辑程序中涉及的常用概念后，我们将通过满足（satisfy）性来定义逻辑程序的模型。

定义 2.8： 一个原子集S满足一个正合取范式R，当且仅当$Atoms(R) \subseteq S$；一个原子集S满足一个正析取范式R，当且仅当$Atoms(R) \cap S \neq \emptyset$。满足符号为$\models$。

我们把满足的概念放在逻辑程序及其规则中，由于逻辑程序中的头部中的文字为析取关系，且仅有正文字，体部中的文字为合取关系。则相应地定义一个原子集S满足一个规则r的$body(r)$当且仅当$body^+(r) \subseteq S$且$body^-(r) \cap S = \emptyset$，记为$S \models body(r)$。一个原子集$S$满足一个规则$r$的$head(r)$当且仅当$head(r) \cap S \neq \emptyset$，记为$S \models head(r)$。

例 2.3： 给定一个原子集$S = \{a, b, c\}$，规则$r1$为$a \lor b \leftarrow c, not\ d.$，规则$r2$为$d \lor e \leftarrow c, not\ b.$，则有：$S \models head(r1)$，$S \models body(r1)$，$S \not\models head(r2)$，$S \not\models body(r2)$。

接下来，我们定义如何判断一个原子集是否满足一个规则，以及一个原子集是否满足一个逻辑程序。

定义 2.9： 一个原子集S满足一个规则r，当且仅当$S \models body(r)$蕴涵$S \models head(r)$，并记为$S \models r$。一个原子集S满足一个逻辑程序P，当且仅当S满足P中的所有规则。

定义 2.10： 如果一个原子集S满足一个逻辑程序P，我们将S称为P的一个模型（$model$）。若一个原子集I是逻辑程序P的模型，且不存在另一个原子

集J符合$J \subseteq I$且J是逻辑程序P的模型，则称原子集I是逻辑程序P的极小模型（$minimal\ model$）。

命题 2.11： 一个不包含not（失败即否定）的逻辑程序，称为正逻辑程序。正逻辑程序有且仅有一个极小模型。

在了解了上述的逻辑程序基础后，我们接下来将引入回答集编程的语法和语义的相关基础知识。

2.3 回答集编程

2.3.1 回答集编程基础

回答集编程时一种声明式编程，它主要应用于极为困难的搜索问题当中。Lifschitz认为在非单调推理的知识表示领域中，回答集编程在知识密集型的应用场景中尤其重要和高效[23]。ASP语法结构与Prolog类似，但其计算机制不同，主要借助于高效的命题逻辑可满足求解器进行计算[10]。本节将讲述ASP逻辑程序的语义和主要性质，如环和环公式等。

经典逻辑程序一般情况下只能推出正文字的结论，而实际情况下，我们也是需要负文字结论的。Reiter提出的封闭世界假定[24]和Clark的失败即否定以类似的思想为逻辑程序可以推出负文字形式的结论带来了支持。

失败即否定的基本思想就是：无法证明一个命题为真，则判定其为假。这也是最为自然的一种逻辑假定。在ASP逻辑程序中一些规则的体部会出现“$not\ A$”这样的文字，而对于这个文字，我们可以直观地把它看作是命题:“不能确定A为真”，所以如果ASP逻辑程序中不能推出A为真，则可以推出$not\ A$为真。失败即否定中这种推理过程其实是一种倾向否定的假定逻辑。即使当前推出$not\ A$为真，那是基于暂时无法推出A为真。一旦加入新的规则，可以推出A为真，那么原来的$not\ A$则被认为是假。这一点恰恰是非单调逻辑的本质所在：加入新的结论时可能会推翻之前结论的有效性。

在失败即否定之外，ASP逻辑程序中还引入了对经典否定。不过这里说的引

入是操作层面上的非语法上的[2]。ASP逻辑程序中只通过失败即否定的not来表示负文字。对于经典否定，即经典逻辑下的非，ASP逻辑程序通过加入新原子和约束来引入经典否定。

如：为了表示原子p的经典否定，可以定义新原子$p_{_}$，并向原逻辑程序中加入约束"$\leftarrow\ p, p_{_}.$"，保证了p和$p_{_}$不能同时为真。这样就将含有经典否定的逻辑程序转化为不含经典否定的逻辑程序。

非单调逻辑最初是为了解决框架问题（$frame\ problem$）和缺省规则（$default\ rule$）的推理[2]。在非单调逻辑的发展过程中，自上世纪七十年代起，主要出现了缺省逻辑（$Default\ Logic$）、自认知逻辑（$Auto-epistemic\ Logic$）和限定理论（$Circumscription$）[25]。然而，这些理论由于缺乏高效的开发工具和不具备模块化程序设计并没有得到广泛应用[10]。而一直到了1988年，Gelfond和Lifschitz提出了稳定模型语义（$Stable\ Model\ Semantic$）。稳定模型语义帮助解决了非单调推理无法解释失败即否定的问题。基于稳定模型语义进行的研究和扩展，发展出回答集编程这个领域。

回答集编程所涉及的语法即2.2节中介绍的逻辑程序的内容。现在，我们来介绍回答集编程的语义和主要属性，从ASP逻辑程序的回答集开始。

2.3.2 回答集编程的语义

Gelfond和Lifschitz提出了稳定模型语义（$Stable\ Model\ Semantic$）时，也给出了一个规约方法，以化简一个ASP逻辑程序中的失败即否定。

定义 2.12 (G-L规约)： 给定一个原子集S和逻辑程序P，P基于S的G-L规约记为P^S。逻辑程序P通过以下两个化简规则得到P^S：

- 若一个规则的体部中有$not\ p$，且$p \in S$，则删掉该规则；
- 对剩下的所有规则，删除体部中的$not\ p$，p为$Atoms(P)$中任意一个原子。

例 2.4： 已知原子集$S = \{a, b, c\}$，且逻辑程序P如下：

$$a \leftarrow b,\ c. \tag{2.9}$$

$$e \leftarrow b,\ not\, a. \tag{2.10}$$

$$f \leftarrow not\, e. \tag{2.11}$$

根据G-L规约的规则，P中第二条规则的负文字中包含S里的原子，所以直接删掉；P中第三条规则的负文字中包含S以外的原子，所以只把负文字删掉。

所以P^S为：

$$a \leftarrow b,\ c. \tag{2.12}$$

$$f. \tag{2.13}$$

显然，通过G-L规约进行化简后得到的新逻辑程序P^S是一个不包含任何失败即否定的逻辑程序。这样的逻辑程序只有一个唯一的极小的模型，我们称这个模型为稳定模型（$stable\, model$），并记其为$\Gamma(P^S)$。

定理 2.1： 对于一个不含约束的正规逻辑程序（$NLP\, Without\, Constraints$）$P$和一个原子集$S$，$S$是$P$的一个回答集（$Answer\ Set$）当前仅当$S = \Gamma(P^S)$。

对于包含有约束的逻辑程序，我们依旧可以通过G-L规约来定义其回答集，通过对程序中的约束和一般规则分离式判断即可。

定理 2.2： 对于一个包含约束的正规逻辑程序（$NLP\, With\, Constraints$）P和一个原子集S，S是P的一个回答集（$Answer\ Set$）当前仅当$S = \Gamma(PD^S)$，且S满足P中的所有约束。其中PD是P去掉所有约束后所得到的逻辑程序。

Gefond和Lifschitz在1991年对析取逻辑程序（DLP）的回答集定义进行了补充。关于析取逻辑程序的回答集，依旧可以通过G-L规约得到。对一个析取逻辑程序P进行G-L规约后得到的程序记为P^S，P^S里不再含有not，然而不同于正规逻辑程序（NLP），该逻辑程序将有一系列集合意义上的极小模型，这里记为$\Psi(P^S)$。若一个原子集S是$\Gamma(P^S)$中的元素，则S是逻辑程序P的回答集[26]。

定理 2.3： 给定一个析取逻辑程序P和一个原子集S，S是P的一个回答集（$Answer\ Set$）当前仅当$S \in \Psi(P^S)$。

在介绍了ASP逻辑程序的回答集后，接下来将说明ASP逻辑程序中的一个重要性质，即ASP逻辑程序中的环以及对应的环公式。

2.3.3 回答集编程的主要性质：环与环公式

Lin和Zhao则在2002年时给出了正规逻辑程序中的环，同时使用命题公式基于环定义了其环公式[27]。Lee和Lifschitz在2003年时给出了析取逻辑程序中的环的概念[28]。

在给出具体的环及环公式定义前，需要先给出一个逻辑程序的正依赖图（$Positive\ Dependence\ Graph$）的定义。

定义 2.13 **(正依赖图)**： 已知一个逻辑程序P，以P中的原子作为顶点，规则作为构成边的依据，可以构造出一个有向连通图，称其为逻辑程序P的正依赖图，记为G_P。其中，当存在P中的一个规则形如$p \in head(r)$且$q \in body^+(r)$，则正依赖图中存在一条从原子p指向原子q的有向边。

例 2.5： 给定逻辑程序P如下：

$$a \leftarrow not\ d. \tag{2.14}$$

$$d \leftarrow not\ c. \tag{2.15}$$

$$a \leftarrow c,\ d. \tag{2.16}$$

$$c \leftarrow a. \tag{2.17}$$

根据定义，关于逻辑程序P的正依赖图如下：

有了正依赖图后，我们引入基于正依赖图拓扑结构所定义的环。

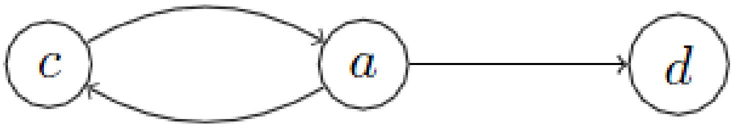


图 2.1: P的正依赖图

定义 2.14 (环[27])： 给定一个逻辑程序P，对于$Atoms(P)$的任意非空子集L，如果对于L中的任意两个原子p、q，G_P中都有至少一条长度大于0的路径使得p可达q，则称L是P的一个环（$Loop$）。此外，任意单原子集合也属于一个环。

逻辑程序的环实质上就是其对应的正依赖图中的强连通分量。逻辑程序的环还有一个重要性质用于定义环公式，那就是环的外部支持规则（$External\ Support\ Rules$）。环的外部支持是一个规则集合。环L在逻辑程序P中的外部支持规则用集合符号定义如下：

$$R^{-}(L,P) \;=\; \{r \in P \mid head(r) \cap L \neq \emptyset,\ body^{+}(r) \cap L = \emptyset\} \tag{2.18}$$

还可以更进一步地定义环L在逻辑程序P中基于原子集X的外部支持规则，用集合符号定义如下：

$$R^{-}(L,P,X) \;=\; \{r \in R^{-}(L,P) \mid X \models body(r) \wedge \bigwedge_{q \in head(r)L} \neg q\} \tag{2.19}$$

定义 2.15 (环公式[27])： 给定逻辑程序P，对于其中的任意一个环L，$R^{-}(L,P)$为L在P中的外部支持规则，L对应的环公式记为$LF(L,P)$，定义如下：

$$\bigwedge_{p \in L} p \supset \bigvee_{r \in R^{-}(L,P)} (body(r) \wedge \bigwedge_{q \in head(r)L} \neg q) \tag{2.20}$$

命题 2.16： [28] 给定一个逻辑程序P和一个原子集合S。如果S满足P，则以下说法是等价的：

- S是P的一个回答集；

- S满足P中所有环L的环公式$LF(L,P)$；
- S满足$Atoms(P)$所有非空子集E的环公式$LF(E,P)$。

研究ASP逻辑程序中的环，是很有意义的。在ASP逻辑程序中，使用SAT求解器得到的模型不完全是ASP逻辑程序的回答集。其根本原因就在于环的存在。如果ASP逻辑程序中存在环，即存在环内所有原子相互推导的行为，即环内的原子相互推导对方为真，这样就会产生大量模型。然而，从回答集语义出发，这其中如果没有事实或者失败即否定的支持，推导不成立，所以这些模型不是回答集。而环公式就是提取正体部原子全在环外的规则以支持环内原子的成立，这也是外部支持规则的名字由来。基于这样的事实，Lin和Zhao提出了使用环公式求解ASP逻辑程序回答集的方法，并实现了ASP求解器－－ASSAT。

在介绍Lin和Zhao的结论之前，先引入逻辑程序的完备（$Completion$）。逻辑程序的完备的具体定义如下：

定义 2.17 (完备[27])： 给定一个逻辑程序P，其完备（$Completion$），记为$Comp(P)$。$Comp(P)$是以下规则的集合：

- 对于任意$p\in Atoms(P)$，P中所有以p作为头部的规则形如$p\leftarrow G_k$，则$p\equiv G_1\vee G_2\vee ...\vee G_n$为$Comp(P)$中的元素，特别地，如果一个原子$p$没有作为头部出现过，那么把$\neg p$加入到$Comp(P)$中；
- 对于P中的所有限制，形如$\leftarrow G_k$，把$\neg G_k$加入到$Comp(P)$中。

例 2.6： 有以下逻辑程序P：

$$a\ \leftarrow\ b,\ c,\ not\ d. \tag{2.21}$$

$$a\ \leftarrow\ b,\ not\ c,\ not\ d. \tag{2.22}$$

$$\leftarrow\ b,\ c,\ not\ d. \tag{2.23}$$

则其完备$Comp(P)$为：

$$\{a \equiv (b \wedge c \wedge \neg d) \vee (b \wedge \neg c \wedge \neg d),\ \neg b,\ \neg c,\ \neg d,\ \neg(b \wedge c \wedge \neg d)\} \tag{2.24}$$

在有了逻辑程序完备的概念后，我们引入一个新的回答集求解方法。

定理 2.4： [27] 给定逻辑程序P，记$Comp(P)$为其完备，LF表示P中所有环的环公式集合。一个原子集S是逻辑程序P的回答集，当且仅当它是$Comp(P) \cup LF$的一个模型。

Lin和Zhao所提出的环及环公式在后续至今的时间被不断使用。Gebser在2010年时提出了基本环（$Elementary\ Loop$）[29]，其是一系列可以代表其他环的环，Gebser还将基本环应用到clasp求解器中。2014年，Ji和Wan等在Gebser的基本环的基础上更进一步地在正规逻辑程序中提出了更具有代表性的特征环（$Proper\ Loop$）[30]，并在不久后将特征环拓展到析取逻辑程序[31]。

2.4 分割集与程序分割

Lifschitz和Turner在1994年时便提出了分割集（$Splitting\ Set$）的概念。其本意是为了把逻辑程序划分成若干个较小规模的程序，然后通过这些小规模的程序的回答集去求解原程序的回答集。由于逻辑程序的规模对求解效率有很大的影响，通过把逻辑程序的规模降低，把指数性的关系降为加性关系，将大大有助于提高逻辑程序的求解效率。但ASP逻辑程序是非单调的，要达到分割程序后得到的回答集是原程序的回答集，需要满足一定的条件[2]。

Lifschitz和Turner在给出分割集的概念后，也基于分割集提出了相应的程序分割（$Program\ Splitting$）方法。程序分割方法具体就是根据定义找出分割集，通过分割集把原程序分割为底部（$bottom$）和顶部（top）两部分，并且证明原程序的回答集可以通过$bottom$和top两部分的回答集所求得。

定义 2.18 (分割集[16])： 给定一个逻辑程序P，其分割集（$Splitting\ Set$）是一个原子集，标记为U，分割集U需要满足：对于任意的规则$r \in P$，$head(r) \cap U \neq \emptyset$蕴涵$Atoms(r) \subseteq U$。

直观地，分割集中原子对应为逻辑程序P的正依赖图中出度为零的顶点[2]。基于分割集，可以把原程序划分成$bottom$和top两部分，具体定义如下：

定义 2.19 (底部和顶部[16])： 给定一个逻辑程序P，及其分割集U，标记P的底部为$b_U(P)$，顶部为$t_U(P)$，使用集合符号定义为：

$$b_U(P) = \{r \in P \mid head(r) \cap U \neq \emptyset\} \tag{2.25}$$

$$t_U(P) = P \setminus b_U(P) \tag{2.26}$$

其中"$\setminus$"为集合减。

明显地，逻辑程序的$bottom$是把分割集U相关的规则都抽取出来，而top则是头部与分割集U无关的规则集合。$\emptyset$和$Atoms(P)$是任意一个逻辑程序P的分割集[2]。

为了求解原程序的回答集，需要引入化简操作，Lifschitz和Turner定义了操作$e_U(P, X)$来联合$bottom$的回答集和top求出另一部分的回答集。

定义 2.20： 给定逻辑程序P，X和U为原子集合，定义操作$e_U(P, X)$如下：

- 删除符合以下条件的规则r：$head(r) \cap X \neq \emptyset$且$body^+(r) \cap U \not\subseteq X$，或者$(body^-(r) \cap U) \cap X \neq \emptyset$；
- 在剩下的规则的体部中把所有形如a或$not\ a$，其中$a \in U$的文字删掉。

定义 2.21： 给定逻辑程序P，及其分割集U，P关于U的一个方案（$Solution$）是一个原子集组合$\langle X, Y\rangle$，具体如下：

- X是$b_U(P)$的回答集；
- Y是$e_U(P\setminus b_U(P), X)$的回答集。

例 2.7： 考虑逻辑程序P：

$$a \ \leftarrow \ not\ d. \tag{2.27}$$

$$d \leftarrow not\ c. \tag{2.28}$$

$$a \leftarrow c,\ d. \tag{2.29}$$

$$c \leftarrow . \tag{2.30}$$

根据定义可知$U = \{c, d\}$是P的一个分割集，并且可以计算得到$b_U(P) = \{d \leftarrow not\ c.\ c \leftarrow .\}$，而$P \backslash b_U(P) = \{a \leftarrow not\ d.\ a \leftarrow c,\ d.\}$，$\{c\}$是$b_U(P)$的回答集，故令$X = \{c\}$，有$e_U(P \backslash b_U(P), X) = \{a \leftarrow .\}$，其回答集为$\{a\}$。$\langle\{c\}, \{a\}\rangle$则是$P$关于$U$的一个方案，于本例也是唯一的方案。

关于原程序的回答集，可以通过上述定义的逻辑程序关于分割集的方案得到，Lifschitz和Turner给出了分割集理论就是证明了如果从方案得到回答集。

定理 2.5 (分割集理论[16])： 已知逻辑程序P和其分割集U，则一个原子集S是逻辑程序P的回答集，当且仅当$S = X \cup Y$，其中$\langle X, Y\rangle$是P关于U的一个方案。

关于回答集编程的基础知识和分割集理论的主要知识介绍到此。接下来将进入本文的主要内容，介绍新的分割集和新的程序分割方法，以及通过具体的应用场景来体现分割集的意义。

第 3 章　新分割集与程序分割

本章基于对Lifschitz和Turner提出的分割集和程序分割理论的分析，提出了新的分割集和程序分割方法。实际上是提出了一个可以对任意原子集作为分割集都有效的程序分割方法。本章将分别给出正规逻辑程序和析取逻辑程序的新程序分割方法，然后提出一个强程序分割方法。最后以正规逻辑程序的程序分割方法为例，对新的程序分割方法的计算复杂性进行分析，并指出主要性能瓶颈所在及改进思路。

3.1　新分割集

在给出新分割集之前，本文将对Lifschitz和Turner提出的分割集理论进行分析，然后基于他们对分割集的定义提出一个计算分割集的算法，并对ASP竞赛中的程序进行计算，对计算结果进行分析，随后给出本文定义的新分割集。

3.1.1　原有分割集和程序分割的分析证明

Lifschitz和Turner给出的分割集定义是：一个原子集U称为一个逻辑程序P的分割集，当且仅当对P中的所有规则r都有$head(r) \cap U \neq \emptyset$蕴涵$Atoms(r) \subseteq U$。这个分割集定义的直观含义就是一个分割集若包含规则头部的原子，则也可以包含该规则的所有原子。这样的性质保证了分割集U可以把原程序从结构上划分成两部分，进而保证了原程序的回答集可以从分割后的两部分的回答集求解得到。Lifschitz和Turner在1994年的原文中只给出了程序分割方法的定义，即根据定义(2.21)分别求出底部$b_U(P)$和化简后的顶部$e_U(t_U(P), X)$的回答集X和Y，根据方案$\langle X, Y \langle$得到原程序的回答集。本文在这里补充给出这种分割方法的可行性证明。关于Lifschitz和Turner的程序分割方法可行性的证明如下。

证明： 首先证明分割集可以把逻辑程序划分为回答集互斥的两部分，并且底部的回答集是原逻辑程序回答集子集。

根据定义2.18，分割集U满足对于逻辑程序P中任意的规则r都有，$head(r) \cap U \neq \emptyset$蕴涵$Atoms(r) \subseteq U$。而根据定义2.19有：

$$b_U(P) = \{r \in P \mid head(r) \cap U \neq \emptyset\} \tag{3.1}$$

$$t_U(P) = P \setminus b_U(P) \tag{3.2}$$

根据分割集U的特性，可以知道：$Atoms(b_U(P)) \subseteq U$，并定义：

$$head(t_U(P)) \;=\; \bigcup_{r \in t_U(P)} head(r) \tag{3.3}$$

$$body(t_U(P)) \;=\; \bigcup_{r \in t_U(P)} body(r) \tag{3.4}$$

另，记一个逻辑程序P的回答集为$\Gamma(P)$，而回答集必是一个逻辑程序的头部原子的子集，即$\Gamma(P) \subseteq head(P)$。根据$t_U(P)$的定义可以知道：

$$head(t_U(P)) \subseteq Atoms(P) \backslash U \tag{3.5}$$

$$\Gamma(b_U(P)) \subseteq U \tag{3.6}$$

由于$head(t_U(P)) \cap U = \emptyset$，所以$\Gamma(t_U(P)) \cap U = \emptyset$，故有$\Gamma(b_U(P)) \cap \Gamma(t_U(P)) = \emptyset$，这样保证了$b_U(P)$和$t_U(P)$的回答集是互斥的。此外，我们已知：

$$Atoms(b_U(P)) \subseteq U \tag{3.7}$$

$$head(t_U(P)) \subseteq Atoms(P) \backslash U \tag{3.8}$$

$$head(b_U(P)) \cap head(t_U(P)) = \emptyset \tag{3.9}$$

这些关系表明$b_U(P)$于原逻辑程序而言，是一个独立的模块，其回答集只能从$b_U(P)$这部分推出，因为$head(b_U(P))$中的原子不会出现在$head(t_U(P))$中，即$\Gamma(b_U(P))$中的元素不需要依赖$t_U(P)$中的规则，一旦$head(b_U(P))$中的某个元素被推出为真，那么放在整个程序它都将是真的，即它必为原程序回答集的元素，即$\Gamma(b_U(P))$中的元素必将出现在原程序的某个回答集中。

然而，并不是直接计算$b_U(P)$和$t_U(P)$的回答集就可以得到最终的回答集。因为$body(t_U(P))$中可能会包含有U中的原子，而如果这些原子是在$\Gamma(b_U(P))$中的

话，我们是可以知道其真值的，因为$\Gamma(b_U(P))$必是原程序回答集的一部分。那么需要定义一个操作来删除这些可以肯定真值的原子。这个操作就是定义2.20中的$e_U(P,X)$。

接着证明$e_U(P,X)$的有效性。

令$X=\Gamma(b_U(P))$。对于X中的原子，其在$t_U(P)$中的形式只有正文字和负文字，即x或$not\ x$，$x \in X$。

在$e_U(P,X)$的定义中，它只保留满足以下条件的规则r：$body^+(r) \cap U \subseteq X$，及$(body^-(r) \cap U) \cap X = \emptyset$。并在保留下来的规则中删去所有形如$a$或$not\ a$，其中$a \in U$的文字。

1. 对于满足$body^+(r) \cap U \subseteq X$的规则$r$，后续的化简操作是$body^+(r) \backslash U$，而$X \subseteq U$，且$body^+(r) \cap U \subseteq X$，所以实质的意义就是$body^+(r) \backslash X$。对于$x \in body^+(r) \cap X$，因为$x \in X$，$X=\Gamma(b_U(P))$，并且$X$必为原程序回答集的一部分，所以$x$也为原逻辑程序中某个回答集的元素，所以可以确定$x$为真，且体部为合取关系，根据$head(t_U(P)) \subseteq Atoms(P) \backslash U$，即$x$在$t_U(P)$中不会在出现在规则的头部，即在$t_U(P)$中不可能推出$x$为真，但从$X$可以确定其为真，故需要删去$x$。所以保留满足$body^+(r) \cap U \subseteq U$的规则$r$，并执行$body^+(r) \backslash U$操作。

2. 对于满足$(body^-(r) \cap U) \cap X = \emptyset$的规则$r$，后续的化简操作是$body^-(r) \backslash U$。反向考虑，如果$(body^-(r) \cap U) \cap X \neq \emptyset$，即规则$r$的体部中包含$not\ a$，$a \in X$。而从上面的说明可以知道$a \in X$，则可以判定$a$为真，即$not\ a$为假（因为“推不出$a$为真”为假）。而体部为合取关系，一旦确定体部中有为假的元素，则体部为假，该规则无法推出头部为真，所以可以删去该规则，故只考虑满足$(body^-(r) \cap U) \cap X = \emptyset$的规则。此外，对于满足这个条件的规则，其$body^-(r) \cap X = \emptyset$，而对于$body^-(r)$可能存在$U \backslash X$中的元素，$U \backslash X$中的元素直接含义就是原程序回答集中推不出为真的原子，所以若$b \in U \backslash X$，则$not\ b$为真。跟1中一样，体部为真的元素应该直接删去。所以保留满

足$(body^-(r)\cap U)\cap X=\emptyset$的规则$r$，并执行$body^-(r)\backslash U$的操作。

根据上面的证明可以知道，X作为原程序回答集的一部分，$e_U(t_U(P),X)$利用X中元素的真值化简出一个计算原程序剩下回答集部分的逻辑程序。记$Y=\Gamma(e_U(t_U(P),X))$。

$b_U(P)$中只包含U中的原子，它自身是一个命题闭包，所以求解得到的回答集X必是原程序回答集的部分。而$e_U(t_U(P),X)$得到一个跟U无关的程序，同时确保了X中的元素为真，所以这部分求解得到的回答集Y就是原程序回答集剩下的部分。且有

$$X\subseteq Atoms(P)\cap U \tag{3.10}$$

$$Y\subseteq Atoms(P)\backslash U \tag{3.11}$$

所以$X\cap Y=\emptyset$，故$X\cup Y$是确保一致的，并且为原程序的回答集。 ■

上述证明过程说明了Lifschitz和Turner提出的程序分割方法的有效性。由于U是一个命题闭包，它能有效地把一个逻辑程序的规则根据自己的闭包性在结构上一分为二。

3.1.2 提出新的分割集

Lifschitz和Turner给出的分割集的定义实际上是一个验证型定义。他们定义一个原子集U是逻辑程序P的分割集，当且仅当P中的每个规则r满足$head(r)\cap U\neq\emptyset$蕴涵$Atoms(r)\subseteq U$。这个定义从直观上来说就是如果一个原子集合是一个逻辑程序的分割集，那么它满足逻辑程序中凡是头部与其有交集的规则，都会有该规则的所有原子都在该原子集合内。本文根据Lifschitz和Turner对分割集的定义给出了一个计算逻辑程序的分割集的算法，为$Alogrithm1$。

明显地，$\emptyset$和$Atoms(P)$都是任何逻辑程序的分割集。而本文把上述算法应用在ASP竞赛的逻辑程序中，得到的结果为大部分逻辑程序的分割集都是$Atoms(P)$。这样的事实说明，Lifschitz和Turner所定义的分割集从理论上是美

Algorithm 1: 计算分割集的算法$U(P)$

输入: 一个逻辑程序P

输出: 分割集U

$U := \emptyset$;

$r := U$中的第一个规则;

$U := U \cup Atoms(r)$;

$a := U$中的第一个原子;

while U *is changed* **do**

  $rules := \{r \in P | head(r) \cap a \neq \emptyset\}$

  for $rule \in rules$ **do**

    $U := U \cup Atoms(rule)$

  end

  $a := U$中的下一个原子;

end

return U

好的，它是一个从原程序中抽取出来的命题闭包，直观来看，这样的一个命题闭包就是逻辑程序正依赖图中没有出边的子图，这样的子图不受其他命题作为外部支持，所以在求解回答集上也自封闭，具有良好的独立性。然而从ASP竞赛的逻辑程序的计算结果可以知道，实际中的程序并非都具有如此性能优良的子图。为了能让分割集的思想能应用到一般的情景下，本文对其Lifschitz和Turner的分割集理论进行了扩展。事实上，本文并没有定义一个新的分割集，而是定义了一个新的程序分割方法，并确保这个新的程序分割方法能对任意原子集构成的分割集都有效。即新的分割集被扩展为任意原子集，摆脱了原有分割集对逻辑程序的拓扑结构的强依赖性。

3.2 新程序分割方法

本节中将分别给出正规逻辑程序和析取逻辑程序的程序分割方法，这两者间的主要不同在于对顶部（*top*）的定义，这也由正规逻辑程序和析取逻辑程序的回答集语义所决定。在介绍正规逻辑程序的程序分割方法过程中，本节会在定义操作符的同时给出其直观上的含义，并证明新的程序分割方法对任意原子集构成的分割集的有效性。然后，本节会提出一个强程序分割方法，该方法是为了补充把

程序分割方法在分配律下的有效性。

3.2.1 正规逻辑程序分割方法

Lifschitz和Turner的程序分割方法是基于分割集进行的，而根据3.1.2的结果表明，对于大部分逻辑程序，分割集往往就是$Atoms(P)$。这样的分割集无法分割逻辑程序，所以本文提出了新的程序分割方法，以支持分割集可以为任意原子集。首先，本文继续使用定义2.19中的$b_U(P)$和定义2.20中的操作$e_U(P, X)$，依旧记分割集为U。而此时分割集U并不存在命题封闭性，即：

$$Atoms(b_U(P)) \subseteq U \tag{3.12}$$

不一定成立。可以更为直接地说，大部分情况下都不再成立。为了保证新程序分割方法的普遍适用性，我们考虑：

$$Atoms(b_U(P)) \nsubseteq U \tag{3.13}$$

事实上此时有效的方法对$Atoms(b_U(P)) \subseteq U$情况下一样有效，因为附加操作都是为了保证最坏情况的。

在原来的分割集下，$b_U(P)$内保证了命题封闭性，即有：

$$Atoms(b_U(P)) \cap head(P \backslash b_U(P)) = \emptyset \tag{3.14}$$

即如果$Atoms(b_U(P))$中的原子属于回答集，那么它只会在$b_U(P)$中被推出。而对于U为任意原子集的情况下，有：

$$Atoms(b_U(P)) \backslash U \neq \emptyset \tag{3.15}$$

所以需要考虑$Atoms(b_U(P)) \backslash U$这部分原子的真值可能性问题。而事实上，这部分原子的真值仅靠$b_U(P)$是无法确定的，因为它们可能是$P \backslash b_U(P)$中某个规则的头部，即可能会在$P \backslash b_U(P)$部分被推出。所以最直接也是最有效的方法就是在$\Gamma(b_U(P))$中加入这些不确定原子真值的全排列。然而，这些原子的真值也并非可以任意猜测，需要配合$b_U(P)$中的逻辑关系，所以本文定义了规则集合，来确保这些原子的真值在合理推导下被遍历。

定义 3.1： 规则集合$EC_U(P)$为$Atoms(b_U(P))\backslash U$中的原子的真值可能性提供补充规则，有：

$$EC_U(P) \; = \; \{p \; \leftarrow \; not\, p'.\, p' \; \leftarrow \; not\, p. \mid p \in Atoms(b_U(P))\backslash U\} \tag{3.16}$$

$EC_U(P)$为$Atoms(b_U(P))\backslash U$中的原子引入一组规则：

$$p \; \leftarrow \; not\, p'.\, p' \; \leftarrow \; not\, p. \tag{3.17}$$

其中的p'其实代表的就是$\neg p$，上面一组规则在ASP逻辑程序中的含义是$p \vee not\, p.$，换句话说即，如果一个ASP逻辑程序只有这两个规则，那么它的回答集就是$\{\{p\},\{p'\}\}$，实际就是$\neg p$或p。接着，通过一个简单的例子来展现$EC_U(P)$的效果。

例 3.1： 给定逻辑程序P_1：

$$a \; \leftarrow \; not\, d. \tag{3.18}$$

$$d \; \leftarrow \; not\, c. \tag{3.19}$$

$$c \; \leftarrow \; a. \tag{3.20}$$

$$a \; \leftarrow \; c, \; d. \tag{3.21}$$

令分割集$U = \{a\}$，则有：

$$b_U(P_1) \; = \; \{a \; \leftarrow \; not\, d.\, a \; \leftarrow \; c, \; d.\} \tag{3.22}$$

使用$gringo$和$clasp$求解得到$\Gamma(b_U(P_1)) = \{\{a\}\}$。而$Atoms(b_U(P_1))\backslash U = \{c, d\}$，所以根据定义有：

$$EC_U(P_1) \; = \; \{d \; \leftarrow \; not\, d'.\, d' \; \leftarrow \; not\, d.\, c \; \leftarrow \; not\, c'.\, c' \; \leftarrow \; not\, c.\} \tag{3.23}$$

使用$gringo$和$clasp$求解$b_U(P_1) \cup EC_U(P_1)$，得到：

$$\Gamma(b_U(P_1) \cup EC_U(P_1)) = \{\{a, c, d\}, \{a, c, d'\}, \{a, c', d'\}, \{c', d\}\} \tag{3.24}$$

其中的c'和d'实质代表$\neg c$和$\neg d$。

因为$head(b_U(P)) \subseteq U$，而且$\Gamma(b_U(P)) \subseteq head(b_U(P))$，所以有$\Gamma(b_U(P)) \subseteq U$。$Atoms(b_U(P))\backslash U$部分的原子真值则无法由$b_U(P)$确定，因为它们有可能被$P\backslash b_U(P)$中的规则推出。加入$EC_U(P)$使得这些原子的真值的有效可能性被加入到$b_U(P)$的回答集中。当然，这里也不是随意的遍历插入。如$\Gamma(b_U(P) \cup EC_U(P))$中只有$\{c', d\}$而没有$\{a, c', d\}$，这是因为在$\{c', d\}$为真的情况下，原程序无法推出$a$为真。这就是本文定义$EC_U(P)$而没有直接往$b_U(P)$的回答集中遍历插入剩下原子的可能真值组合的原因。

此外，由于现在$Atoms(b_U(P))\backslash U \neq \emptyset$且在引入$EC_U(P)$后，新的$bottom$（底部）为$b_U(P) \cup EC_U(P)$，记$bottom$的回答集$X = \Gamma(b_U(P) \cup EC_U(P))$，那么这时就有$X\backslash U \neq \emptyset$。

在$Atoms(b_U(P))\backslash U \neq \emptyset$ 且$X\backslash U \neq \emptyset$的情况下，定义2.20中的操作$e_U(P, X)$只会去掉有形如$a$或$not a$，其中$a \in U$的文字，即只对$U$中的原子进行操作。而实际上，需要保证$U$以外但在$X$中的原子的真值。所以本文为$top$（顶部）定义一个新的规则集合$ECC_U(P, X)$，以保证$X\backslash U$的原子的真值确定性。

定义 3.2： 规则集合$ECC_U(P, X)$为$P\backslash b_U(P)$提供$X\backslash U$中原子的真值确定性，具体有：

$$\begin{aligned} ECC_U(P) = \{ &\leftarrow\ not\ p. \mid p \in Atoms(b_U(P))\backslash U\ and\ p \in X\} \\ \cup\{ &\leftarrow\ p. \mid p \in Atoms(b_U(P))\backslash U\ and\ p \notin X\} \end{aligned} \tag{3.25}$$

对于$ECC_U(P, X)$直观上的功能就是保证在$P\backslash b_U(P)$中每个原子p，若$p \in X$，则要保证p为真，即往逻辑程序中加入$\leftarrow\ not\ p.$；若$p \notin X$，则保证p为假，即往逻辑程序中加入$\leftarrow\ p.$。基于这样的分析，本文可以得到以下命题。

命题 3.3： 已知P是一个正规逻辑程序，U是一个原子集。那么一个原子集$S \subseteq Atoms(P)$能满足P，当且仅当$S = (X \cup Y) \cap Atoms(P)$，其中的$X$和$Y$分别为：

- $X \models b_U(P) \cup EC_U(P)$
- $Y \models e_U(P\backslash b_U(P), X) \cup ECC_U(P, X)$

显然，依据定义$EC_U(P)$和$ECC_U(P,X)$过程中的分析便可以清楚地知道命题3.3的正确性。接下来，看一个验证命题3.3的例子。

例 3.2： 继续使用逻辑程序P_1，并令分割集$U=\{a\}$，那么可以计算得到：

$$b_U(P_1) \ = \ \{a \ \leftarrow \ not\, d.\ a \ \leftarrow \ c,\ d.\} \tag{3.26}$$

及：

$$EC_U(P_1) \ = \ \{d \ \leftarrow \ not\, d'.\ d' \ \leftarrow \ not\, d.\ c \ \leftarrow \ not\, c'.\ c' \ \leftarrow \ not\, c.\} \tag{3.27}$$

且求解得到：

$$\Gamma(b_U(P_1)\cup EC_U(P_1)) = \{\{a,c,d\},\{a,c,d'\},\{a,c',d'\},\{c',d\}\} \tag{3.28}$$

令$X=\{a,c,d'\}$，显然，$X \models b_U(P_1)\cup EC_U(P_1)$。另，根据定义，可以计算得：

$$\begin{aligned} e_U(P_1 \backslash b_U(P_1), X) &= \{d \ \leftarrow \ not\, c.\ c \ \leftarrow .\} \\ ECC_U(P_1, X) &= \{\leftarrow \ not\, c.\ \leftarrow \ d.\} \end{aligned} \tag{3.29}$$

并求解得到：

$$\Gamma(e_U(P_1 \backslash b_U(P_1), X)) = \{\{c\}\} \tag{3.30}$$

令$Y=\{c\}$，显然，$Y \models e_U(P_1 \backslash b_U(P_1), X) \cup ECC_U(P_1, X)$。$X\cup Y=\{a,c,d'\}$，明显也有$X \cup Y \models P_1$。

在引入了$EC_U(P)$和$ECC_U(P,X)$后，依据命题3.3，可以知道原程序的模型可以通过$bottom$和top两部分的模型计算得到。然而，直接使用$P\backslash b_U(P)$作为top并不能求解出原程序剩下的回答集。这是因为分割集U把逻辑程序P划分为$bottom$和top两部分的同时也会可能破坏了P中的某些环L。接下来，本文将使用逻辑程序中的环与环公式的理论，来定义可以用于求解原程序的top部分。首先，这里定义两个结构性的规则集合。

定义 3.4： 对一个逻辑程序P基于一个原子集U，定义以下两个规则集合：

$$in_U(P) = \{r \in P \mid head(r) \cap U \neg \emptyset \ and\ (body^+(r) \cup head(r)) \subseteq U\} \tag{3.31}$$

$$out_U(P) = \{head(r) \subseteq U \ and\ (body^+(r) \cup head(r)) \cap U \neq \emptyset\} \tag{3.32}$$

$in_U(P)$和$out_U(P)$是与逻辑程序正依赖图结构性相关的规则集合。在正规逻辑程序中直观的含义是，$in_U(P)$代表了头部属于分割集U，而体部正原子存在与U中原子不同的原子；$out_U(P)$代表了头部不属于分割集U，而体部正原子存在与U中原子相同的原子。根据两者的定义，显然可以得到：

$$in_U(P) \subseteq b_U(P) \tag{3.33}$$

$$out_U(P) \subseteq P \backslash b_U(P) \tag{3.34}$$

在有了$in_U(P)$和$out_U(P)$的概念后，基于环理论给出以下定义。

定义 3.5 (半环)：一个非空原子集E称为逻辑程序P基于原子集U的半环（$smei - loop$），当且仅当在P中存在一个环L使得$E = L \cap U$且$E \subset L$。

定义 3.6：已知正规逻辑程序P和原子集X、U，且E为P基于U的半环，定义一个半环集合如下：

$$SL_U(P, X) = \{E \mid E \subseteq X \ and \ R^{-}(E, P, X) \subseteq in_U(P)\} \tag{3.35}$$

根据$semi - loop$和$SL_U(P, X)$的定义，可以得到以下引理。

引理 3.7：一个半环E属于$SL_U(P, X)$，则P中存在一个环L满足$E \subset L$且$X \nvDash LF(L, P)$。

证明：使用反证法。

分割集U在把逻辑程序P划分为两部分的同时可能会破坏P中的某些环L。这些被破坏的环可以分为两类：

- 该环已经被$bottom$部分的回答集X所满足；
- 该环不能被$bottom$部分的回答集X所满足，需要top部分的回答集与X的并集去满足。

对于逻辑程序P中一个环L，其与分割集U的关系可以分为以下的情况：

1）$L \subseteq U$。则对于规则$r \in R^{-}(L,P)$有$head(r) \cap L \neq \emptyset$，故$head(r)\ capU \neq \emptyset$，所以$r \in b_U(P)$，因此可以得到：

$$R^{-}(L,P) = R^{-}(L, b_U(P) \cup EC_U(P)) \tag{3.36}$$

故有：

$$LF(L,P) = LF(L, b_U(P) \cup EC_U(P)) \tag{3.37}$$

而$X \in \Gamma(b_U(P) \cup EC_U(P))$，所以有：

$$X \models LF(L, b_U(P) \cup EC_U(P)) \tag{3.38}$$

进而有：

$$X \models LF(L,P) \tag{3.39}$$

2）$L \not\subseteq U$且$L \cap U = \emptyset$，即$L \subseteq Atoms(P) \backslash U$。这样的环$L$即与$b_U(P)$，完全在$P \backslash b_U(P)$中。因为$L \cap U = \emptyset$，则$L \not\subseteq X$。根据环公式被一个原子集满足的定义可以知道$L \not\subseteq X$使得前键为假，故定义中的蕴涵式为真，即有$X \models LF(L,P)$。

当环L完全在$b_U(P)$中时，或者环L完全在$P \backslash b_U(P)$中时，X能满足其环公式。实际上这样的环不属于被U破坏的环。以下情况的环为被U破坏的环。

3）$L \not\subseteq U$且$L \cap U \neq \emptyset$。

当$(L \cap U) \not\subseteq X$时，$L \not\subseteq X$，$X \models LF(L,P)$。

当$(L \cap U) \subseteq X$时，若$L \not\subseteq X$，则也有$X \models LF(L,P)$。这两种即前面说到的环被破坏了，但X能满足其环公式的情况。

若$L \subseteq X$时，若$R^{-}(L \cap U, P, X) \backslash in_U(P) \neq \emptyset$，即存在规则$r \in R^{-}(L \cap U, P, X) \backslash in_U(P)$，且有$R^{-}(L \cap U, P, X) \in b_U(P)$。

对于$r \notin in_U(P)$，所以有$body^{+}(r) \subseteq U$。

对于$r \in R^{-}(L \cap U, P, X)$，所以有$body^{+}(r) \cap (L \cap U) \neq \emptyset$，且已有$L \cap U \neq \emptyset$和$body^{+}(r) \subseteq U$，故$body^{+}(r) \cap L = \emptyset$。同时，$head(r) \cap (L \cap U) \neq \emptyset$，所以$head(r) \cap L \neq \emptyset$。故可以得到存在一个规则$r \in R^{-}(L,P)$且$X \models body(r)$，所以有$X \models LF(L,P)$。

故令$E = L \cap U$，$E \subseteq X$，且有$R^{-}(E,P,X) \in in_U(P)$时，就存在一个环L有$X \nvDash LF(L,P)$。 ■

被分割集U破坏的环的环公式可能不能为$bottom$的回答集X所满足，所以需要剩下部分的回答集联合X一起去满足。定义$SL_U(P,X)$是为了找出P中环公式不能被X所满足的环。根据这些被破坏的环，引用Lin和Zhao在2004年提出的环理论，本文将定义新的top以保证最后得到的回答集能满足这部分环的环公式。

定义 3.8： 给定P为一个正规逻辑程序，X和U为原子集，P基于U通过X得到的top记为$t_U(P,X)$，其由以下三部分组成：

- $P \backslash (b_U(P) \cup out_U(P)$，
- $\{x_E \leftarrow body(r) \,|\, r \in in_U(P) \, and \, r \in R^{-}(E,P,X)\}, \, for \, each \, E \in SL_U(P,X)$，
- $\{head(r) \ \leftarrow \ x_{E_1}, x_{E_2}, ..., x_{E_t}, body(r) \mid r \in out_U(P), \ for \ all \ possible \ E_i \in SL_U(P,X)(1 \leq i \leq t) \ s.t. \ body^{+}(r) \cap E_i \neq \emptyset\}$。

其中x_{E_i}为基于$SL_U(P,X)$中的$smei-loop$引入的新原子，对于这些新原子，最后通过逻辑交$Atoms(P)$即可消去。

在明确了新的top后，本文将对定义2.21的重定义，给出新的逻辑程序P基于原子集U的方案（$Solution$）。

定义 3.9： 给定正规逻辑程序P和原子集U，P基于U的方案（$Solution$）是一组原子集$\langle X, Y \rangle$，其中：

- X是$b_U(P) \cup EC_U(P)$的一个回答集；
- Y是$e_U(t_U(P,X),X) \cup ECC_U(P,X)$的一个回答集。

下面给出一个计算逻辑程序P基于原子集U的方法的例子。

例 3.3： 继续使用逻辑程序P_1，并令分割集$U = \{a\}$，根据定义3.4可以计算得到：

$$in_U(P_1) = \{a \ \leftarrow \ c, \ d.\} \tag{3.40}$$

$$out_U(P_1) = \{c \ \leftarrow \ a.\} \tag{3.41}$$

且可以从P_1中取环$L = \{a, c\}$，则有$E = L \cap U = \{a\}$，满足$E \subset L$。并由例3.2已经算得：

$$\Gamma(b_U(P_1) \cup EC_U(P_1)) = \{\{a, c, d\}, \{a, c, d'\}, \{a, c', d'\}, \{c', d\}\} \tag{3.42}$$

这里，取$X = \{a, c, d'\}$和$X' = \{a, c, d\}$，那么可以计算得到：

$$R^-(E, P_1, X) = \{a \ \leftarrow \ not\ d.\} \tag{3.43}$$

$$R^-(E, P_1, X') = \{a \ \leftarrow \ c, \ d.\} \tag{3.44}$$

则有：

$$SL_U(P_1, X) = \emptyset \tag{3.45}$$

$$SL_U(P_1, X') = \{\{a\}\} \tag{3.46}$$

并基于定义3.8计算得到：

$$t_U(P_1, X) = \{d \ \leftarrow \ not\ c.\ c \ \leftarrow \ a.\} \tag{3.47}$$

$$t_U(P_1, X') = \{d \ \leftarrow \ not\ c.\ x_{\{a\}} \ \leftarrow \ c, \ d.\ c \ \leftarrow \ x_{\{a\}}, \ a.\} \tag{3.48}$$

另外，有：

$$ECC_U(P_1, X) = \{\leftarrow \ not\ c. \ \leftarrow \ d.\} \tag{3.49}$$

$$ECC_U(P_1, X') = \{\leftarrow \ not\ c. \ \leftarrow \ not\ d.\} \tag{3.50}$$

通过求解器计算得到：

$$\Gamma(e_U(t_U(P_1, X), X) \cup ECC_U(P_1, X)) = \{\{c\}\} \tag{3.51}$$

$$\Gamma(e_U(t_U(P_1, X'), X') \cup ECC_U(P_1, X')) = \emptyset \tag{3.52}$$

其中$\langle X, \{c\} \rangle$为P基于U的一个方案。

在有了新的方案的定义后，本文提出一个引理，具体如下：

引理 3.10： 对任意的正规逻辑程序P和原子集U，如果$\langle X, Y\rangle$是P基于U的一组方案，且$SL_U(P, X) = \emptyset$，那么$(X \cup Y) \cap Atoms(P)$是逻辑程序P的一个回答集。

证明： 根据前面已提及的$SL_U(P, X)$的属性，如果一个$smei - loop\ E$属于$SL_U(P, X)$，则表示存在一个环L，有$E \subset L$，且$X \nvDash LF(L, P)$。所以在$SL_U(P, X) = \emptyset$时，可以得到：在逻辑程序P中不存在环L满足以下三点：

- $L \cap U = \emptyset$,
- $L \cap (Atoms(P)\backslash U) \neq \emptyset$,
- $X \cup Y \nvDash LF(L, P)$。

所以可以进一步得到$(X \cup Y) \cap Atoms(P)$满足P中的所有环公式。而命题3.3中指出$(X \cup Y) \cap Atoms(P)$是P的模型，联合Lin和Zhao的环理论，即定理2.4，可以知道P的模型若能满足其所有环公式，则是P的一个回答集，所以可以得到$S = (X \cup Y) \cap Atoms(P)$是$P$的一个回答集。 ■

更进一步地，本文依据新的程序分割方法，提出新的分割理论。

定理 3.1 (新分割理论)： 已知正规逻辑程序P和原子集U，一个原子集S是P的回答集，当且仅当$S = (X \cup Y) \cap Atoms(P)$，其中$\langle X, Y\rangle$是$P$基于$U$的某组方案。

在定理3.1中给出的新分割理论实质上是跟Lifschitz和Turner当初提出的分割理论是一样的。本质上的不同在于定义来计算P基于U的方案$\langle X, Y\rangle$的运算符的不同。

$SL_U(P, X)$根据Lin和Zhao的环理论，定义的让底部回答集X所不能满足的环公式的环与分割集的交集所构成的半环。定义这些半环的目的就是为了依据它们构造出能保证X为真的规则。然而，更进一步地，基于Ji和Wan等在环理论的基础下提取出特征环（$Proper\ Loop$）的思想下，可以想象得到并非所有的半环都是必要的，本文在此根据Ji和Wan等的思想，提取出必要的半环，定义一个$SL_U(P, X)$的子集以指代必须的半环集合。

定义 3.11 (关键半环（$Dominated\ Semi-loop$）)： 给定正规逻辑程序P和原子集U，一个半环E被称为关键半环，当且仅当存在另一个半环E'满足$E' \subseteq E$，且有$E' \cap head(in_U(P)) = E \cap head(in_U(P))$，以及$E' \cap body^+(out_U(P)) = E \cap body^+(out_U(P))$，同时称$E$能代表$E'$。

根据关键半环的特性，因为关键半环能涵盖其所代表的半环的头部和体部正原子，所以关键半环的环公式可以推出其所能代表的半环的环公式。由于关键半环能够代替其他半环，所以这里定义一个关键半环的集合。

定义 3.12 (关键半环集)： 给定析取逻辑程序P和原子集U，P关于U的关键半环集记为$DSL_U(P,X)$，并定义为以下集合：

$$\begin{aligned} DSL_U(P,X) = \{E \mid E \in SL_U(P,X)\ and\ there\ dose\ not \\ exist\ another\ E' \in SL_U(P,X)\ s.t.\ E\ is\ dominated\ by\ E'\} \end{aligned} \tag{3.53}$$

本文在此提供一个计算$DSL_U(P,X)$的算法$Algorithm2$，具体如下：

基于$DSL_U(P,X)$，本文定义关键顶部（$Dominated\ Top$）和关键方案（$Dominated\ Solution$），它们的具体内容就是通过把其中用到的$SL_U(P,X)$替换为$DSL_U(P,X)$，并记关键顶部为$dt_U(P,X)$。最后本节扩展新的分割集理论，得到以下定理。

定理 3.2： 给定一个正规逻辑程序P和原子集U，一个原子集S是P的一个回答集，当且仅当$S = (X \cup Y) \cap Atoms(P)$，其中$\langle X, Y\rangle$是$P$基于$U$的某组关键方案。

对于$bottom$部分，通过新定义$EC_U(P)$来补充$Atoms(b_U(P))\backslash U$中原子真值的可能性，以保证这部分原子不会被忽略。

对于top部分，定义了$ECC_U(P,X)$来在top中确保$bottom$部分的回答集中的原子必为真。同时使用环理论，引入$smei-loop$的集合$SL_U(P,X)$，其代表的是环公式不能被底部回答集所满足的环所相关的$smei-loop$集合。基于$SL_U(P,X)$和原来的$P\backslash b_U(P)$来构建出新的top，即定义3.8中的$t_U(P,X)$。

Algorithm 2: 计算关键半环集的算法$dsl_U(P,X)$

输入: 一个逻辑程序P和一个原子集X

输出: 关键半环集dsl

$dsl := \emptyset$;
for 非空子集$S_1 \subseteq head(in_U(P))$和$S_2 \subseteq body^+(out_U(P))$ **do**
  $G_P^S := G_P$关于$S_1 \cup S_2 \cup (Atoms(P) \backslash (head(in_U(P)) \cup body^+(out_U(P))))$的子图;
  $L := G_P^S$的一个强连通分量，并满足$S_1 \cup S_2 \subseteq L$;
  if L不存在 **then**
    break;
  end
  if $L \cap U \subseteq X$且$R^-(L \cap U, P, X) \subseteq in_U(P)$ **then**
    把$L \cap U$加入到dsl中;
  end
  else
    $S := head(R^-(L \cap U, P, X) \backslash in_U(P)) \cup ((L \cap U) \backslash X)$;
    $G_P^S := G_P$关于$L \backslash S$的子图;
    $goto$ 5;
  end
end
return dsl

在上述的定义和分析证明过程中可以明确地知道新的程序分割方法能对任意原子集构成的分割集都有效。

在给出了正规逻辑程序的新程序分割方法后，下一节中将新的程序分割方法扩展到析取逻辑程序中。

3.2.2 析取逻辑程序分割方法

本小节将把新的程序分割方法从正规逻辑程序扩展到析取逻辑程序。由于之前所定义下的规则集合都是基于普遍规则所定义的，只是在分析证明时用了正规逻辑程序的性质，所以对于析取逻辑程序，这些规则集合的操作符依旧有用。

具体地，$b_U(P)$，$e_U(P,X)$，$EC_U(P)$，$ECC_U(P,X)$，$in_U(P)$，$out_U(P)$，$SL_U(P,X)$和$DSL_U($
保持原来的定义不变。跟正规逻辑程序有所不同的是，在析取逻辑程序

中，$in_U(P) \cap out_U(P) \neq \emptyset$。在明确了需要用到的基本概念后，本文将重新定义析取逻辑程序的top部分，依旧标记为$t_U(P, X)$。

定义 3.13 (析取逻辑程序的top)： 给定一个析取逻辑程序P和原子集X，U，P基于U关于X的top记为$t_U(P, X)$，并由以下三部分组成：

- $P \backslash (b_U(P) \cup out_U(P)$，
- $\{\{x_E\} \cup head(r) \backslash E \leftarrow body(r) \mid r \in in_U(P) \, and \, r \in R^-(E, P, X)\}, \, for \, each \, E \in SL_U(P, X)$，
- $\{head(r) \leftarrow x_{E_1}, x_{E_2}, ..., x_{E_t}, body(r) \mid r \in out_U(P), \, for \, all \, possible \, E_i \in SL_U(P, X)(1 \leq i \leq t) \, s.t. \, body^+(r) \cap E_i \neq \emptyset\}$。

这里给出如此定义析取逻辑程序的$t_U(P, X)$的有效性的证明。

在证明析取逻辑程序的top的有效性后，关于一个析取逻辑程序P基于一个原子集U的方案定义依旧，只是相应的$t_U(P, X)$修改为定义3.8中的。同时，基于上述的分析和证明，本文给出关于析取逻辑程序在程序分割理论下的回答集定理。

定理 3.3： 给定析取逻辑程序P和原子集U，一个原子集S是P的一个回答集，当且仅当$S = (X \cup Y) \cap Atoms(P)$，其中$\langle X, Y \rangle$是$P$基于$U$的某组方案。

联合析取逻辑程序的$t_U(P, X)$的有效性的证明和3.2.1中关于定理3.1的证明可以直接地得到定理3.3的正确性。

3.2.3 强程序分割方法

在Lifschitz和Turner的分割理论中，分割集U把逻辑程序P分割为$b_U(P)$和$P \backslash b_U(P)$两部分，原逻辑程序的回答集可以通过这两部分的回答集来求解得到。除了这个直接面对回答集求解的特性之外，Lifschitz和Turner的分割理论还能引申出来其他的特性。对此，本文总结出如下命题。

命题 3.14： 给定逻辑程序P，原子集U是P在Lifschitz和Turner的分割理论定义下的分割集。另有逻辑程序P'满足$Atoms(P') \cap U = \emptyset$则原子集$S$是逻辑

程序$P \cup P'$的一个回答集，当且仅当$S = X \cup Y$，其中X为$b_U(P)$的某个回答集，Y为$e_U(P \backslash b_U(P), X) \cup P'$的某个回答集。

证明： 由于$Atoms(P') \cap U = \emptyset$，且$\Gamma(b_U(P)) \subseteq U$，所以$\Gamma(P') \cap \Gamma(b_U(P)) = \emptyset$，同时有$\Gamma(P') \cap U = \emptyset$。所以$P \cup P'$中划分出的$b_U(P)$依旧具有命题封闭性，其回答集依旧不受$(P \cup P') \backslash b_U(P)$影响。

又在$Atoms(P') \cap U = \emptyset$及$Atoms(b_U(P)) \subseteq U$情况下，可以得到：

$$(p \cup P') \backslash b_U(P) \equiv (P \backslash b_U(P)) \cup P' \tag{3.54}$$

而$e_U(P, X)$操作中只会依旧条件删去与U有关的文字以保证X中的原子的真值确定性。所以在$Atoms(P') \cap U = \emptyset$情况下，$e_U(P, X)$不需要对$P'$操作，而$P'$所能推出的结论也不会干扰$P \backslash b_U(P)$或受其干扰，所以$e_U(P \backslash b_U(P), X) \cup P'$可以求解出$P \cup P'$的另一部分回答集。

综上所述，可以得到命题3.14的正确性。 ■

然而，对于本文所关注和扩展的使用任意原子集作为分割集时，命题3.14并不成立。但基于这个思想，为了继续扩展分割集理论对逻辑程序的结合律有效，本文提出了强程序分割方法（*Strong Splitting Method*）。为此，本文对$SL_U(P, X)$进行了扩展。

定义 3.15： 给定析取逻辑程序P以及原子集X和U，标记以下原子集的集合：

$$\begin{aligned} SS_U(P, X) \ = \ \{E \mid & E \ is \ an \ nonempty \ subset \ of \ U, \\ & E \subseteq X \ and \ R^{-}(E, P, X) \subseteq in_U(P)\} \end{aligned} \tag{3.55}$$

很直观地，$SS_U(P, X)$把$SL_U(P, X)$从P关于U的$smei - loop$扩展到U的任意非空子集。同时，对任何满足$Atoms(P') \cap U = \emptyset$的析取逻辑程序$P'$，可以得到$b_U(P \cup P') = b_U(P)$，且对任意的原子集$X$有$SL_U(P \cup P', X) \subseteq SS_U(P, X)$。此外，若$U$是Lifschitz和Turner的分割理论定义下的分割集，则对任意原子集X都有$SS_U(P, X) = \emptyset$。基于$SS_U(P, X)$，本文提出了析取逻辑程序P基于原子集U关于原子集X的强顶部（*Strong Top*）。

定义 3.16 (强顶部（$Strong\,Top$）)： 给定析取逻辑程序P以及原子集X和U，记P基于U关于X的强顶部（$Strong\,Top$）为$st_U(P,X)$，其由以下三部分组成：

- $P\backslash(b_U(P)\cup out_U(P)$，
- $\{\{x_E\}\cup head(r)\backslash E \leftarrow body(r)\,|\,r\in in_U(P)\;and\,r\in R^-(E,P,X)\},\;for\,each\,E\in SS_U(P,X)$，
- $\{head(r)\ \leftarrow\ x_{E_1},x_{E_2},...,x_{E_t},body(r)\ |\ r\ \in\ out_U(P),\ for\ all\ possible\ E_i\ \in SS_U(P,X)(1\le i\le t)\ s.t.\ body^+(r)\cap E_i\neq\emptyset\}$。

在给出$st_U(P,X)$后，本文定义一个析取逻辑程序P关于原子集U的强方案（$Strong\ Solution$）。

定义 3.17 (强方案（$Strong\,Solution$）)： 给定析取逻辑程序P和原子集U，P关于U的强方案为一个原子集组合$\langle X,Y\rangle$，其中X和Y满足：

- X是$b_U(P)\cup EC_U(P)$的一个回答集；
- Y是$e_U(st_U(P,X),X)\cup ECC_U(P,X)$的一个回答集。

在有了强方案的定义后，依据$SS_U(P,X)$对$SL_U(P,X)$性质的扩展，可以得到以下两个定理。

定理 3.4： 给定析取逻辑程序P和原子集U。一个原子集S是P的一个回答集，当且仅当$S=(X\cup Y)\cap Atoms(P)$，其中$\langle X,Y\rangle$是P基于U的某组强方案。

定理 3.5： 给析取定逻辑程序P和原子集U。另有逻辑程序P'满足$Atoms(P')\cap U=\emptyset$。则原子集$S$是逻辑程序$P\cup P'$的一个回答集，当且仅当$S=(X\cup Y)\cap Atoms(P)$，其中$X$为$b_U(P)\cup EC_U(P)$的某个回答集，$Y$为$e_U(st_U(P,X),X)\cup ECC_U(P,X)\cup P'$的某个回答集。

定理3.5把程序分割方法的结合性扩展到以任意原子集作为分割集的情况。而这种结合性的本质是一个二次分割，这种扩展有助于程序分割方法的递归问题研究。

3.3 计算复杂性分析

在上一节中，本文详细地给出了正规逻辑程序和析取逻辑程序的新程序分割方法，并证明了新的程序分割方法对任意原子集构成的分割集都有效。当真正的自由是规矩。任意原子集给了程序分割很大的自由度，但这更多是理论层面的。本节将分析讨论分割集如何影响着程序分割过程的计算复杂性，并指出怎样的原子集作为分割集才会让程序分割更高效。此外，由于析取逻辑程序跟正规逻辑程序的本质区别在于头部的基，即头部中原子的数量。所以本节以正规逻辑程序在程序分割方法上的计算复杂性为例进行分析。

对于一个正规逻辑程序P通过一个原子集U进行程序分割后被划分为底部和顶部两部分，而程序分割过程中主要耗时的有两个地方，分别为底部的$EC_U(P)$和顶部的$SS_U(P,X)$所造成。具体体现为以下两个问题：

1. 在底部中，$EC_U(P)$中的规则在最坏情况下会为$b_U(P) \cup EC_U(P)$带来$2^{|Atoms(b_U(P)\setminus U|}$个回答集，这是一种指数爆炸的极端；

2. 在顶部中，$DSL_U(P,X)$中的半环会为$dt_U(P)$引入指数爆炸数量的新原子。

接下来的小节将分别对上述两个问题进行具体的分析和给出提速方案。

3.3.1 底部计算复杂性

本小节主要对第一个问题进行分析。底部中的$EC_U(P)$是为了补充$Atoms(b_U(P))\setminus U$中的原子在回答集中的真值可能性而定义的。对于$EC_U(P)$带来的耗时，本节给出以下两个降低计算复杂性的方法：

- 保证$|Atoms(b_U(P))\setminus U|$足够小，最好是在某个常数值以内；
- 让分割集U中的原子能被逻辑程序P的每个回答集所满足。

证明： 第一个方法中，在$|Atoms(b_U(P))\setminus U|$足够小的情况下，特别地，是保持在某个常数值以内的话，$2^{|Atoms(b_U(P))\setminus U|}$的大小便会被限制，这样就能防止底部回答集数量因为引入的新原子而造成指数爆炸。这是一个非常直观的措施。此

外，由于分割集能够是任意原子集，所以限制$|Atoms(b_U(P))\backslash U|$的大小是具备可行性的。

第二个方法中，若能使得分割集U中的原子能被逻辑程序P的每个回答集所满足，那么这在后续计算顶部的$dt_U(P)$时只需考虑$b_U(P) \cup EC_U(P)$中满足$U \subseteq X$的回答集X。这样就能大大降低$dt_U(P)$的复杂性。 ■

需要明确的一点是，能被逻辑程序P的每个回答集所满足的原子集有独立的定义和算法，不需要求解出逻辑程序的所有回答集才能得到。所以找出这样的原子集去作为分割集是可行的。而这样的原子集也将给本文带来分割集应用的新思路。具体的定义和算法将会在第四章给出。

3.3.2 顶部计算复杂性

本小节主要对第二个问题进行分析。关于顶部中$DSL_U(P, X)$所带来的复杂性，依据其定义可以知道其中的关键半环E与其所能代表的半环E'必须满足：

$$E \cap head(in_U(P)) = E' \cap head(in_U(P)) \tag{3.56}$$

$$E \cap body^+(out_U(P)) = E' \cap body^+(out_U(P)) \tag{3.57}$$

而关键半环本来是可以代表其他一般半环，此外，半环也是由一般的环交分割集而来，所以保证不存在两个不同的关键半环可以同时代表一个一般半环的情况。在关键半环的互斥性之下，关键半环的数量的理论最大值将由$head(in_U(P))$和$body^+(out_U(P))$的基所决定。

具体来说就是$E \cap head(in_U(P))$和$E \cap body^+(out_U(P))$的可能情况数量所决定。设$n = |head(in_U(P))|$和$m = |body^+(out_U(P))|$，那么有：

$$|E \cap head(in_U(P))| \leq \sigma_{0 \leq i \leq n} C_n^i = 2^n \tag{3.58}$$

其中C_n^i为排列组合中的组合计算符号。同理，还有：

$$|E \cap body^+(out_U(P))| \leq 2^m \tag{3.59}$$

故有：

$$|DSL_U(P,X)| \leq |E \cap head(in_U(P))| * |E \cap body^+(out_U(P))| \leq 2^n * 2^m \qquad (3.60)$$

实际上，上述的$|DSL_U(P,X)|$规模结论也能从其算法中得到。从上述的结论中可以明显地知道通过限制$|head(in_U(P))|$和$|body^+(out_U(P))|$的大小可以限制$|DSL_U(P,X)|$的大小。而$|head(in_U(P))|$和$|body^+(out_U(P))|$的大小依旧可以通过分割集U来限制。而限制$|DSL_U(P,X)|$的大小后，在$dt_U(P)$中的第二和第三部分中将会大大减少新原子的引入，进而降低程序规模，加快求解速度。

上述的计算复杂性都是基于正规逻辑程序而进行的。而事实上，正规逻辑程序的程序分割的计算复杂性的相关结论可以应用于析取逻辑程序上。因为两者仅在规则头部的基上有所不同，同时头部原子为析取关系。

3.4 本章小结

本章给出了Lifschitz和Turner的分割集理论的证明，并提出了在正规逻辑程序中对任意原子集作为分割集都有效的新程序分割方法，并把这个方法推广到析取逻辑程序。此外，在分析程序分割方法的计算复杂性时，引申出通过程序结论来扩展新分割理论的应用场景，这个应用场景将紧接着在下一章描述。

第 4 章　新分割理论的应用

本章将把分割集的思想应用到程序化简的应用中，并在程序化简里进行扩展，在分割集的思想上找出对程序化简更高效的原子集，并命名为可靠集（$Reliable\ Set$）。

4.1　程序化简

本节将把分割集理论应用到程序化简中。实际上是把程序结论作为分割集应用于程序化简中。本节将定义两个化简操作和程序结论顶部来完成分割集理论在程序化简上的应用，并给出相关的证明。程序结论中的文字的真值在对应的逻辑程序中的所有回答集里都为真。基于这个确定性，可以把原程序中的一些文字及其所影响的规则删掉，得到一个简化的逻辑程序，并对简化后的逻辑程序进行求解，得到的回答集联合程序结论即可得到原程序的回答集。这就是程序化简后求解逻辑程序的基本思路。

4.1.1　程序结论与补

本节将给出程序结论和下一小节的化简操作中需要用到的补的定义。本文所指的程序化简问题即为如何使用程序结论来化简程序。化简后的程序在求解回答集上将比原程序要快得多，因为程序结论将有效地在程序化简过程中把一些已经可以确认真值的文字对程序求解的影响去掉。首先，本文给出程序结论和补的定义。

定义 4.1 (程序结论（$Consequence$）)： 给定正规逻辑程序P和一个文字集L，如果L能被P的每个回答集所满足，则称文字集L为逻辑程序P的一个程序结论（$Consequence$）。

程序结论的原始定义是面向文字集的，而在只考虑正文字的情况下，文字集可以被视为原子集，下面所涉及的程序结论的使用都将其视为一个原子集。此

外，程序结论并不需要在求解出逻辑程序的所有回答集后才能得到，这样也保证了本文使用程序结论的可行性。以下为计算一个程序结论的算法。

接下来，本文给出一个文字集的补（$Complement$）的定义，文字集的补在后续的化简操作符中将被使用。

定义 4.2 (补（$Complement$）)： 对于一个文字l，其补记为$\bar{l}$。其中，若l为a形式的正文字，则$\bar{l}$为$\neg a$，若l为$\neg a$形式的负文字，则$\bar{l}$为a。对于一个文字集L，其补定义为：$\overline{L} = \{\bar{l} \mid l \in L\}$。

在有了程序结论和文字集的补的定义后，下一小节基于这两个概念定义了两个化简操作，这两个化简操作是程序化简的基础。

4.1.2 两个化简操作符

基于程序结论中的文字的真值确定性，本小节定义两个基于程序结论去简化逻辑程序的化简操作符。它们为原逻辑程序删去受程序结论影响的文字和规则。下面将给出这两个化简操作符的具体的定义，以及给出对它们有效性和合理性的证明。

定义 4.3 (负文字简化)： 给定逻辑程序P和它的一个程序结论L，P基于L中负文字的化简记为$tr_n(P, L)$。$tr_n(P, L)$通过对原逻辑程序P执行以下两步操作得到：

- 若P中的一个规则r中存在原子$p \in body^+(r)$，且有$\neg p \in L$，则从P中删去r；
- 若P中的一个规则r中存在原子$p \in head(r)$或$p \in body^-(r)$，且有$\neg p \in L$，则把该规则r替换为以下形式：

$$head(r) \backslash \overline{L} \leftarrow body^+(r),\ body^-(r) \backslash \overline{L} \tag{4.1}$$

定义 4.4 (正文字)： 给定逻辑程序P和它的一个程序结论L，P基于L中正文字的化简记为$tr_p(P, L)$。$tr_p(P, L)$通过对原逻辑程序P执行以下两步操作得到：

- 若P中的一个规则r中存在原子$p \in head(r)$或$p \in body^-(r)$，且有$p \in L$，则从P中删去r；
- 若P中的一个规则r中存在原子$p \in body^+(r)$，且有$p \in L$，则把该规则r替换为以下形式：

$$head(r) \leftarrow body^+(r) \backslash L, \; body^-(r) \tag{4.2}$$

接下来本文给出关于上述两个定义的合理性和正确性的证明。

证明： 对于负文字化简$tr_n(P,L)$，其只考虑程序结论L中的负文字，即满足$\neg p \in L$的原子p。在$\neg p \in L$时，基于程序结论的定义性质，可以得到推不出p为真，所以得到p为假，$not\ p$为真。对于$tr_n(P,L)$的两个步骤有：

1. 若$p \in body^+(r)$，且规则体部文字为合取关系。对于一个合取式$A \wedge B \wedge C$，若可以确定$C \equiv false$，那么有$A \wedge B \wedge C \equiv false$。所以在可以确定$p$为假的情况下，则可以确定$body(r)$为假，故这样的体部无法推出头部，进而可以把这个规则删掉。

2. 若$p \in head(r)$，且规则头部文字为析取关系。对于一个析取式$A \vee B \vee C$，若可以确定$C \equiv false$，那么有$A \vee B \vee C \equiv A \vee B$。所以在可以确定$p$为假的情况下，可以从$head(r)$中删掉$p$。若$p \in body^-(r)$，且规则体部文字为合取关系。对于一个合取式$A \wedge B \wedge C$，若可以确定$C \equiv true$，那么有$A \wedge B \wedge C \equiv A \wedge B$。所以在可以确定$not\ p$为真的情况下，则可以确定$body^-(r)$删去$p$。所以$tr_n(P,L)$的第二步把符合要求的规则替换为$head(r) \backslash \overline{L} \leftarrow body^+(r), \; body^-(r) \backslash \overline{L}$。

对于正文字化简$tr_p(P,L)$，其只考虑程序结论L中的正文字，即满足$p \in L$的原子p。在$p \in L$时，基于程序结论的定义性质，可以推出p为真，$not\ p$为假。对于$tr_p(P,L)$的两个步骤有：

1. 若$p \in head(r)$，且规则头部文字为析取关系。对于一个析取式$A \vee B \vee C$，若可以确定$C \equiv true$，那么有$A \vee B \vee C \equiv true$。所以在可以确定$p$为真

的情况下，则可以确定$head(r)$为真，故此规则无须考虑，可以删去。若$p \in body^{-}(r)$，且规则体部文字为合取关系。对于一个合取式$A \wedge B \wedge C$，若可以确定$C \equiv false$，那么有$A \wedge B \wedge C \equiv false$。所以在可以确定$not\ p$为假的情况下，则$body(r)$为假，无法推出结论，故可以删去此规则。

2. 若$p \in body^{+}(r)$，且规则体部文字为合取关系。对于一个合取式$A \wedge B \wedge C$，若可以确定$C \equiv true$，那么有$A \wedge B \wedge C \equiv A \wedge B$。所以在可以确定$not\ p$为真的情况下，则可以确定$body^{+}(r)$删去$p$。所以$tr_p(P, L)$的第二步把符合要求的规则替换为$head(r) \leftarrow body^{+}(r) \backslash L,\ body^{-}(r)$。

综上所述，定义4.3和4.4中的基于程序结论中文字的真值特性所定义的正负文字化简操作符是合理而且有效的。 ■

根据正文字化简和负文字化简操作符的定义可以知道给定一个逻辑程序P和它的一个程序结论L，$tr_n(P, L)$得到的逻辑程序中将不含任何满足$\neg p \in L$的原子p。同理，$tr_p(P, L)$得到的逻辑程序中将不含任何满足$p \in L$的原子p。那么$tr_p(tr_n(P, L), L)$得到的逻辑程序中将不含任何满足$p \in Atoms(L)$的原子p。

对于一个逻辑程序P，$tr_p(tr_n(P, L), L)$使得P基于它的程序结论L把已经可以确定真值的文字及这些文字对P的影响移除。这个操作的实质效果便是：在已知一部分回答集（即程序结论）的情况下，去掉这部分对逻辑程序P的影响，在剩下的部分里求解出余下的回答集部分。这个过程跟程序分割通过底部求解出X，代入顶部求解出剩下的回答集部分Y是一致的。

Van和Gelder等人在1991年提出的正规逻辑程序的良序模型（$Well-founded\ Model$）[32]也是正规逻辑程序的一个程序结论。当前的主要ASP求解器都内置地使用了良序模型去化简程序。但存在其他比良序模型的基要大的程序结论，而使用这些程序结论将比良序模型更有效地化简程序。明显地，程序结论的基越大，能消去的确定性的的文字越多，程序规模越小，求解越快。Chen，Ji和Lin在2013年时提出了通过使用最多只有一个外部支持的环的环公式去计算程序结论，将有效地得到比良序模型的基要大的程序结论[33]。

事实上，$tr_n(P,L)$和$tr_p(P,L)$两者虽然只是根据程序结论中的不同类型的文字（负/正）去化简逻辑程序，但两者的适用性并不完全一致。具体情况将由下面的命题带出。

命题 4.5： 给定逻辑程序P，文字集L是P的一个程序结论，那么有：

- 一个原子集S是逻辑程序P的回答集当且仅当S是$tr_n(P,L)$的一个回答集；
- 一个原子集S是逻辑程序P的回答集可以推出$S\backslash L$是$tr_p(P,L)$的一个回答集，但反之不然。

例 4.1： 给定逻辑程序P如下：

$$a \leftarrow b. \tag{4.3}$$

$$c \leftarrow a. \tag{4.4}$$

$$b \leftarrow c. \tag{4.5}$$

$$c \leftarrow d. \tag{4.6}$$

$$a \leftarrow f. \tag{4.7}$$

$$d \leftarrow not\ e. \tag{4.8}$$

$$e \leftarrow not\ d. \tag{4.9}$$

$$\leftarrow not\ a. \tag{4.10}$$

$$f \leftarrow a. \tag{4.11}$$

使用本节提供的算法可以算的程序结论$L=\{a,f\}$，并可以计算得到：

$$tr_p(P,L)=\{c.\ b \leftarrow c.\ c \leftarrow d.\ d \leftarrow not\ e.\ e \leftarrow not\ d.\} \tag{4.12}$$

且有程序P的回答集为：

$$\Gamma(P)=\{a,b,c,d,f\} \tag{4.13}$$

及$tr_p(P,L)$的回答集为：

$$\Gamma(tr_p(P,L))=\{\{b,c,d\},\ \{b,c,e\}\} \tag{4.14}$$

其中$a, b, c, d, f \backslash L = \{b, c, d\} \neq \{b, c, e\}$。

$tr_n(P, L)$和$tr_p(P, L)$从定义上看只是差异在两者对程序结论中文字类型的应用上。但本质上来说，在逻辑程序中删去正负文字的影响是不一样的。具体的原因涉及Lin和Zhao提出的环理论。下面将给出其正确性的具体的证明。

证明： 给定逻辑程序P，令E为P中的一个环，L为P的一个程序结论。

对于命题4.5中的第一个结论，E是P中的一个环，但E不一定能保持也是$tr_n(P, L)$中的一个环，因为$tr_n(P, L)$中可能删掉某些规则导致环E被破坏。然而$tr_n(P, L)$的回答集仍然能满足环E的环公式。这是因为$tr_n(P, L)$删去的是程序结论中的负文字及其相关的影响。负文字代表推不出为真的原子，这些原子不会作为任何一个环的外部支持，所以去掉这些相关的文字并不会影响环公式被满足的基本要求。故保证了$tr_n(P, L)$的回答集仍然能满足逻辑程序中所有环的环公式。同时负文字必然不会作为回答集的部分，所以得到第一个结论的正确性。

对于命题4.5中的第二个结论，E是P中的一个环，但E也不一定能保持也是$tr_p(P, L)$中的一个环，因为$tr_p(P, L)$中同样可能会删去某些正体部的原子导致环E被破坏。但由于$tr_p(P, L)$是基于程序结论的正文字进行化简的，会删去这些正文及其影响。而这些正文字可能作为环E的外部支持，故当这些正文字被删除后，求解出来的回答集将不满足环E的环公式。同时，这些正文字是会出现在原程序的回答集中的，故$S \backslash L$作为$tr_p(P, L)$的回答集时并不一定能保证S是原程序的一个回答集，它可能不能满足原程序的所有环公式。 ■

基于命题4.5的两个结论，可以知道$tr_n(P, L)$比$tr_p(P, L)$的泛用性更好。然而$tr_p(P, L)$使用的程序结论中的正文字才更有意义，因为这部分将是原程序回答集中的内容。所以本文后续将顺着需要满足所有环公式的思路去设计能让$tr_p(P, L)$泛化的操作符。

4.1.3 程序化简操作

本节将提出使用程序结论去分割逻辑程序，并对分割出来的结论顶部（$Consequence\ Top$）进行$tr_p(P, L)$化简，并给出一个$tr_p(P, L)$具有泛用性的命题

结论。

对于一个逻辑程序P及其一个程序结论U，本节将使用U去分割P以得到一个不含U中任何原子的新逻辑程序，再使用$tr_p(P,L)$对新的逻辑程序进行化简，求解回答集，以得到原程序的实际回答集。首先，本节给出后续需要用到的相关定义。

定义 4.6： 给定逻辑程序P及其程序结论U，标记与U无关的规则集合为$in'_U(P)$，且有：

$$in'_U(P) = \{r \in P \mid head(r) \cap U \neq \emptyset,\ Atoms(r) \nsubseteq U\} \tag{4.15}$$

定义 4.7： 给定逻辑程序P及其程序结论U，标记P基于U的结论半环（$Consequence\ Semi-Loop$）为$CS_U(P)$，且有：

$$CS_U(P) = \{E \mid E \subseteq U,\ E \neq \emptyset,\ R^-(E,P) \subseteq in'_U(P)\} \tag{4.16}$$

定义 4.8 (结论顶部（$Consequence\,Top$）)： 给定逻辑程序P及其程序结论U，标记P基于U的结论顶部为$ct_U(P)$，并由以下四部分的并集组成：

- $P \backslash (b_U(P) \cup out_U(P)$，
- $\{\{x_E\} \cup head(r) \backslash E \leftarrow body(r) \mid r \in in_U(P)\, and\, r \in R^-(E,P,X)\},\ for\, each\, E \in CS_U(P,X)$，
- $\{head(r) \leftarrow x_{E_1}, x_{E_2}, ..., x_{E_t}, body(r) \mid r \in out_U(P),\ for\ all\ possible\ E_i \in CS_U(P,X)(1 \leq i \leq t)\ s.t.\ body^+(r) \cap E_i \neq \emptyset\}$，
- $\{\leftarrow\ not\ x_E\},\ for\ each\ E \in CS_U(P)$。

结论顶部的设计跟析取逻辑程序的顶部类似，其中加入了最后一部分以保证原程序的所有环公式都能被满足。

依此，本文给出使$tr_p(P,L)$具有泛用性的命题结论，并给出相关的证明和例子。注意，这里只考虑只包含正文字的程序结论，所以下面也将程序结论看做一个原子集。

命题 4.9： 给定正规逻辑程序P和其程序结论U，一个原子集S是P的一个回答集，当且仅当存在$tr_p(ct_U(P),U)$的一个回答集$S*$满足$S\backslash U = S*\cap Atoms(P)$。

证明： lack of ■

在计算复杂性方面，$tr_p(ct_U(P),U)$同样会因为$CS_U(P)$而引入新原子，这里也存在指数爆炸的最坏情况。但类似分割理论中的分析，只要通过限制$|in'_U(P)|$的大小来限制$|CS_U(P)|$的大小，则可以有效地把$tr_p(ct_U(P),U)$的计算变得可以接受。

以下通过一个例子直观地展示命题4.9的效果。

例 4.2： 继续使用例4.1中的逻辑程序，令$U=\{a,f\}$，则根据定义可以计算得到$CS_U(P)=\{\{a,f\}\}$，并有：

$$ct_U(P)=\{b \leftarrow c. c \leftarrow d. d \leftarrow not\, e. e \leftarrow not\, d. \leftarrow a. x_{\{a,f\}} \leftarrow b. c \leftarrow x_{\{a,f\}}, a. \leftarrow not\, x_{\{a,f\}}.\} \tag{4.17}$$

则有：

$$tr_p(ct_U(P),U)=\{b \leftarrow c. c \leftarrow d. d \leftarrow not\, e. e \leftarrow not\, d. x_{\{a,f\}} \leftarrow b. c \leftarrow x_{\{a,f\}}. \leftarrow not\, x_{\{a,f\}}.\} \tag{4.18}$$

通过ASP求解器可以计算得到：

$$\Gamma(tr_p(ct_U(P),U)=\{b,c,d,x_{\{a,f\}}\} \tag{4.19}$$

则有：

$$(\Gamma(tr_p(ct_U(P),U))\cap Atoms(P))\cap U=\{a,b,c,d,f\} \tag{4.20}$$

结果等于原程序的回答集。

事实上，还能使用通过程序结论找出更具代表性的原子集去化简逻辑程序。下一节将顺着这个思路，给出可靠集的概念，并给出将其应用到程序化简中的理论。

4.2 可靠集

上一节中提及良序模型（$Well-founded\ Model$）被当前主流的ASP求解器用作程序化简的文字集，良序模型本身也是一个程序结论，而Chen和Ji在2013年时就指出一些比良序模型要大的程序结论将能更有效地化简程序。本小节将基于这个思路，找出比良序模型更大的程序结论，并以此进行程序化简，同时将结论扩展到析取逻辑程序。

4.2.1 可靠集定义及应用

本小节将给出可靠集的定义，以及其在程序化简上的可用性结论。

4.2.2 转换算法数据结构

4.2.3 翻译模块数据结构

4.3 主要算法实现

4.3.1 解析模块

4.4 本章小结

基于第三章翻译理论，本章分模块详细介绍了原型求解器cfo2lp的实现和核心翻译算法。本章设计了一阶限定的问题编码描述，经过解析，再经过限定翻译，输出析取逻辑程序，在使用回答集程序求解器。本章的求解器能够处理一阶稳定理论、一阶并行限定和优先级限定的翻译，并且实现了在后继结构下任意公式的翻译优化算法，即存在量词的消去。

第 5 章　实验结果与分析

5.1　实验概述

本文将进行两方面的实验，一是求解器正确性的验证，该部分将对求解器分模块的验证算法正确性；二是求解器效率分析，该部分实验分析了效率的影响因素及不同求解器的比较。

对于求解器的实现，本文使用C/C++语言编写，需要用到工具包括：编译器为gcc version 4.4.3、词法分析器为flex 2.5.35、语法分析器为bison 2.4.1、gringo 3.0.3、ClaspD 1.1.1、DLV build BEN、lparse 1.1.2、GnT2。实验的环境如表5.1所示。

表 5.1: 实验环境

环境	描述
处理器	AMD A10-5800K 3.8GHz
内存	8GB
系统	Linux Ubuntu 12.04LTS 64bit

5.2　正确性验证

本部分实验将会选取计算机科学的经典的问题和限定理论研究领域中出现较多的经典例子作为正确性验证的测例。

5.2.1　最小集合覆盖

最小集合覆盖问题是一个经典的数学问题，是Karp的21个NP-complete问题之一[34]，实验选取这个问题来验证稳定翻译模块的正确性。下面是该问题的描述：

给定一个包含索引的集合 I 和一组集合 $S_i(i \in I)$ ，称一个集合覆盖表示 I 的一个子集 J ，满足所有出现在 $S_i(i \in I)$ 中的元素 a 都出现在 $S_j(j \in J)$ 中。这个

问题可以进行如下编码：

$$\forall\, x\; (Cvr(x) \rightarrow Idx(x)) \tag{5.1}$$

$$\forall x\, \forall y\; (Idx(x) \wedge Set(x,y) \rightarrow \exists\, z(Cvr(z) \wedge Set(z,y))) \tag{5.2}$$

其中$Cvr(x)$表示x是被覆盖的，$Idx(x)$表示x是一个集合，$Set(x,y)$表示元素y 属于x。设φ是上述两个公式的合取，则该问题等价于求解**SM**$[\varphi; cvr]$。公式(5.2)可以运用算法Tr_{QE} 消去存在量词，结果如下：

$$Cvr(x)\ \rightarrow\ Idx(x) \tag{5.3}$$

$$(succ(z,u)\ \wedge\ S(x,y,u))\ \vee\ \theta^{\neg\neg}(x,y,z)\ \rightarrow\ S(x,y,z) \tag{5.4}$$

$$\neg\neg S(x,y,min) \tag{5.5}$$

$$\neg(succ(z,y)\ \wedge\ S(x,y,u))\ \wedge\ S(x,y,z)\ \rightarrow\ (T(x,y,max)\ \leftrightarrow\ \theta(x,y,z)) \tag{5.6}$$

$$succ(z,u)\ \rightarrow\ (T(x,y,z)\ \leftrightarrow\ \theta(x,y,u)\ \vee\ T(x,y,u)) \tag{5.7}$$

$$T(x,y,min)\ \vee\ \theta(x,y,min) \tag{5.8}$$

其中

$$\theta(x,y,z)\ =\ Idx(x)\ \wedge\ Set(x,y)\ \rightarrow\ Cvr(z)\ \wedge\ Set(z,y) \tag{5.9}$$

$$\theta^{\neg\neg}(x,y,z)\ =\ Idx(x)\ \wedge\ Set(x,y)\ \rightarrow\ \neg\neg Cvr(z)\ \wedge\ Set(z,y) \tag{5.10}$$

将θ和$\theta^{\neg\neg}$带入上述公式后，再经过算法**??**-ConvertToDLP得到问题的DLP。对于实际的测试用例，实验中为每一个实例随机生成了一组集合，生成格式如下（以claspD为例）：

```
%claspd的输入 4个集合 4个元素 可随机生成
idx(1..4).e(1..4).
set(1,1).set(1,2).set(2,2).set(2,3).
set(3,1).set(3,4).set(4,2).set(4,3).
min1(1).max1(4).%后继结构的最小最大元素
```

5.2.2 积木世界

（Lifschitz的例子[35]）设φ为如下公式的全称闭包：

$$Block(x)\ \wedge\ \neg Ab(x)\ \rightarrow\ Ontable(x) \tag{5.11}$$

$$\neg Ontable(B_1) \tag{5.12}$$

其中谓词$Block(x)$表示x是一个积木块，$Ontable(x)$表示x在桌子上，$Ab(x)$表示x是不正常的，对应的限定理论是$\mathbf{CIRC}[\varphi; Ab; Ontable]$。根据上述定义，得到如下转换：

$\varphi^{\neg\neg}\ \Leftrightarrow$

$$\theta_1^{\neg\neg}\ =\ \neg Block(x)\ \vee\ \neg\neg Ab(x)\ \vee\ Ontable(x) \tag{5.13}$$

$$\theta_2^{\neg\neg}\ =\ \neg Ontable(B_1) \tag{5.14}$$

$\tilde{\varphi}\ \Leftrightarrow$

$$\tilde{\theta}_1\ =\ \neg Block(x)\ \vee\ Ab(x)\ \vee\ Q_{On}(x) \tag{5.15}$$

$$\tilde{\theta}_2\ =\ Q_{On}(x)\ \rightarrow\ \gamma \tag{5.16}$$

$\phi\ \Leftrightarrow$

$$(\gamma\ \rightarrow\ Ab(x)\ \vee\ \neg Ab(x))\ \wedge\ Tr_{QE}(\exists(Ab(x)\ \vee\ \neg Ab(x)\ \rightarrow\ \gamma)) \tag{5.17}$$

$$\gamma\ \rightarrow\ Q_{On}(x) \tag{5.18}$$

设ψ为公式(5.13-5.18) 合取的一阶全称闭包，则$\exists Q_{On}\exists\gamma\mathbf{SM}[\psi; Ab, Q_{On}, \gamma]$ 等价于$\mathbf{CIRC}[\varphi; Ab; Ontable]$。根据文献[35]，该问题的结果应为$Ab(x)\ \equiv\ x\ =\ B_1$。对$\psi$运用算法**??**-ConvertToDLP得到问题的析取逻辑程序。

使用ASP求解器求解时，需输入问题实例，实验中问题实例使用随机生成的方式，格式如下：

```
#const n=5.
block(1..n).
min1_f(1).max1_f(n).% 为生成DLP中的后继结构谓词
```

设$B_1 = 1$，该实验的结果模型只有一个，即谓词Ab只有1满足，谓词$Ontable$除1之外均满足，因此可以编写了一个验证本问题结果的程序，该程序检测每个问题实例结果的谓词满足情况是否正确。实验结果显示本问题求解是正确。

5.2.3 和平主义者

使用和平主义者例**??**，将翻译Tr_v^p应用到该语句上，首先得到如下的转换：

$\varphi^{\neg\neg} \Leftrightarrow$

$$\theta_1^{\neg\neg} = \neg Quaker(x) \vee \neg\neg Ab_1(x) \vee Pacifist(x) \tag{5.19}$$

$$\theta_2^{\neg\neg} = \neg Republican(x) \vee \neg\neg Ab_2(x) \vee \neg Pacifist(x) \tag{5.20}$$

$\tilde{\varphi}_1 \Leftrightarrow$

$$\tilde{\theta}_{11} = \neg Quaker(x) \vee Ab_1(x) \vee Q_P^1(x) \tag{5.21}$$

$$\tilde{\theta}_{21} = \neg Republican(x) \vee Q_{Ab_2}^1(x) \vee (Q_P^1(x) \to \gamma_1) \tag{5.22}$$

$\tilde{\varphi}_2 \Leftrightarrow$

$$\tilde{\theta}_{12} = \neg Quaker(x) \vee \neg\neg Ab_1(x) \vee Q_P^2(x) \tag{5.23}$$

$$\tilde{\theta}_{22} = \neg Republican(x) \vee Ab_2(x) \vee (Q_P^2(x) \to \gamma_2) \tag{5.24}$$

$\phi_1 \Leftrightarrow$

$$(\gamma_1 \to Ab_1(x) \vee \neg Ab_1(x)) \wedge Tr_{QE}(\exists x(Ab_1(x) \vee \neg Ab_1(x) \to \gamma_1)) \tag{5.25}$$

$$\gamma_1 \to Q_P^1(x) \wedge \gamma_1 \to Q_{Ab_2}^1(x) \tag{5.26}$$

$\phi_2 \Leftrightarrow$

$$(\gamma_2 \to Ab_2(x) \vee \neg Ab_2(x)) \wedge Tr_{QE}(\exists x(Ab_2(x) \vee \neg Ab_2(x) \to \gamma_2)) \tag{5.27}$$

$$\gamma_2 \to Q_P^2(x) \tag{5.28}$$

设ψ为公式(5.19-5.28) 合取的一阶全称闭包，则$\mathbf{SM}[\psi; Ab_1, Ab_2, Q_P^1, Q_{Ab_2}^1, Q_P^2, \gamma_1, \gamma_2]$等价于$\mathbf{CIRC}[\varphi; Ab_1 > Ab_2; Pacifist]$。该问题的结果应为$Ab(x) \equiv Quaker(x) \leftrightarrow$

$Republican(x)$。对ψ运用算法**??**-ConvertToDLP得到问题的析取逻辑程序。问题实例使用随机生成的方式，格式如下：

```
d(1..1000).
quaker(1..1000).
republican(5..10).
min1_f_1(1).max1_f_1(1000).% 为生成DLP中后继结构谓词
min1_f_2(5).max1_f_2(10).
```

实验结果只有一个模型，该模型中，谓词Ab满足性等价于$Quaker \leftrightarrow Repulican$，谓词$Pacifist$满足性等价于$Quaker$。同样编写了一个结果验证程序，实验结果显示该问题求解正确。

文献[36]提供了很多限定理论的一些小的应用实例，在实验中也对其中一些进行了测试，在此就不一一叙述了。

由于缺乏现有求解器的对比，在这类实验中，使用了一个验证限定理论求解器结果正确性的技巧：对于一个限定编码输入，像上述例子一样进行手工的翻译后，经过稳定模型求解器得到问题的模型，和直接使用限定求解器翻译求解得到的模型对比，这样便轻松的验证了一阶限定理论实验结果正确与否。

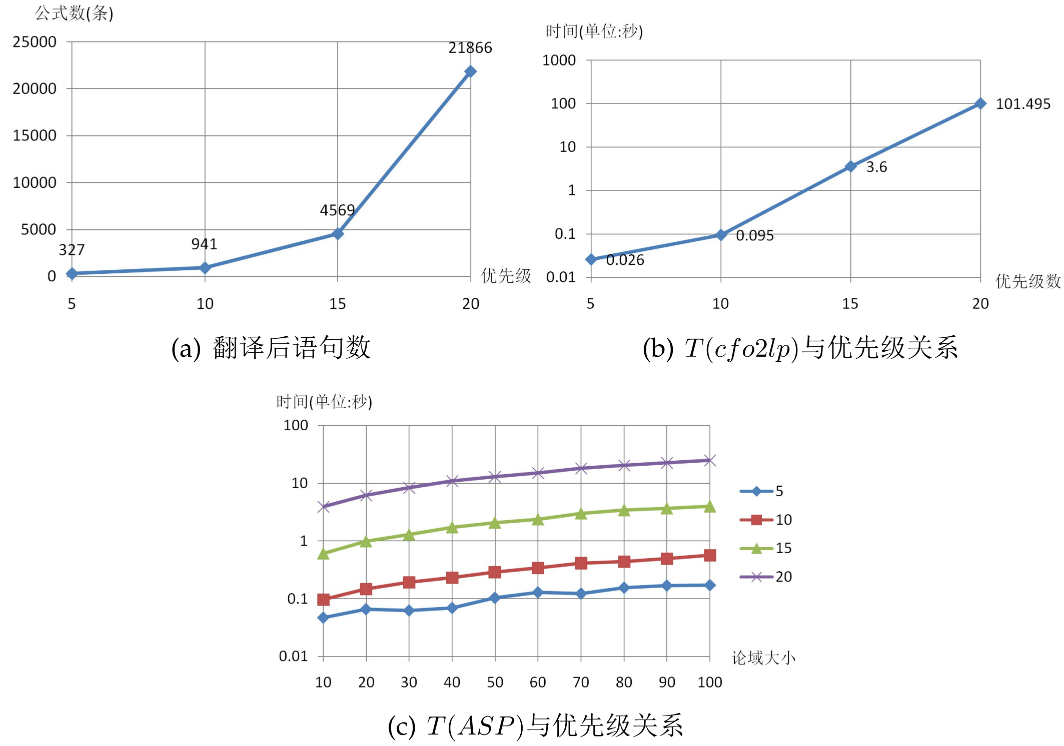
5.3 效率分析

本小节旨在测试求解器的效率，因此本实验选取的测例均为无特殊意义的一阶语句，测量的运行时间是cfo2lp的翻译时间加上求解器ClaspD、DLV或GnT的求解时间，分别记为$T(cfo2lp)$ 和$T(ASP)$。效率问题可以概括为三个方面的因素，分别为：

1. Σ_n-公式（或Π_n-公式）的规模，同时影响$T(cfo2lp)$和$T(ASP)$；

2. 优先级规模，同时影响$T(cfo2lp)$和$T(ASP)$；

3. 问题规模，只影响$T(ASP)$；



(a) 翻译后语句数 (b) $T(cfo2lp)$与优先级关系

(c) $T(ASP)$与优先级关系

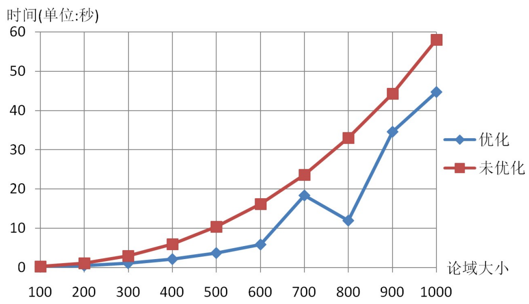
图 5.1: 优先级数对比实验结果

4. 不同求解器实现的机制不同，只会影响$T(ASP)$。

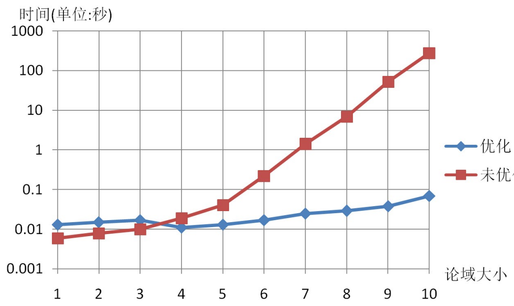
主要分为三个对比实验，分别为优先级数对比实验、翻译优化对比实验和不同求解器的对比实验，第一个实验由于与求解器无关，因此采用ClaspD求解器进行实验，三个实验均采用问题规模作为第二变量因子。

5.3.1 优先级数对比实验

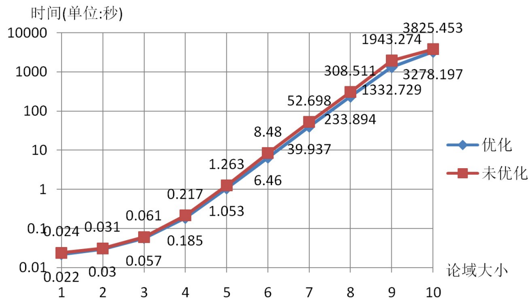
如图5.1为优先级对比实验结果。图5.1(a)显示了优先级数翻译后语句条数的关系，在这个实验例子中，随着优先级的增多，输入公式的复杂程度增大，翻译后语句条数呈现剧增的现象；由图5.1(b)可以看出，随着优先级数的增多，翻译时间接近指数的增长，翻译过程也是经过多次迭代或递归完成，这与优先级计算公式(**??**)复杂度呈指数增长一致；图5.1(c)的数据是使用回答集求解器ClaspD求解得出的，随着问题论域的扩大，翻译出的程序求解时间随之增加。



(a) ClaspD实验结果



(b) DLV实验结果



(c) GnT实验结果

图 5.2: 翻译优化对比实验结果

5.3.2 翻译优化对比实验

本文进行了一系列翻译优化对比实验，不同的公式输入称为问题实例，问题实例采用随机方式生成，生成的公式满足带有存在量词和优先级数小于5（大于5的逻辑程序用时较多，结果较难统计）等特点，这些例子中，$T(cfo2lp)$的翻译时间可以忽略不计，每个问题实例都产生经过翻译优化的逻辑程序和未经过翻译优化的逻辑程序，并分别使用三种求解器在各自适宜的一段n长度的论域上得到时间值，统计各自的平均优化率ρ_{avg}，一个问题实例在一个论域$i(1 \leq i \leq n)$上的优化率ρ_i 计算方法如下：

$$\rho_i = \frac{T(Opt)}{T(Org) - T(Opt)} \tag{5.29}$$

其中$T(Opt)$是优化的逻辑程序时间，$T(Org)$是未优化的逻辑程序时间。平均优化率可以计算如下：

$$\rho_{avg} = \frac{\rho_1 + \rho_2 + \cdots + \rho_n}{n} \tag{5.30}$$

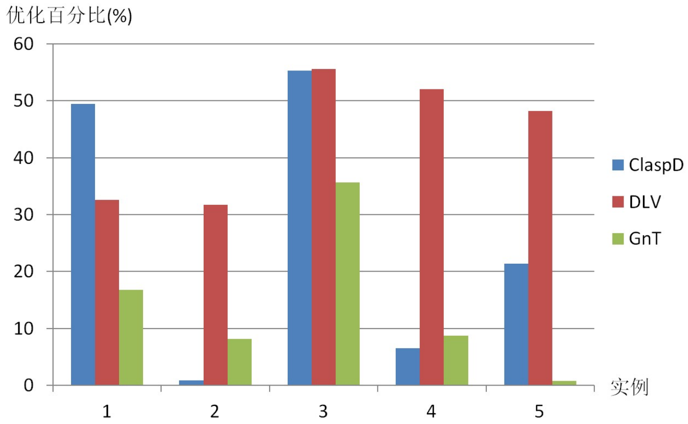


图 5.3: 翻译优化实验结果

下面介绍一个问题实例的统计方法，该例子选取了一个简单的带有存在量词的公式为

$$\forall x(p(x) \ \vee \ (q(x) \ \wedge \ r(x))) \tag{5.31}$$

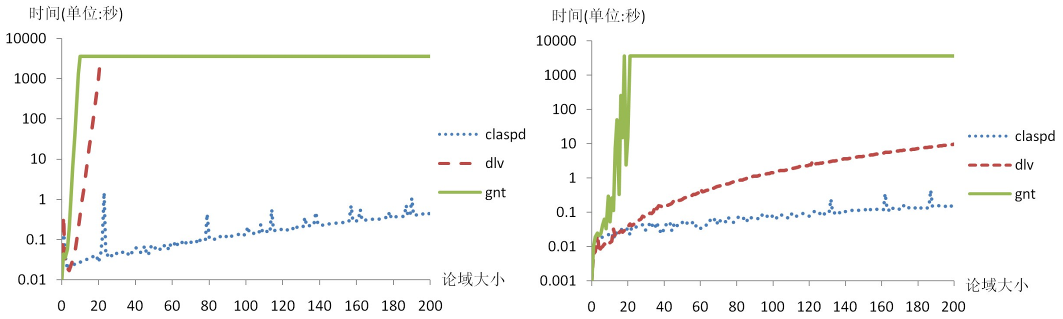
$$\exists x(p(x \ \wedge \ q(x) \ \rightarrow \ r(x))) \tag{5.32}$$

翻译时间可以忽略不计，只记录求解全部模型的回答集求解器时间。由图5.2可以看出，经过翻译优化和未经过翻译优化的时间对比中，在这个实验例子下ClaspD平均可以优化约50% 的时间，DLV在论域较小时基本处于持平，当论域增大时，优化了接近了100%的时间，GnT优化的时间较为平稳，平均约17%的时间；同样观察翻译优化后和与未优化的析取逻辑程序，在翻译后的语句数量上，优化的程序会减少约20%的语句。

翻译优化实验统计结果如图5.3所示，优化率最高可达50%，最低只能达到0.7%左右，不同实例的结果区别较大，不同求解器效率的区别同样较大，两个方面均无规律可循，然而翻译优化的结果总体来说效率是得到提升的。

5.3.3 不同求解器对比实验

如图5.4显示了不同求解器对比实验，实验中，如果求解时间超过一个小时，实验会中止计算，并记录时间为3600秒，表示该论域下的问题不可被对应的求解器求解。其中图5.4(a) 是计算全部模型的时间结果，该实验结果显示ClaspD 在求



(a) 全部模型　　(b) 一个模型

图 5.4: 求解器对比实验结果

解模型中体现了绝对的优势，求解时间接近于0，DLV和GnT 都只能求解出论域比较小的问题，且计算时间也很小，但当论域较大时，DLV 和GnT计算时间急剧增长；图5.4(b)是计算一个模型的时间结果，ClaspD求解时间近乎于0，DLV在求解一个模型上呈现出了较高的效率，然而GnT 同样在论域增至一定量时计算时间急剧增长。

5.3.4 结果分析

基于上述几个对比实验，关于求解器效率方面，主要有如下结论：

1. 并行限定或层数较少的优先级限定的翻译时间很小，优先级数对翻译时间影响较大，呈指数级增长；

2. 限定理论翻译优化对求解效率有一定的提高；

3. 问题规模对求解速度的影响较大，在求解论域较大的问题时，回答集求解器ClaspD性能最佳，DLV 次之，GnT最差；

5.4 本章小结

本章通过对第四章实现的一阶限定理论求解器的两类实验，分别验证了求解器的正确性和分析了求解器效率因素，由于本文是第一个做出一阶限定理论求解器的，因此无法与其他求解器进行对比实验。

第 6 章　应用的研究

本章主要研究限定理论的应用，深入研究了电路诊断问题和稳定婚姻问题。给出了这两个问题的具体定义和一阶限定描述，然后使用前文中实现的求解器翻译成析取逻辑程序，使用程序生成测试用例即问题的论域，然后使用流行的答案集程序求解器求解出问题的模型。

6.1　电路诊断问题

6.1.1　故障诊断背景

随着科技的发展，系统规模越来越大，结构越来越复杂，监视和诊断的难度变大，一旦系统某个部件发生故障，检查出故障所在将是一个难题，因此，智能故障诊断的问题在人工智能领域受到研究者的关注。

上个世纪八十年代，智能诊断系统逐步发展，智能诊断问题成为了一类重要的应用问题。基于模型的诊断被提出，基于Reiter[37]在1987 年对于基于模型诊断的研究，一个普遍的诊断理论被提出，使用一阶逻辑描述电路系统和诊断信息成为了研究方向。该理论的思想如图6.1所示， 可以表述为：根据系统模型预测出系统的预期行为，根据实际的设备监控系统，得到实际系统的观察值，然而如果预期行为与实际观察值之间存在差异的话，说明系统存在异常，从而根据异常得到候选的诊断结果。

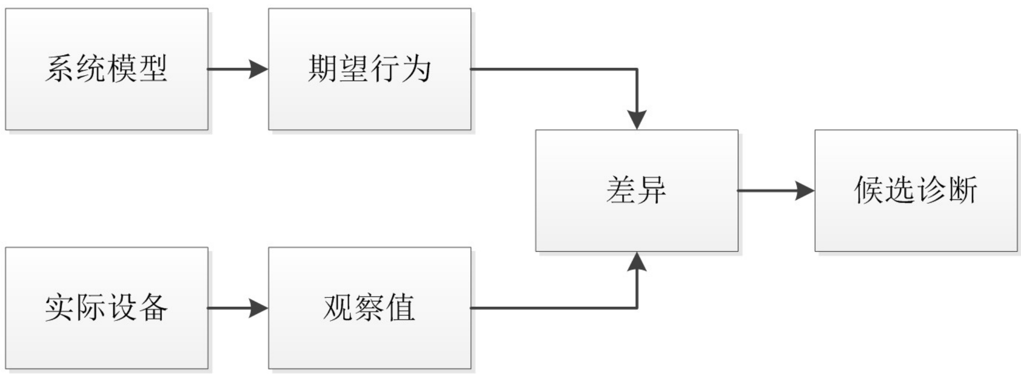


图 6.1: 基于模型的诊断思想

6.1.2 诊断的定义

de Kleer等人在文献[38]进一步发展和完善了基于模型诊断的理论，下面给出诊断问题的一些定义。

定义 6.1： 一个带有观察值的系统（system）是一阶语句的有限集合，记为三元组(SD, $COMP$, OBS)，其中

- SD，即系统描述（system description）,是一阶语句的集合；
- $COMP$，即系统组件(system components)，是包含常量的有限集合；
- OBS，系统的观察值集合（Observations），包含常原子。

$COMP$中每一个元素是一个组件，每一个组件都有一个状态与之关联，该状态可以表示为组件的工作状态，使用谓词Ab表示该组件的状态，$Ab(c)(c \in COMP)$ 表示组件c工作不正常，$\neg Ab(c)$表示组件c正常。

如果一个观察值OBS与所有组件都正常的系统结果冲突，则认为系统(SD, $\{c_1, ..., c_n\}$) 是错误的。系统错误表示某个或某些组件不正常。观察值OBS与系统的正常结果冲突的这一事实可以形式化为公式

$$SD \cup \{\neg Ab(c_1), ..., \neg Ab(c_n)\} \cup OBS \tag{6.1}$$

是不一致的。

直观地，一个电路诊断是关于某些组件错误与否的推测。首先假设$\neg Ab(c_1), ..., \neg Ab(c_n)$，然而解释公式(6.1)不一致性的自然方法是取消足够的假设$\neg Ab(c_1), ..., \neg Ab(c_n)$，以使得公式(6.1)一致。这个过程并未足够，因为这样会产生多个可能的诊断，因此，需要通过增加额外的标准，使得选出一些诊断使其更加符合电路的真实状态。标准主要有如下四种：

- 极小化（minimality）标准：倾向于不正常组件的个数极小的诊断；
- 简约（parsimony）标准：倾向于不正常组件个数尽可能少；
- 免除（exoneration）标准：根据模型免除所有表现正常的组件；
- 可解释性（explainability）标准：倾向于可以解释电路观察值的诊断。

由于这些偏好标准的应用，当引入新的观察值时，被选出的诊断结果不再保持单调的性质，因此，非单调逻辑很好的适用于这样的偏好形式。Reiter[37]使用了缺省逻辑（default logic）形式化极小化标准，Raiman[39] 使用限定理论作为一个方法去解释显示出正常行为的组件。本文主要关注于极小化标准，因为它的极小化诊断与限定理论的极小化模型语义类似。

下面给出诊断的定义：

定义 6.2： $\Delta \subseteq COMP$是$(SD, COMP, OBS)$的一个诊断，当且仅当

$$SD \cup OBS \cup \{\neg Ab(c) \mid c \in COMP - \Delta\} \cup \{Ab(c) \mid c \in \Delta\} \tag{6.2}$$

是一致的。

定义6.2给出了一般诊断的定义。

命题 6.3： 如果Δ是$(SD, COMP, OBS)$的一个诊断，则对于任意的$c_i \in \Delta$，

$$SD \cup OBS \cup \{\neg Ab(c) \mid c \in COMP - \Delta\} \models Ab(c_i) \tag{6.3}$$

命题 6.4： $\Delta \subseteq COMP$是$(SD, COMP, OBS)$的一个极小诊断，当且仅当Δ是一个极小集合，并且

$$SD \cup OBS \cup \{\neg Ab(c) \mid c \in COMP - \Delta\} \tag{6.4}$$

是一致的。

极小子集的含义即是不存在诊断$\Delta' \subset COMP$，使得$\Delta' \subset \Delta$。下文中所讨论的诊断的求解均为极小诊断。

计算所有诊断最基本的算法是一个”生成-测试“机制，基于命题6.4，首先，系统的生成组件的子集Δ，然后，选出极小的Δ候选集，最后，测试公式(6.4)的一致性。然而，当组件数量较多时，这种方法效率极低。文献[37]给出了计算所有诊断的一个有效算法。首先给出几个重要的定义与命题。

定义 6.5： $(SD, COMP, OBS)$的一个冲突集是组件的一个子集$\{c_1, ..., c_k\}$，该集合满足

$$SD \cup OBS \cup \{\neg Ab(c_1), ..., \neg Ab(c_k)\} \tag{6.5}$$

是不一致的。一个冲突集是极小的当且仅当它没有真子集是冲突集。

命题 6.6： $\Delta \subseteq COMP$是$(SD, COMP, OBS)$的一个诊断，当且仅当Δ是满足$COMP - \Delta$不是一个冲突集的极小集合。

定义 6.7： 设C是一组集合，称C的一个压缩集是一个集合H，满足$H \subseteq \bigcup_{S \in C} S$，且对于任意$S \in C$，$H \cap S \neq \emptyset$。$H$是极小的当且仅当不存在$H$的真子集是$C$ 的压缩集。

定理 6.1： $\Delta \subseteq COMP$是$(SD, COMP, OBS)$的一个诊断，当且仅当Δ是$(SD, COMP, OBS)$ 的所有冲突集的一个极小压缩集。

文献[37]提供了计算冲突集和压缩集的方法，并且给出了计算所有诊断的算法。

计算诊断模型最大的挑战就是计算复杂度，基于模型的诊断求解多错误组件诊断模型的复杂度为Σ_2^P-完全[40]。

6.1.3 电路诊断问题求解

本文使用n位串行进位加法器作为电路系统。一个n 位串行进位加法器是由n个全加器组成，一个全加器使用了两个异或门（xor gate）、两个与门（and gate）和一个或门（or gate）。全加器的电路连接如图6.2所示。图中显示A和B为加法器的两个输入数值，C为低位的进位输入，$A \oplus B \oplus C$为加法输出，$(A \oplus B)C + AB$为进位输出。

一个串行进位加法器如图6.3所示， 从图中可以看出，一个n位串行进位加法器是将n个全加器串联起来，高位全加器的进位输入来自相邻的低位全加器的进位输出，最低位全加器的进位输入为系统输入值。加法器的观察值为该加法器的

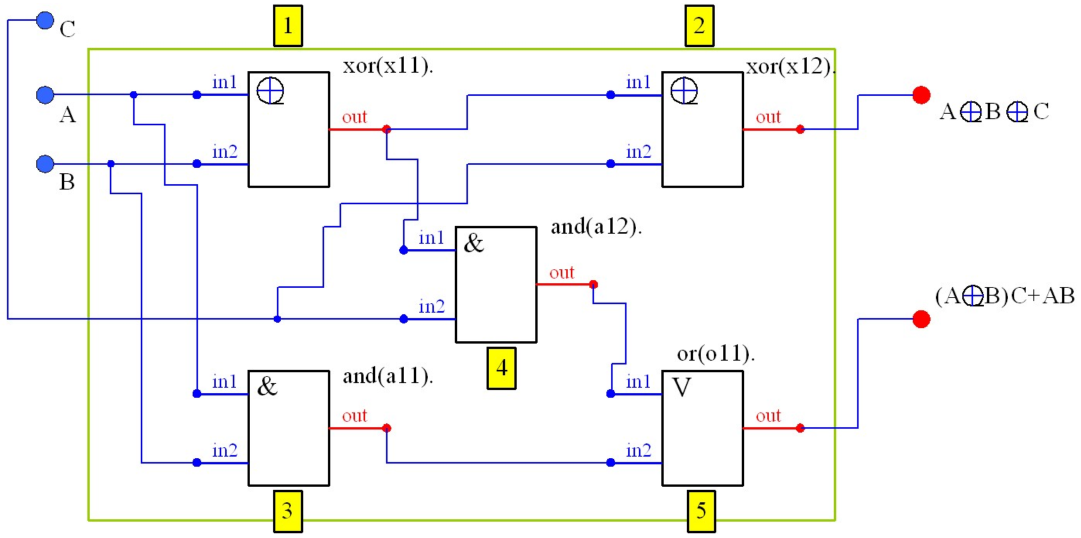


图 6.2: 全加器电路图

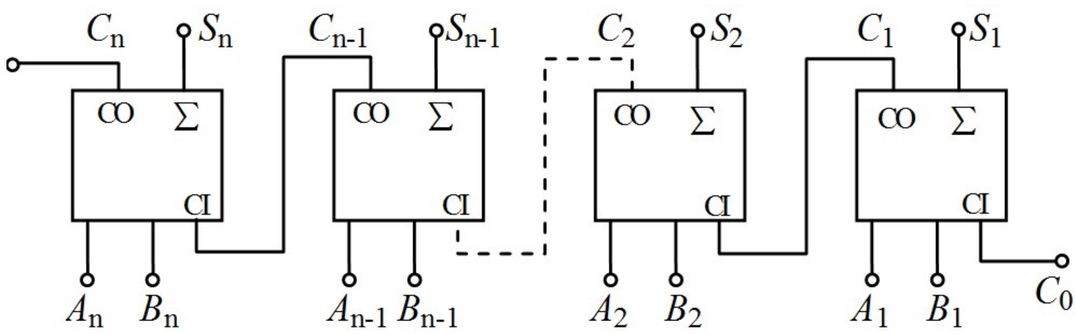


图 6.3: n位串行进位加法器电路图

输入与输出，即对应于图6.3 中接口$A_1, ..., A_n, B_1, ..., B_n, S_1, ..., S_n, C_0, C_n$的输入输出值。

基于工程的经验，每一种电路组件的出错概率是不同的，假设与门的出错的概率最大，其次是异或门，或门的出错概率最小。

在逻辑程序中，一元谓词And、Or和Xor分别表示与门、或门和异或门；使用一元谓词Ab_a表示与门的异常，二元谓词Ab_x表示异或门的异常，二元谓词Ab_o表示或门的异常，在限定的语义下，可以定义优先级$Ab_o > Ab_x > Ab_a$；谓词Ina和Inb表示门的输入，谓词Out表示门的输出，它们均为二元谓词，且第一个元均表示加法器的编号，第二个元均表示加法器中门电路的类型。

设Φ_{SD}为串行进位加法器电路系统描述SD的一阶语句，具体如下：

与门逻辑：

$$\forall i \forall x(And(x) \wedge Ina(i,x) \wedge Inb(i,x) \wedge \neg Ab_a(i,x) \rightarrow Out(i,x)) \tag{6.6}$$

$$\forall i \forall x(And(x) \wedge Ina(i,x) \wedge \neg Inb(i,x) \wedge \neg Ab_a(i,x) \rightarrow \neg Out(i,x)) \tag{6.7}$$

$$\forall i \forall x(And(x) \wedge \neg Ina(i,x) \wedge Inb(i,x) \wedge \neg Ab_a(i,x) \rightarrow \neg Out(i,x)) \tag{6.8}$$

$$\forall i \forall x(And(x) \wedge \neg Ina(i,x) \wedge \neg Inb(i,x) \wedge \neg Ab_a(i,x) \rightarrow \neg Out(i,x)) \tag{6.9}$$

或门逻辑：

$$\forall i \forall x(Or(x) \wedge Ina(i,x) \wedge Inb(i,x) \wedge \neg Ab_o(i,x) \rightarrow Out(i,x)) \tag{6.10}$$

$$\forall i \forall x(Or(x) \wedge Ina(i,x) \wedge \neg Inb(i,x) \wedge \neg Ab_o(i,x) \rightarrow Out(i,x)) \tag{6.11}$$

$$\forall i \forall x(Or(x) \wedge \neg Ina(i,x) \wedge Inb(i,x) \wedge \neg Ab_o(i,x) \rightarrow Out(i,x)) \tag{6.12}$$

$$\forall i \forall x(Or(x) \wedge \neg Ina(i,x) \wedge \neg Inb(i,x) \wedge \neg Ab_o(i,x) \rightarrow \neg Out(i,x)) \tag{6.13}$$

异或门逻辑：

$$\forall i \forall x(Xor(x) \wedge Ina(i,x) \wedge Inb(i,x) \wedge \neg Ab_x(i,x) \rightarrow \neg Out(i,x)) \tag{6.14}$$

$$\forall i \forall x(Xor(x) \wedge Ina(i,x) \wedge \neg Inb(i,x) \wedge \neg Ab_x(i,x) \rightarrow Out(i,x)) \tag{6.15}$$

$$\forall i \forall x(Xor(x) \wedge \neg Ina(i,x) \wedge Inb(i,x) \wedge \neg Ab_x(i,x) \rightarrow Out(i,x)) \tag{6.16}$$

$$\forall i \forall x(Xor(x) \wedge \neg Ina(i,x) \wedge \neg Inb(i,x) \wedge \neg Ab_x(i,x) \rightarrow \neg Out(i,x)) \tag{6.17}$$

单个全加器电路结构逻辑：

$$\forall i\ ((Out(i,1) \leftrightarrow Ina(i,4)) \wedge (Out(i,1) \leftrightarrow Ina(i,4))) \tag{6.18}$$

$$\forall i\ ((Out(i,4) \leftrightarrow Ina(i,5)) \wedge (Inb(i,4) \leftrightarrow Inb(i,2))) \tag{6.19}$$

$$\forall i\ ((Ina(i,1) \leftrightarrow Ina(i,3)) \wedge (Inb(i,1) \leftrightarrow Inb(i,3))) \tag{6.20}$$

$$\forall i\ (Out(i,3) \leftrightarrow Inb(i,5)) \tag{6.21}$$

串行进位加法器逻辑：

$$\forall i\ \forall j\ (i = j - 1 \rightarrow Inb(j,4) \leftrightarrow Out(i,5)) \tag{6.22}$$

输入输出的基本逻辑：

$$\forall i \forall x((Ina(i,x) \lor \neg Ina(i,x)) \land (Inb(i,x) \lor \neg Inb(i,x)) \\ \land (Out(i,x) \lor \neg Out(i,x))) \quad (6.23)$$

设Φ_{COMP}为组件$COMP$的一阶语句，具体如下：

$$\forall i\ (Xor(i,1) \land Xor(i,2) \land And(i,3) \land And(i,4) \land Or(i,5)) \quad (6.24)$$

设Φ_{OBS}为观察值Obs的一阶语句，具体如下是一个观察值的例子：

$$Ina(1,1) \land \neg Inb(1,1) \land \neg Out(1,2) \land \cdots \land Out(n,2) \land Out(n,5) \quad (6.25)$$

其中变元i和j均表示加法器的编号，变元x表示加法器中门的编号。

为了求出该电路的极小诊断模型，可以将该问题编码成优先级限定，直观地，可以将出错概率关系映射成为优先级关系，Ab_o、Ab_x 和Ab_a 作为极小化（$minimaize$）谓词，Ina、Inb和Out作为可变的（$varied$）谓词，该问题即为求解$\mathbf{CIRC}[\Phi_{SD} \cup \Phi_{COMP} \cup \Phi_{OBS}; Ab_o > Ab_x > Ab_a; Ina, Inb, Out]$ 的模型。

6.2 稳定婚姻问题

6.2.1 基本稳定婚姻问题

Gale和Shapley[41]首先提出了稳定婚姻问题（记为SMP），随后的几十年中，SMP 从开始一个有趣的数学问题，延伸成为了一个重要的应用问题，被广泛研究。

Marek等人[42]研究了稳定婚姻问题与非单调逻辑之间的关系，初次用非单调逻辑去描述和解决稳定婚姻问题。Dung[43]将其提出的逻辑程序的特殊形式——论证（argumentation）应用到稳定婚姻问题，进一步印证了稳定婚姻问题可以使用非单调推理解决。

稳定婚姻问题几个重要的概念：

- 偏好列表（Grade List），一个人对异性的喜爱程度的排序，在本文中出现的偏好列表，均按喜爱程度的降序排列；

- 匹配（Matching），是一些人到异性的一一映射；
- 配对（Partner），在一个匹配M中，一个人p映射到另一个人q，称p和q是一个配对，记为(p, q)，则有$M(p) = q$和$M(q) = p$。

基本的稳定婚姻问题（简记为SMP-CLTO）可以如下描述：

定义 6.8： 有n个男人和n个女人，每个人都有其完整且严格有序的偏好列表$Grade$，即包含对所有异性成员的偏好排序。设M是一个包含所有人的匹配。称p和q构成一个阻塞对（$blocking\,pair$），当且仅当下面三个条件同时成立：

1. p和q不是一个配对；
2. p喜欢q多于$M(p)$;
3. q喜欢p多于$M(q)$.

如果在一个匹配M中不存在$blocking\,pair$，则M称为一个稳定匹配（$Stable\,Matching$），基本稳定婚姻问题就是找出这样的稳定匹配。

Gale和Shapley证明了在任何情况下都至少存在一个稳定匹配，并且给出了一个计算时间复杂度$O(N^2)$的算法（记为Gale-Shapley算法）计算出一个稳定匹配。算法3为Gale-Shapley算法的流程。很多学者研究了寻找一个稳定匹配的算法，如文献[44]提出的使用回跳法求解，文献[45]提出的一种快速枚举算法，都是在Gale-shapley算法的基础上进行了改进。

下面给出一个具体地例子了解稳定婚姻问题的相关概念。

例 6.1： 有三个男人$\{m_1, m_2, m_3\}$和三个女人$\{w_1, w_2, w_3\}$，男人的偏好列表为：

- $m_1 : w_1 > w_2 > w_3$
- $m_2 : w_2 > w_1 > w_3$
- $m_3 : w_1 > w_2 > w_3$

女人的偏好列表为

Algorithm 3: Gale-Shapley算法

input ：n个男人和n个女人的偏好列表$ManList[n][n], WomanList[n][n]$

output: 一个稳定匹配M

```
queue iqueue;//定义一个未配对男人队列
iqueue.push(man);//将所有男人入队列
while !iqueue.empty() do
    m = iqueue.front();//选取一个未配对的男人
    for i=0;i<n;i++ do
        w = ManList[m][i];//从偏好列表中按排名选取求婚对象
        if status[w] == false then //  求婚对象未配对
            status[w] = true;//设置配对状态
            status[m] = true;
            M ⇐ (m, w);//添加匹配信息
            break;
        end
        else
            m′ = M[w];//找到w的现任对象
            if WomanList[w][m] > WomanList[w][m′] then //  w喜
            欢m多于现在的伴侣m′
                status[m′] = false;//设置配对状态
                status[m] = true;
                iqueue.push(m′);//加入未配对队列
                M[w] = m;//更新配对信息
                break;
            end
        end
    end
end
```

- $w_1: m_2 > m_1 > m_3$
- $w_2: m_1 > m_2 > m_3$
- $w_3: m_1 > m_2 > m_3$

匹配M_1 : $\{(m_1,w_3),(m_2,w_2),(m_3,w_1),(w_3,m_1),(w_2,m_2),(w_1,m_3)\}$ 不是一个稳定匹配，因为(m_1,w_2) 构成了一个阻塞对；

匹配M_2 : $\{(m_1,w_1),(m_2,w_2),(m_3,w_3),(w_1,m_1),(w_2,m_2),(w_3,m_3)\}$ 是一个稳定匹配；

匹配M_3 : $\{(m_1,w_2),(m_2,w_1),(m_3,w_3),(w_2,m_1),(w_1,m_2),(w_3,m_3)\}$ 也是一个稳定匹配。

稳定婚姻问题可以存在很多个稳定匹配，它们之间如何比较也成为了研究的重点。Gale-shapley算法是找到男士最优选择、女士最差选择的一个匹配，即没有其他的稳定匹配比该匹配中任意男士的伴侣更好，而在所有其他的稳定匹配中，所有女士的选择没有比该匹配的选择更差，例6.1的稳定匹配M_2 符合这类匹配。当然，可以稍微修改Gale-shapley算法使其按照单身女士的顺序遍历，最终找到女士最优选择、男士最差选择的一个匹配，例6.1的稳定匹配M_3符合这类匹配。

这种稳定匹配差别的现象体现出，对稳定匹配结果的要求影响着求解的算法，于是产生了很多稳定匹配之间进行比较的方法，以便得到一个更好更适合的稳定匹配。例如，设SM_1和SM_2为两个稳定匹配，使SM_1比SM_2中更多的男人高兴，需满足SM_1中所有男人至少和SM_2 的男人一样高兴，即需要和他更喜欢（至少相同）的女人结婚。现实中可以找到使更多男人（或女人）高兴的稳定婚姻，当然也有一些算法找到更“公平”的稳定婚姻，这里就不作详细介绍了。

6.2.2 扩展稳定婚姻问题

稳定婚姻问题无论在理论上还是实践上成为了一个比较流行的问题，考虑实际应用的例子，上面描述的偏好列表要求全部有序和完整的，这个条件对于大多

数场合下的应用都是过于严格的，因此，研究者们提出了很多种稳定婚姻问题的变种，统称为扩展稳定婚姻问题，下面介绍几种较为常见的扩展问题。

不完备（Incomplete）偏好的稳定婚姻问题（简记为SMP-ILTO），与SMP-CLTO的定义类似，不同的是它允许每个人可以声明一个或多个不能接受的异性，从而每个人的偏好列表是不完整的，但是同样是严格有序的。如果一个人p的偏好列表中不包括q，则(p, q)组成一个$blocking\ pair$。一个稳定匹配M不需要是一个完整的所有人到所有异性的匹配，但是，一个SMP-ILTO问题的所有稳定匹配都是相同大小的，且包含相同的男人和相同的女人[46]。这个问题同样被证明可以在多项式时间内判断是否存在一个稳定匹配[47]，并且如果存在的话就找到一个，通过改变Gale-Shapley算法，可以计算得到一个稳定匹配。

另一个扩展问题是**带有相等偏好（Tie）的稳定婚姻问题**（简记为SMP-CLT），一个人对某些异性的偏好并不是确定的，因此某些人偏好列表中可以存在对一些异性相等的偏好，对于带有相等偏好的SMP的一个稳定匹配，同样要求$blocking\ pair$中必须为严格喜欢，一个稳定匹配中不存在$blocking\ pair$。然而，这样的条件同样没有使问题变得异常困难，并且该问题被证明存在多项式时间复杂度的算法判断和找到一个稳定匹配[47]。

本文主要关注于既包含不完备偏好又包含相等偏好的稳定婚姻问题，记为SMP-ILT。该问题被证明是NP- 完全的[48]。

下面给出一个具体例子了解扩展稳定婚姻问题中的相关概念。

例 6.2： 有两个男人$\{m_1, m_2\}$和两个女人$\{w_1, w_2\}$，男人的偏好列表：

- $m_1: w_1$
- $m_2: w_1 > w_2$

女人的偏好列表：

- $w_1: (m_1\, m_2)$
- $w_2: m_2$

其中w_1对m_1和m_2有相等的偏好。

匹配$M_1: \{(m_2, w_1), (w_1, m_2)\}$ 是一个稳定匹配；

匹配$M_2 : \{(m_1, w_1), (m_2, w_2), (w_1, m_1), (w_2, m_2)\}$ 是一个稳定匹配。

基于上述稳定婚姻问题，本文提出了新的扩展稳定婚姻问题，下面给出该问题的具体定义：

定义 6.9： 问题描述：对于n个男人和n个女人的SMP-ILT问题，设p和q是稳定匹配中的一个配对，若q 在p 的偏好列表的第x个位置，表示**不满意值**为x，如果p不满意值高于$\frac{2}{3}n$，表示极其不满意，记为$H_1(p)$；如果p的不满意值低于$\frac{2}{3}n$，而高于$\frac{1}{3}n$，表示不满意，记为$H_2(p)$，求解的问题是：在优先级$H_1 > H_2$ 的情况下，所有稳定匹配中找出使得不满意的人最少的稳定匹配，称这个问题为**最少不满意的稳定婚姻问题。**

这个问题的实际意义在于能够尽量满足每个人的喜好，而非整体的满意情况，即考虑到由于每个人的接受能力是不同的，有些人可能不能接受排在最后的异性，有些人则可以接受，这样，本文中这个定义可以给出多个稳定匹配的方案可供选择，最终可以选择一种每个人都尽量符合自己接受范围的一种匹配。

下面本文将该扩展稳定婚姻问题应用于限定理论上，谓词$List(x, p, y)$表示y在x偏好列表的第p 个位置，显然，$Like$ 即为偏好列表的描述，它是不完备的和有相同偏好的，即可以同时出现$Like(x, i, y_1)$和$Like(x, i, y_2)$；谓词$Block(x, y)$ 表示x和y 是一个阻塞对；谓词$Ssex(x, y)$ 表示x和y 是相同性别的；谓词$Prefer(x, y, z)$表示x 偏好于y对于z；谓词$Pair(x, y)$ 表示x和y是一个配对；谓词$H_1(x)$ 表示x 的稳定配对在列表的最后$\frac{1}{3}n$的位置，$H_2(x)$表示x 的稳定配对在列表的中间$\frac{1}{3}n$ 的位置。对这个问题的编码如下：

$$\forall x \forall y \neg Block(x, y) \quad (6.26)$$

$$\forall x \forall y (Pair(x, y) \leftrightarrow Pair(x, y)) \quad (6.27)$$

$$\forall x \forall y (Ssex(x, y) \rightarrow \neg Pair(x, y)) \quad (6.28)$$

$$\forall x \exists y \exists p (Pair(x, y) \wedge Like(x, p, y)) \quad (6.29)$$

$$\forall x \forall y \forall z (y \neq z \rightarrow \neg(Pair(x, y) \wedge Pair(x, z))) \quad (6.30)$$

$$\forall x \forall y \forall z_1 \forall z_2 (Pair(x, z_1) \wedge Pair(y, z_2) \wedge \neg Pair(x, y) \wedge$$
$$Prefer(x, y, z_1) \wedge Prefer(y, z, x_2) \rightarrow Block(x, y)) \quad (6.31)$$
$$\forall x \forall y \forall p (Pair(x, y) \wedge List(x, p, y) \wedge p > (2n/3) \rightarrow Hate_1(x)) \quad (6.32)$$
$$\forall x \forall y \forall p (Pair(x, y) \wedge List(x, p, y) \wedge (n/3) < p \leq (2n/3) \rightarrow Hate_2(x)) \quad (6.33)$$

上述公式的直观含义：公式(6.26) 表示不存在阻塞对；公式(6.27) 表示x 与y配对和y与x配对是同时发生的；公式(6.28) 表示相同性别的人不能配对；公式(6.29) 表示对于所有的人都存在与自己可接受的人的配对；公式(6.30)表示一个人只能与剩余的一个人配对；公式(6.31)描述阻塞对的形成条件；公式(6.32) 表示如果一个人的配偶出现在偏好列表的末位$\frac{1}{3}n$ 的位置，形成不满意条件$Hate_1$；公式(6.33)表示如果一个人的配偶出现在偏好列表的中间$\frac{1}{3}n$ 的位置，形成不满意条件$Hate_2$。由于公式(6.29) 使用了存在量词，大大简化了对该问题的逻辑描述。

设φ为上述公式的合取，可以通过计算$\mathbf{CIRC}[\varphi; Hate_1 > Hate_2; Pair]$ 的模型获得最少不满意的稳定婚姻问题的结果。

6.3 实验结果

该部分实验延续第五章的实验环境。对于这两个问题，为每个问题各选取了5个实例，每个实例均计算了5次取得平均值，得到如表6.1和表6.2 实验结果。其中表6.1 的参数n 表示全加器的个数，该实验测试了范围为1到40个全加器，即5到200 个门，对于每个n，用一个随机算法生成5个n位串行加法器的观察值，对于每个实例，都采用了三种方式。表6.2中参数n表示人的个数，因为本文扩展的稳定婚姻问题的特点，选取了6 到72 的测试范围，对于每个n，同样用一个随机算法生成5个偏好列表。

在实验结果表中，需要说明的是，表格中的实数数据均为对应的回答集求解器计算得到第一个稳定模型的计算时间，真正的限定求解器翻译时间很小，可以微乎不计，然而，如果求解器在一个小时内仍没有返回任何结果，则在实验中将中断该实例的计算，即表格中显示“$> 1h$”的表示计算结果均大于一个小时。表中结果同样显示了求解器ClaspD比DLV和GnT都要快。

表 6.1: 电路诊断问题实验结果

n	实例1			实例2			实例3			实例4			实例5		
	cfo2lp+ ClaspD	cfo2lp+ DLV	cfo2lp+ GNT	cfo2lp+ ClaspD	cfo2lp+ DLV	cfo2lp+ GNT	cfo2lp+ ClaspD	cfo2lp+ DLV	cfo2lp+ GNT	cfo2lp+ ClaspD	cfo2lp+ DLV	cfo2lp+ GNT	cfo2lp+ ClaspD	cfo2lp+ DLV	cfo2lp+ GNT
01	0.022	0.032	0.098	0.013	0.020	0.188	0.016	0.020	0.078	0.016	0.017	0.086	0.012	0.026	0.069
02	0.021	0.236	1.502	0.020	0.167	59.137	0.018	0.163	10.074	0.018	0.209	27.301	0.018	0.261	13.189
04	0.065	$> 1h$	$> 1h$	0.047	$> 1h$	$> 1h$	0.049	$> 1h$	$> 1h$	0.068	$> 1h$	$> 1h$	0.044	$> 1h$	$> 1h$
08	0.127	$> 1h$	$> 1h$	0.105	$> 1h$	$> 1h$	0.081	$> 1h$	$> 1h$	0.096	$> 1h$	$> 1h$	0.059	$> 1h$	$> 1h$
12	0.168	$> 1h$	$> 1h$	0.168	$> 1h$	$> 1h$	0.191	$> 1h$	$> 1h$	0.222	$> 1h$	$> 1h$	0.182	$> 1h$	$> 1h$
16	0.494	$> 1h$	$> 1h$	0.262	$> 1h$	$> 1h$	0.312	$> 1h$	$> 1h$	0.344	$> 1h$	$> 1h$	0.272	$> 1h$	$> 1h$
20	1.350	$> 1h$	$> 1h$	0.265	$> 1h$	$> 1h$	1.826	$> 1h$	$> 1h$	0.703	$> 1h$	$> 1h$	0.820	$> 1h$	$> 1h$
24	2.885	$> 1h$	$> 1h$	2.339	$> 1h$	$> 1h$	5.675	$> 1h$	$> 1h$	1.828	$> 1h$	$> 1h$	1.987	$> 1h$	$> 1h$
28	1.099	$> 1h$	$> 1h$	3.726	$> 1h$	$> 1h$	1.619	$> 1h$	$> 1h$	4.180	$> 1h$	$> 1h$	0.628	$> 1h$	$> 1h$
32	1.253	$> 1h$	$> 1h$	8.679	$> 1h$	$> 1h$	30.718	$> 1h$	$> 1h$	394.678	$> 1h$	$> 1h$	19.055	$> 1h$	$> 1h$
36	44.386	$> 1h$	$> 1h$	66.565	$> 1h$	$> 1h$	1.153	$> 1h$	$> 1h$	10.982	$> 1h$	$> 1h$	155.838	$> 1h$	$> 1h$
40	253.105	$> 1h$	$> 1h$	222.552	$> 1h$	$> 1h$	623.243	$> 1h$	$> 1h$	285.055	$> 1h$	$> 1h$	37.580	$> 1h$	$> 1h$

表 6.2: 扩展稳定婚姻问题实验结果

n	实例1			实例2			实例3			实例4			实例5		
	cfo2lp+ ClaspD	cfo2lp+ DLV	cfo2lp+ GNT	cfo2lp+ ClaspD	cfo2lp+ DLV	cfo2lp+ GNT	cfo2lp+ ClaspD	cfo2lp+ DLV	cfo2lp+ GNT	cfo2lp+ ClaspD	cfo2lp+ DLV	cfo2lp+ GNT	cfo2lp+ ClaspD	cfo2lp+ DLV	cfo2lp+ GNT
06	0.050	0.040	0.165	0.030	0.085	1.275	0.040	0.035	0.170	0.050	0.045	1.610	0.030	0.065	2.120
12	0.195	158.500	98.545	0.195	164.355	$> 1h$	0.170	0.525	15.290	0.180	1.175	$> 1h$	0.200	2.075	$> 1h$
18	0.620	4.650	$> 1h$	1.295	$> 1h$	$> 1h$	0.625	4.635	$> 1h$	0.650	3.215	$> 1h$	0.920	94.135	$> 1h$
24	1.935	$> 1h$	$> 1h$	2.060	381.270	$> 1h$	1.880	$> 1h$	$> 1h$	1.855	$> 1h$	$> 1h$	1.795	500.185	$> 1h$
30	4.430	$> 1h$	$> 1h$	7.620	$> 1h$	$> 1h$	20.995	$> 1h$	$> 1h$	4.585	$> 1h$	$> 1h$	12.155	$> 1h$	$> 1h$
36	10.520	$> 1h$	$> 1h$	11.475	$> 1h$	$> 1h$	14.825	$> 1h$	$> 1h$	10.260	$> 1h$	$> 1h$	10.140	105.845	$> 1h$
42	22.940	$> 1h$	$> 1h$	22.760	256.480	$> 1h$	19.900	255.120	$> 1h$	22.395	$> 1h$	$> 1h$	20.400	$> 1h$	$> 1h$
48	482.930	$> 1h$	$> 1h$	44.670	510.360	$> 1h$	153.115	$> 1h$	$> 1h$	40.660	$> 1h$	$> 1h$	232.315	$> 1h$	$> 1h$
54	235.290	$> 1h$	$> 1h$	143.035	$> 1h$	$> 1h$	1127.490	$> 1h$	$> 1h$	115.360	$> 1h$	$> 1h$	101.035	$> 1h$	$> 1h$
60	456.000	$> 1h$	$> 1h$	336.495	$> 1h$	$> 1h$	260.555	$> 1h$	$> 1h$	1148.390	$> 1h$	$> 1h$	258.205	$> 1h$	$> 1h$
66	509.970	$> 1h$	$> 1h$	627.590	$> 1h$	$> 1h$	539.275	$> 1h$	$> 1h$	505.020	$> 1h$	$> 1h$	623.210	$> 1h$	$> 1h$
72	1027.565	$> 1h$	$> 1h$	2028.780	$> 1h$	$> 1h$	2564.765	$> 1h$	$> 1h$	1158.345	$> 1h$	$> 1h$	1158.795	$> 1h$	$> 1h$

6.4 本章小结

本章对电路诊断问题和扩展稳定婚姻问题进行了深入研究，根据实际的情况，提出更符合应用场景的问题，使用一阶限定理论对其进行描述并求解，可以得到两方面的结果，第一，这两个问题的计算复杂度都超过了NP，这很好的说明了一阶限定理论求解问题的能力，第二，其他实际问题中，都普遍存在着极小和优先级的概念，对比本文对这两个问题的做法，其他问题也可以类比到使用一阶限定理论去解决，这样就将限定理论应用到更多的问题中。

第 7 章　总结与展望

7.1　本文的工作总结

本文前面章节介绍了关于一阶限定理论求解器实现及应用研究的主要工作，下面简要的总结如下：

本文从非单调逻辑之间可译性研究出发，深入学习了限定理论、稳定模型语义和析取逻辑程序的基础理论，对于一阶限定理论，由于定义中出现二阶量词使得其计算变得异常复杂，在本文的研究中，阐明了一阶限定理论与一阶稳定模型语义之间的关系，提出了在一般结构下，一阶限定理论到一阶稳定模型语义的翻译。实际中，由于只考虑有穷结构下一阶限定理论的计算，本文首先将求解限定理论中极小模型翻译为稳定模型语义下的稳定模型，再通过将稳定模型的求解转换成ASP 问题，从而得到计算一阶限定理论的途径。根据这一求解方法，本文设计并实现了其原型求解器cfo2lp，该求解器基于现有的答集程序求解器，可以计算有穷结构下的一阶限定理论问题。本文设计了若干实验，为验证求解器的正确性与效率相关因素。

由于在本文之前，一阶限定理论求解器并未被实现过，这也导致了限定理论的实际应用的匮乏。在本文最后一部分中，本文研究了电路诊断问题和扩展稳定婚姻问题，并且分别给出了问题模型的建立、一阶限定理论对问题的描述和问题的求解这三个步骤，提供了限定理论求解问题的思路。这两个应用问题的计算复杂度都是超过NP，对于这两个问题的求解也验证了一阶限定理论的计算能力。在多数的问题中，都会出现极小化语义，可以将这样的问题对应到限定理论的极小化谓词语义，并且在对问题的逻辑描述中允许出现存在量词，这样就能能用更为接近自然语言的对问题进行编码与求解。

7.2　未来的工作

本文的翻译理论部分只能够处理带可变谓词的一阶并行和优先级限定理论，这仅是一阶限定理论的片断，限定理论仍然包含很多计算理论欠缺的部分，例如

限定理论中的函词的处理。求解器的设计实现主要偏向于实验性质的，部分算法仍有可以优化的空间，然而要设计出工业应用的求解器，仍然有大量的工作需要完成。未来工作可以补充理论部分，提高求解效率，完善一阶限定理论的求解器。

对于限定理论的应用问题仍需要较多的研究与开发，相信限定理论强大的表述能力与计算能力会应用到很多领域的难题中去。

参考文献

[1] Van Harmelen F, Lifschitz V, Porter B. Handbook of knowledge representation [M]. Elsevier, 2008.

[2] 吉建民. 提高ASP 效率的若干途径及服务机器人上应用[D]. 合肥: 中国科学技术大学, 2010.

[3] Horn A. On sentences which are true of direct unions of algebras [J]. The Journal of Symbolic Logic, 1951, 16(01):14–21.

[4] 刘富春. 关于逻辑程序不动点语义的讨论[J]. 广东工业大学学报, 2005, 22(2):120–124.

[5] Colmeraner A, Kanoui H, Pasero R, et al. Un systeme de communication homme-machine en francais [C].//. Luminy. 1973.

[6] Clark K L, Tärnlund S A. Logic programming [M]. Academic Press New York, 1982.

[7] McCarthy J. Circumscription—a form of nonmonotonic reasoning [J]. & &, 1987.

[8] Clark K L. Negation as failure [J]. Logic and data bases, 1978, 1:293–322.

[9] Gelfond M, Lifschitz V. The stable model semantics for logic programming [C].// Proceedings of the 5th International Conference on Logic programming. vol 161. 1988.

[10] 翟仲毅, 程渤. 回答集程序设计: 理论, 方法, 应用与研究[J]. 2014.

[11] Nogueira M, Balduccini M, Gelfond M, et al. An A-Prolog decision support system for the Space Shuttle [C].// Practical Aspects of Declarative Languages. Springer, 2001: 169–183.

[12] Eiter T, Faber W, Leone N, et al. Answer set planning under action costs [J]. Journal of Artificial Intelligence Research, 2003, 25–71.

[13] Soininen T, Niemelä I, Tiihonen J, et al. Representing Configuration Knowledge With Weight Constraint Rules. [J]. Answer Set Programming, 2001, 1.

[14] Xin L, Fan L, Xingbin B, et al. Answer Set Programming Representation for ER Model [J]. Journal of Computer Research and Development, 2010, 1:025.

[15] 赖河澴, 陈红英, 赖博先, et al. 基于回答集编程的Banks 选举求解方法[J]. 计算机工程, 2013, 39(8):266–269.

[16] Lifschitz V, Turner H. Splitting a Logic Program. [C].// ICLP. vol 94. 1994: 23–37.

[17] Dao-Tran M, Eiter T, Fink M, et al. Modular nonmonotonic logic programming revisited [C].// Logic Programming. Springer, 2009: 145–159.

[18] Gebser M, Kaminski R, Kaufmann B, et al. Engineering an incremental ASP solver [C].// Logic Programming. Springer, 2008: 190–205.

[19] Oikarinen E, Janhunen T. Achieving compositionality of the stable model semantics for smodels programs [J]. Theory and Practice of Logic Programming, 2008, 8(5-6):717–761.

[20] Ferraris P, Lee J, Lifschitz V, et al. Symmetric Splitting in the General Theory of Stable Models. [C].// IJCAI. vol 9. 2009: 797–803.

[21] 李未, 黄雄. 命题逻辑可满足性问题的算法分析[J]. 计算机科学, 1999, 26(3):1–9.

[22] Luger G F. 人工智能: 复杂问题求解的结构和策略: 英文版· [M]. 机械工业出版社, 2003.

[23] Lifschitz V. What Is Answer Set Programming?. [C].// AAAI. vol 8. 2008: 1594–1597.

[24] Reiter R. On closed world data bases [M]. Springer, 1978.

[25] Bonatti P, Calimeri F, Leone N, et al. Answer set programming [C].// A 25-year perspective on logic programming. Springer-Verlag. 2010: 159–182.

[26] Gelfond M, Lifschitz V. Classical negation in logic programs and disjunctive databases [J]. New generation computing, 1991, 9(3):365–385.

[27] Lin F, Zhao Y. ASSAT: Computing answer sets of a logic program by SAT solvers [J]. Artificial Intelligence, 2004, 157(1):115–137.

[28] Lee J, Lifschitz V. Loop formulas for disjunctive logic programs [C].// Logic Programming. Springer, 2003: 451–465.

[29] Gebser M, Lee J, Lierler Y. On elementary loops of logic programs [J]. Theory and Practice of Logic Programming, 2011, 11(06):953–988.

[30] Ji J, Wan H, Xiao P, et al. Elementary Loops Revisited [C].// Twenty-Eighth AAAI Conference on Artificial Intelligence. 2014.

[31] Ji J, Wan H, Xiao P. On Elementary Loops and Proper Loops for Disjunctive Logic Programs [J]. 2015.

[32] Van Gelder A, Ross K A, Schlipf J S. The well-founded semantics for general logic programs [J]. Journal of the ACM (JACM), 1991, 38(3):619–649.

[33] Chen X, Ji J, Lin F. Computing loops with at most one external support rule [J]. ACM Transactions on Computational Logic (TOCL), 2013, 14(1):3.

[34] Karp R M. Reducibility among combinatorial problems [M]. Springer, 1972.

[35] Lifschitz V. Circumscription [C].// Handbook of logic in artificial intelligence and logic programming. vol 3. Oxford University Press, Inc. 1994: 297–352.

[36] McCarthy J. Applications of circumscription to formalizing common-sense knowledge [J]. Artificial Intelligence, 1986, 28(1):89–116.

[37] Reiter R. A theory of diagnosis from first principles [J]. Artificial intelligence, 1987, 32(1):57–95.

[38] De Kleer J, Mackworth A K, Reiter R. Characterizing diagnoses and systems [J]. Artificial Intelligence, 1992, 56(2):197–222.

[39] Raiman O. A circumscribed diagnosis engine [J]. Expert Systems in Engineering Principles and Applications, 1990, 462:90–101.

[40] Eiter T, Gottlob G. The complexity of logic-based abduction [J]. Journal of the ACM (JACM), 1995, 42(1):3–42.

[41] Gale D, Shapley L S. College admissions and the stability of marriage [J]. The American Mathematical Monthly, 1962, 69(1):9–15.

[42] Marek W, Nerode A, Remmel J. A theory of nonmonotonic rule systems I [J]. Annals of Mathematics and Artificial Intelligence, 1990, 1(1):241–273.

[43] Dung P M. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games [J]. Artificial intelligence, 1995, 77(2):321–357.

[44] 郭东亮, 张立臣. 用回跳法求解稳定婚姻问题[J]. 计算机应用研究, 2005, 22(1):59–60.

[45] 宋旭东, 纪秀花. 稳定婚姻匹配问题的一个快速枚举算法[J]. 工程图学学报, 2010, (003):187–192.

[46] Gale D, Sotomayor M. Some remarks on the stable matching problem [J]. Discrete Applied Mathematics, 1985, 11(3):223–232.

[47] Gusfield D, Irving R W. The stable marriage problem: structure and algorithms [M]. MIT press Cambridge, 1989.

[48] Iwama K, Miyazaki S, Morita Y, et al. Stable marriage with incomplete lists and ties [J]. Automata, Languages and Programming, 1999, 1644:706–706.

致 谢

首先感谢我的导师万海老师，对于本文的选题和撰写方面，万海老师都给出建设性的意见，在万老师的悉心指导下，让我在知识表示和限定理论学习中受益匪浅，在此，对老师无私的指导和不厌其烦的帮助表示最衷心的感谢！

其次，我要感谢实验室的萧展豪同学，他给出了翻译理论的证明，奠定了翻译算法的基础，一起共同学习讨论的日子，对我来说是一段美好的经历。我还要感谢林尚武同学，感谢他对于本文格式的检查与帮助。同时感谢硕士期间帮助过我的老师和同学们，你们丰富了我的硕士生涯。

最后，我要感谢我的父母，感谢你们一直以来在我背后的支持！