27-34 PROLOG 扩充 程序设计语言 ⑤

计算机科学1992 Vol.19 No.6

三种PROLOG扩充之比较

贲可荣 王 戟 （国防科技大学，长沙410073） TP312PR

摘 要

nH-PROLOG，SPRF和N-PROLOG是PROLOG的三种典型扩充，本文介绍了它们之间的异同。这三个系统采用不同方法扩充了PROLOG，但是均使用情况分析法做为非Horn推理机制。尽管它们的出发点、目的、表达形式各异，但在使用情况分析法这一点上有着惊人的相似性，这说明它们的推理在本质上是一般的和直观的。它们的区别在于：一个系统的性质要加入另一系统，必须做较大的变形。

1. 引 言

逻辑程序设计语言PROLOG以Horn子句逻辑为基础，将PROLOG扩充到逻辑的更大类上是近年来活跃的研究课题之一。本文介绍的PROLOG的三种扩充：nH-PROLOG (near-Horn PROLOG)，SPRF (Simplified Problem Reduction Format)，N-PROLOG分别由Loveland，Plaisted，Gabbay & Reyle给出。

nH-PROLOG将PROLOG扩充到整个一阶逻辑，保留了PROLOG做为程序设计语言的许多形式和特色。SPRF也是将PROLOG扩充至整个一阶逻辑，但其设计更像一定理证明系统。N-PROLOG是一种基于直觉的扩充，对PROLOG增加了假设蕴涵，但是，在一定受限输入形式下，是古典完全的。研究它们的关系，是为了找出其重要区别。尽管这三个系统的出发点、表达方式各异，但均基于情况分析方法，这一点暗示了有一些重要概念尚待揭示。

为简单起见，在命题一级陈述系统，但均可按通常方式提升到一阶子句。我们将系统陈述为反驳系统，引进专用文字FALSE表示“荒谬”。nH-PROLOG和N-PROLOG原先不是以反驳系统方式给出的，但其转换是直观的。

2. nH-PROLOG

nH-PROLOG扩充PROLOG到整个古典逻辑，力求保持PROLOG的格式和特色。我们将给出nH-PROLOG的一般描述，但侧重讨论称为InH-PROLOG (Inheritance nH-PROLOG)的一种简化形式。实现方面，InH-PROLOG相比原系统，在内部循环速度方面慢一些，但它产生较短的反驳且节省搜索空间。

本文我们采用[1]中关于PROLOG的概念，并假设读者熟悉这些概念。一个nH-PROLOG程序类似PROLOG程序，但有两点例外。第一，子句头允许多于一个文字。我们称这样的子句为多头子句或非Horn子句，头之间用分号隔开，表示析取。第二，所有否定子句(无头子句)加上FALSE做为头。FALSE用来表示“荒谬”，加上FALSE，不改变子句的逻辑内涵。我们将nH-PROLOG看做反驳系统。因此，目标是要说明可由程序子句导出FALSE。

nH-PROLOG推导是一分程序(block)序列$B_1, \cdots, B_n$。每一分程序类似一完整的PROLOG推导。开始的分程序B_1是一个关于FALSE的PROLOG推导，区别仅在于，如果

一目标H_i调用带头文字H_1,…, H_m的非Horn子句，则对这分程序可暂不考虑辅助头(H_1, …, H_{i-1}, H_{i+1}, …, H_m)。换句话说，调用子句看成似以H_i为头的Horn子句。类似PROLOG，如延续部分(continuation)为空，则分程序终止。这里，为了删去被延迟的头，必须继续计算。对每一延迟头，存在一重入点分程序B_i($i>1$)，其任务是删去该头；对典型活动头(distinguished active head)，创建延迟头(deferred head)，注销运算允许它满足(取消)出现在重入点分程序中的任何匹配目标。典型活动头可视为在那子程序中成立的条件事实，并可用于满足那目标。除了注销运算，重入点分程序与初始分程序行为相同。所以，通过非Horn子句的归约可引入别的延迟头。如果我们得到空的延续且遭到注销，则分程序成功；并且删去典型活动头，不再考虑。当某分程序取消了最后一延迟头时，计算终止。要求典型活动头在分程序内使用的剪枝规则，称为注销剪枝规则。

InH-PROLOG区别于nH-PROLOG的主要方面是在分程序中允许多个活动头，这些活动头均具注销能力。除了由延迟头表创立的典型活动头外，重入点分程序从典型活动头延迟过的分程序中，"继承"了所有的活动头。

进行非Horn推理的主要思想是情况分析。如果对某个i，$P\cup\{H_i:\neg L_1, \cdots, L_n\}$不可满足，对所有$j\neq i$，$P\cup\{H_j\}$不可满足，则$P\cup\{H_1, \cdots, H_m:\neg L_1, \cdots, L_n\}$不可满足，这里P是子句集。如上所述，初始分程序是情况$P\cup\{H_i:\neg L_1, \cdots, L_n\}$的一反驳，因为别的头被忽视。重入点分程序是形如$P'\cup\{H_j\}$情况的反驳，因为$H_j$成为典型活动头并用做条件事实，因此，输入子句被分解成带比P'少的非Horn子句的子句集。递归使用这种情况分析，从本质上将输入集分成Horn集情形。除了注销运算外，其处理方式与PROLOG相同。这些类PROLOG反驳的合取组成了输入集的一个InH-PROLOG反驳。具体地说，一个InH-PROLOG反驳的每一行格式如下：

C#A{D}。

这里，C是目标(延续部分)序列，用逗号表示合取(与标准PROLOG延续部分的内涵相同)，A是被称为活动头的文字序列(最左边的称为典型活动头)，{D}是延迟头表，其作用相当于左端终结的栈，符号"#"称为墙(wall)。通常，如果延迟头表为空，可不写出；如果不存在活动头或延迟头，则墙不必写出。

一个InH-PROLOG反驳是上述行的有穷序列，并满足如下性质：

(1) 第一行形如"FALSE"，不存在活动头或延迟头。

(2) 如果第n行含一目标，则最左边的目标是调用目标。(a) 如果调用目标与一活动头匹配，则调用目标取消，第n+1行继承第n行的其余部分，或者 (b) 如果存在子句，其头匹配调用目标，选择一个这样的子句，称为C，第n+1行用C的体替代调用目标，继承第n行的其余部分。如果C是一非Horn子句，每一辅助头被加入延迟头表的左边。

(3) 如果第n行不含目标，延迟头表非空，并且在一重入点子程序情况下，通过典型活动头至少有一个注销出现在子程序中(注销剪枝规则)，那么，重入点行开始，重入点行有唯一目标FALSE，并从减去最左延迟头的前一个分程序的最后一行继承了延迟头表，而最左延迟头成为典型活动头。此外，继承了所有来自分程序中的(在这些分程序中，典型活动头被延迟)活动头。

(4)不含目标及不含延迟头的行是InH-PROLOG的结束行。像别的重入点子程序一样，最后重入点子程序必须服从注销剪枝规则。

为了帮助理解系统，给出一个反驳例子。

例1 考虑下列子句集：

FALSE:-h_1, y.
FALSE:-h_2.
y
h_1, h_2:-y。

上述子句有如下InH-PROLOG反驳（下面的空行将两个分程序分隔开）：

1) FALSE
2) h_1, y
3) y, y# {h_2}
4) y #{h_2}
5) #{h_2}

6) FALSE #h_2
7) h_2 #h_2
8) #h_2

注释：第一分程序中无活动头，第5行的延续部分为空，但延迟头表非空，于是后跟重入点行。第二个分程序有典型活动头h_2，最终，分程序成功地终结。

InH-PROLOG反驳的分程序结构强调情况分析的执行。上面反驳的初始分程序中，非Horn子句好像被看成带头h_1的Horn子句，这分程序是情况P∪{h_1:-y}的反驳，这里P代表别的子句。在重入点分程序中，活动头h_2用做满足目标的一条件事实，此分程序是情况P∪{h_2}的反驳。因为这两种情况的不可满足性蕴涵了初始子句集的不可满足性，所以，这两分程序的合取表示了原集的反驳。

3. SPRF

SPRF采用Gentzen型公理系统，其中相继式形如"Γ→L"。这里Γ是正文字集，L是一个正文字。系统的公理形如"Γ→L"满足L∈Γ。下面给出两条推理规则模式，其中变量遍历所有正文字：

(1) 对每一Horn子句"H:-L_1,…,L_n"，存在一推理规则

$$\frac{\Gamma\to L_1,\cdots,\Gamma\to L_n}{\Gamma\to H}$$

这里Γ是任意正文字集。对体为空，即上述推理规则无上相继式的退化情形，得到"Γ→H"，这里H是一事实。

(2) 对每个非Horn子句"H_1, …, H_m:-L_1, …, L_n"，存在一推理规则

$$\frac{\Gamma\to L_1\cdots\Gamma\to L_n\Gamma,H_1\to U\cdots\Gamma,H_m\to U}{\Gamma\to U}$$

这里Γ是任意正文字集，U是一正文字。

第二规则称为分裂规则，Plaisted注意到分裂规则可限制到一个目标FALSE，即规则中变量U由文字FALSE代替，因为这个限制不影响系统的完全性，所以本文我们假定采用受限形式的分裂规则。Plaisted证明了SPRF的古典完全性和可靠性，即一子句集是不可满足的当且仅当相继式"→FALSE"可由公理通过这些推理规则导出。

例2 考虑例1中子句及其对应的SPRF推理规则（这里给规则编号的目的是为了便于理解反驳过程，Horn规则以H标记，分裂规则以S标记）：

FALSE:-h_1,y$_1$ $\dfrac{\Gamma\to h_1,\Gamma\to y}{\Gamma\to FALSE}H_1$

FALSE:-h_2 $\dfrac{\Gamma\to h_2}{\Gamma\to FALSE}H_2$

y $\dfrac{}{\Gamma\to y}H_3$

h_1,h_2:-y $\dfrac{\Gamma\to y\ \Gamma,h_1\to FALSE\ \Gamma,h_2\to FALSE}{\Gamma\to FALSE}S_1$

这些规则，加上公理，可得如下反驳：

$$\dfrac{\dfrac{H_3}{\to y}\quad\dfrac{\dfrac{h_1\to h_1\quad h_1\to y H_3}{h_1\to FALSE}H_1\quad\dfrac{h_2\to h_2}{h_2\to FALSE}H_2}{\to FALSE}S_1$$

3.1 带延迟的SPRF

SPRF的主要缺陷是，在做反向推导时，不知道用哪一个分裂规则，也不知道分裂规则的次序。尽管在控制搜索中，限制分裂规则到目标FALSE，但在反向链反驳的开始处必须使用分裂规则（此时，FALSE是目标），并且此处无指导选择规则的信息。尽可能推迟使用分裂规则，直到搜索决定它们是有用时才用。换句话说，做反向推导，直到得到一可与一非Horn子句头匹配的目标，接着使用适当的分裂规则。Plaisted引进双相继式概念——允许Gentzen型反驳的"双向"生成——推迟使用分裂规则。

下面，给出分裂规则的变形。不要求分裂规则的下相继式以FALSE为目标，而是允许目标为任何非Horn子句头。除了包含下相继式目标的相继式外，上相继式仍维持原状。对一非Horn子句"H_1；…；H_m:—L_1,…,L_n."有m个如下形式的推理规则：

$$\frac{\Gamma\rightarrow L_1 \cdots \Gamma\rightarrow L_n \quad \Gamma, H_1\rightarrow \text{FALSE} \cdots \overset{\neq k}{\cdots} \Gamma, H_m\rightarrow \text{FALSE}}{\Gamma\rightarrow H_k}$$

其中k=1，…，m。这里"……"表示"$\Gamma, H_k\rightarrow$FALSE"从序列中删去。称这些新规则为延迟分裂规则，因为它们以延迟方式实现SPRF的分裂规则。使用这些延迟分裂规则的SPRF称为带延迟SPRF，记为SPRF-D。

例3 考虑例1的子句与SPRF-D规则的对应（注意Horn子句的对应与SPRF相同）。

FALSE, $-h_1$, y. $\dfrac{\Gamma\rightarrow h_1,\ \Gamma\rightarrow y}{\Gamma\rightarrow \text{FALSE}}\ H_1$

FALSE, $-h_2$. $\dfrac{\Gamma\rightarrow h_2}{\Gamma\rightarrow \text{FALSE}}\ H_2$

y. $\dfrac{}{\Gamma\rightarrow y}\ H_3$

$h_1, h_2, -y.$ $\dfrac{\Gamma\rightarrow y \quad \Gamma, h_2\rightarrow \text{FALSE}}{\Gamma\rightarrow h_1}\ DS_1$

$\dfrac{\Gamma\rightarrow y \quad \Gamma, h_1\rightarrow \text{FALSE}}{\Gamma\rightarrow h_2}\ DS_2$

这些推理规则，附加一些公理，产生如下反驳：

$$\dfrac{\dfrac{\dfrac{}{\rightarrow y}H_3 \quad \dfrac{h_2\rightarrow h_2}{h_2\rightarrow \text{FALSE}}H_2}{\rightarrow h_1}DS_1 \quad \dfrac{}{\rightarrow y}H_3}{\rightarrow \text{FALSE}}H_1$$

下面，我们讨论SPRF-D反驳（例3）与InH-PROLOG反驳（例1）的相似性。

1) FALSE
2) h_1, y
3) y, y　　#{h_2}
4) y　　#{h_2}
5)　　#{h_2}
6) FALSE　　#h_2
7) h_2　　#h_2
8)　　#h_2

$$\dfrac{\dfrac{\dfrac{}{3)\rightarrow y}H_3 \quad \dfrac{7)h_2\rightarrow h_2}{6)h_2\rightarrow \text{FALSE}}H_2}{2)\rightarrow h_1}DS_1 \quad \dfrac{}{4)\rightarrow y}H_3}{1)\rightarrow \text{FALSE}}H_1$$

这个例子说明了两系统的行为几乎相同。SPRF-D公理"$\Gamma\rightarrow$L", $L\in\Gamma$，对应InH-PROLOG中的注销规则（如上面的7）。SPRF-D的Horn规则对应InH-PROLOG中类似PROLOG的归约。SPRF-D反驳中，Horn规则H_1从初始目标FALSE产生子目标h_1和y，正如InH-PROLOG反驳中开始归约产生的子目标。SPRF-D中的延迟分裂规则对应InH-PROLOG中通过非Horn子句的归约。延迟分裂规则产生一种情况，它忽视非Horn子句的辅助头和假设头为条件事实的别的情形。上例中，SPRF-D反驳的1～4相继式对应InH-PROLOG反驳中的第一分程序，6～7相继式对应重入点分程序。

SPRF-D和InH-PROLOG仅有两方面的区别：（1）InH-PROLOG提出了关于归约的特别序，即一个分程序必须做完，另一分程序才开始。（2）InH-PROLOG有注销剪枝规则，在SPRF-D中却没有。与注销剪枝规则等价的规则不难加入SPRF-D中。具体地说，以相继式"A_1，…，$A_j\rightarrow$FALSE"为根的树的每个子树包含公理"A_1，…，$A_j\rightarrow A_i$"。

SPRF-D与InH-PROLOG之间的关系，可用下面形式定理表示。

定理1 如果在SPRF-D反驳中加进注销剪枝规则，则在InH-PROLOG反驳与SPRF-D反驳之间存在双射。

3.2 进一步比较

为了把InH-PROLOG直接与SPRF比较，下面讨论SPRF与SPRF-D的差别。对每一非Horn子句，SPRF仅需一个规则，而InH-PROLOG要多个规则，相比之下，SPRF反驳短一些，因为InH-PROLOG中延迟使用情况分析，从而出现重复步。另一方面，

InH-PROLOG延迟使用情况分析以及注销剪枝规则，这是一强有力的控制。

1988年，Plaisted实现了改进型SPRF，他引进了一新的谓词表示否定，重写每一个非Horn子句，使得有一个头移向体内，并否定它。例如，子句“H_1；H_2:-B”重写成“H_1:-B，not(H_2)”或“H_2:-B，not(H_1)”，但只是两者之一。剩下的非Horn子句仅是“L；not(L)”。它定义了否定谓词，并形成唯一的分裂规则。改进型SPRF使用双相继式概念，允许延迟使用唯一的分裂规则，改进型SPRF将非Horn子句头做了能行排序，使得仅有一个头是可达的，这进一步限制了搜索空间，然而，可以证明，非Horn子句的序与InH-PROLOG中的注销剪枝规则不可比较。事实上，有例子表明，带有序非Horn子句的InH-PROLOG是不完全的，带剪枝规则的改进型SPRF也是不完全的。

4. N-PROLOG

N-PROLOG系统的主要设计目标是扩展PROLOG以处理假设蕴涵，从而允许假设的任意嵌套。从本质上说，它是一种直觉主义系统，亦已证明，它对于正直觉主义逻辑（即不含非的直觉主义逻辑）是完全的和可靠的。然而，如果输入按某种形式受限并且经典定义析取，则N-PROLOG对整个经典逻辑是完全的和可靠的。

N-PROLOG中的相继式形如“P？F”，其中P表示输入公式集以及加入该集的假设，公式F表示询问公式，公式中连接词只有合取(∧)和蕴涵(→)。直观上，成功导出“P？F”，表示询问公式F可由P推出。同上述系统一样，仅考虑反驳，这样，为了证明输入公式集P_N不可满足，需证相继式“P_N？FALSE”成功。N-PROLOG有如下规则：

合取规则 “P？($A_1 \wedge \cdots \wedge A_n$)”成功当且仅当对所有i，“P？$A_i$”成功。

蕴涵规则 “P？(A→B)”成功当且仅当“P＋A？B”成功，其中P＋A理解成将A的合取成分分别加入P。

原子规则 对原子q，“P？q”成功当且仅当下述二者之一成立：(1) q∈P (2)对P中某子句“($C_1 \wedge \cdots \wedge C_n$)→q”，有“(P？($C_1 \wedge \cdots \wedge C_n$)”成功。

由于我们的目标是将N-PROLOG与先前的两个系统相比较，故把输入限制为子句形式，然而，与InH-PROLOG和SPRF不同，N-PROLOG基于{∧,→}，由于Horn子句亦仅用析取和蕴涵书写，故在表示Horn子句上没有问题：子句“$H:-L_1, \cdots, L_n$”可表示为“($L_1 \wedge \cdots \wedge L_n$)→H”。此外，我们假设已加入了新文字FALSE，故所有负子句是以FALSE为头的Horn子句。应当注意到，非Horn子句在直觉主义N-PROLOG中是不能表达的。要表示这些子句需加入析取的经典定义。下面，我们分别用两个不同的表示（经典逻辑等价但直觉主义逻辑不等价）将SPRF和InH-PROLOG嵌入到N-PROLOG中去。

4.1 将SPRF嵌入N-PROLOG

Gabbay在[2]中考虑了非Horn子句的第一种表示。考虑析取的经典定义：

$H_1 \vee H_2 \equiv \neg(\neg(H_1) \wedge \neg(H_2)) \equiv ((H_1 \rightarrow FALSE) \wedge (H_2 \rightarrow FALSE)) \rightarrow FALSE$。

于是，非Horn子句“$H_1, \cdots, H_m:-L_1, \cdots, L_n$”可写成：

$L_1 \wedge \cdots \wedge L_n \rightarrow [((H_1 \rightarrow FALSE) \wedge \cdots \wedge (H_m \rightarrow FALSE)) \rightarrow FALSE]$。

该公式等价于：

$[L_1 \wedge \cdots \wedge L_n \wedge (H_1 \rightarrow FALSE) \wedge \cdots \wedge (H_m \rightarrow FALSE)] \rightarrow FALSE$。

这等价性在古典和直觉主义逻辑中均成立。（注意：这种表示与SPRF中对应的分裂规则结构相似。）

例4 例1中子句集可转变成N-PROLOG公式

FALSE:-h_1, y。$\Rightarrow (h_1 \wedge y) \rightarrow FALSE$

FALSE:-h_2。 $\Rightarrow h_2 \rightarrow FALSE$

y。 $\Rightarrow y$

h_1; h_2:-y。 $\Rightarrow [y \wedge (h_1 \rightarrow FALSE) \wedge (h_2 \rightarrow FALSE)] \rightarrow FALSE$

以该公式集为P_{N1}，可得下述N-PROLOG反驳：

1) P_{N_1}?FALSE

2) P_{N_1}?〔y∧(h₁→FALSE)∧(h₂→FALSE)〕
原子规则(2)

3) P_{N_1}?y, P_{N_1}?(h₁→FALSE), P_{N_1}?(h₂→FALSE)
合取规则

4) P_{N_1}?(h₁→FALSE), P_{N_1}?(h₁→FALSE)
原子规则(1)

5) P_{N_1}+h₁?FALSE, P_{N_1}?(h₂→FALSE)
蕴涵规则

6) P_{N_1}+h₁?(h₁∧y), P_{N_1}?(h₂→FALSE)
原子规则(2)

7) P_{N_1}+h₁?h₁, P_{N_1}+h₁?y, P_{N_1}?(h₂→FALSE)
合取规则

8) P_{N_1}+h₁?y, P_{N_1}?(h₂→FALSE)
原子规则(1)

9) P_{N_1}?(h₂→FALSE)　原子规则(1)

10) P_{N_1}+h₂? FALSE　蕴涵规则

11) P_{N_1}+h₂? h₂　原子规则(2)

□　原子规则(1)

由于限制了输入公式的形式，可以组合N-PROLOG规则以得到一种更简单、紧凑的描述。因为初始目标是FALSE，不允许嵌套合取，遇到的合取目标仅可由第二条原子规则引入。这样，原来先写带合取目标的相继式再应用合取规则，可以用一条直接产生最终结果的原子规则来取代。类似地，可以用一条原子规则来直接产生蕴涵规则的结果。因此，我们得到如下紧凑规则：

原子规则　对于原子q，"P?q"成功当且仅当下述三者之一成立：

(1) q∈P

(2) q是FALSE，且对P中某非Horn公式：

〔L_1∧…∧L_n∧(H_1→FALSE)∧…∧(H_m→FALSE)〕→FALSE

我们有，对所有i，j，"P?L_i"成功，"P+H_j? FALSE"成功。

(3) 对P中某Horn公式"(L_1∧…∧L_n)→q"，我们有，对所有i，"P? L_i"成功

使用这些紧凑规则，给出上述例子的反驳：

1) P_{N_1}? FALSE

2) P_{N_1}? y, P_{N_1}+h₁?FALSE, P_{N_1}+h₂?FALSE
规则(2)

3) P_{N_1}+h₁?FALSE, P_{N_1}+h₂?FALSE
规则(1)

4) P_{N_1}+h₁?h₁, P_{N_1}+h₁? y, P_{N_1}+h₂? FALSE
规则(3)

6) P_{N_1}+h₁? y, P_{N_1}+h₂? FALSE
规则(1)

6) P_{N_1}+h₂? FALSE　规则(1)

7) P_{N_1}+h₂? h₂　规则(3)

□　规则(1)

这个反驳与例2中SPRF反驳是相似的。事实上，在紧凑N-PROLOG反驳行与SPRF反驳相继式之间存在一个映射。下面，我们列出二者间的对应：

$$\dfrac{\dfrac{\dfrac{}{2)\to y}H_3 \quad \dfrac{4)\,h_1\to h_1}{3)h_1\to \text{FALSE}} \quad \dfrac{}{5)h_1\to y}H_3}{1)\to \text{FALSE}}H_1 \quad \dfrac{\dfrac{}{7)h_2\to h_2} \quad }{6)h_2\to \text{FALSE}}H_2 \Big/ S_1$$

其中，规则(1)对应于SPRF的公理，规则(2)对应于SPRF的分裂规则，规则(3)对应SPRF的Horn推理规则。

注意到两个系统的相同行为，可以看到把输入限制到子句形式并为非Horn子句选择这种表示，可将SPRF嵌入N-PROLOG中。这个关系可用下面定理形式化表示：

定理2　紧凑型N-PROLOG反驳和SPRF反驳间存在双射。

4.2 将InH-PROLOG嵌入N-PROLOG

上小节，将非Horn子句"H_1；…；H_m:-L_1,…,L_n"表示为〔L_1∧…∧L_n∧(H_1→FALSE)∧…∧(H_m→FALSE)〕→FALSE。注意到该公式(在经典逻辑而非直觉主义逻辑中)逻辑等价于：

〔L_1∧…∧L_n∧(H_2→FALSE)∧…∧(H_m→FALSE)〕→H_1

为了做比较，我们将用如下m个公式表

示一个非Horn子句：

〔$L_1 \wedge \cdots \wedge L_n \wedge (H_1 \rightarrow FALSE) \wedge \cdots \wedge (H_{k-1} \rightarrow FALSE) \wedge (H_{k+1} \rightarrow FALSE) \wedge \cdots \wedge (H_m \rightarrow FALSE)$〕$\rightarrow H_k$

其中k=1，…，m。（注意：这些表示与SPRF-D的延迟分裂规则结构相似）。显然，这m个公式的合取经典等价于原来的公式。

例5 将例1中的子句转变成N-PROLOG公式。

FALSE:-h_1, y. $\Rightarrow (h_1 \wedge y) \rightarrow FALSE$

FALSE:-h_2. $\Rightarrow h_2 \rightarrow FALSE$

y. $\Rightarrow y$

h_1; h_2:-y. $\Rightarrow$〔$y \wedge (h_2 \rightarrow FALSE)$〕$\rightarrow h_1$

〔$y \wedge (h_1 \rightarrow FALSE)$〕$\rightarrow h_2$

该公式集记为P_{N_2}，有如下N-PROLOG反驳：

1) P_{N_2}? FALSE

2) P_{N_2}? $(h_1 \wedge y)$ 原子规则(2)

3) P_{N_2}? h_1, P_{N_2}? y 合取规则

4) P_{N_2}?〔$y \wedge (h_2 \rightarrow FALSE)$〕, P_{N_2}?y 原子规则(2)

5) P_{N_2}? y, P_{N_2}? $(h_2 \rightarrow FALSE)$, P_{N_2}? y 合取规则

6) P_{N_2}? $(h_2 \rightarrow FALSE)$, P_{N_2}? y 原子规则(1)

7) $P_{N_2}+h_2$? FALSE, P_{N_2}? y 蕴涵规则

8) $P_{N_2}+h_2$? h_2, P_{N_2}? y 原子规则(2)

9) P_{N_2}? y 原子规则(1)

□ 原子规则(1)

如同上小节一样，合取规则和蕴涵规则可结合进原子规则，得到如下紧凑规则：

原子规则 对原子q，“P?q”成功当且仅当下面二者之一成立：

(1) $q \in P$

(2) 对P中某子句“〔$L_1 \wedge \cdots \wedge L_n \wedge (H_1 \rightarrow FALSE) \wedge \cdots \wedge (H_m \rightarrow FALSE)$〕$\rightarrow q$”，我们有，对所有i，j，“P?$L_i$”成功，“P+$H_j$?FALSE”成功。

用紧凑规则对例5给出如下反驳：

1) P_{N_2}? FALSE

2) P_{N_2}? h_1, P_{N_2}? y 规则(2)

3) P_{N_2}? y, $P_{N_2}+h_2$? FALSE, P_{N_2}? y 规则(2)

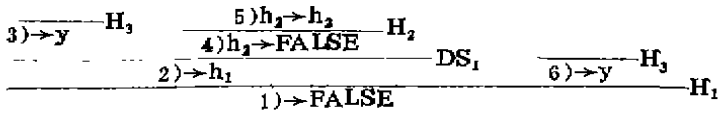
4) $P_{N_2}+h_2$? FALSE, P_{N_2}? y 规则(1)

5) $P_{N_2}+h_2$? h_2, P_{N_2}? y 规则(2)

6) P_{N_2}? y 规则(1)

□ 规则(1)

该反驳和例3给出的SPRF-D反驳相似。紧凑N-PROLOG反驳行与SPRF-D反驳的相继式之间有如下对应：



其中，规则(1)对应于SPRF-D公理，规则(2)对应于SPRF-D的所有推理规则，包括Horn规则和延迟分裂规则。

由两个系统的等同行为，看到以子句形式限制输入并为非Horn子句选择这种表示可将SPRF-D嵌入N-PROLOG中。由于我们已证明InH-PROLOG等同于带注销剪枝规则的SPRF-D，故这种嵌入适用于InH-PROLOG。这里阐述的关系可用如下定理形式化陈述：

定理3 紧凑N-PROLOG反驳与SPRF-D反驳间存在双射。

定理4 加入了注销剪枝规则的紧凑N-PROLOG反驳与InH-PROLOG反驳间存在双射。

5. 结束语

这里描述的三个系统的一些变种已由其作者和同行实现：nH-PROLOG和N-PROLOG作为扩展PROLOG的程序设计语言，SPRF作为一种通用定理证明器。虽然这三种系统从不同的途径被开发，但它们依然是紧密联系的，这表明它们基于的推理方法，尤其是实现非Horn推理的情况分析是一般的和直观的。

尽管它们类似，但这些系统确有本质不同，因此它们有各自的适用领域。我们已看到，InH-PROLOG和SPRF相比，InH-PROLOG有较好的控制搜索，而SPRF有较小的规

计算机科学1992Vol.19No.6

第十四届国际软件工程会议概况

董士海 方 裕 （北京大学，北京100871）
郑国梁 （南京大学，南京210008）
陈海东 （国家教委信息中心，北京100816）

一、简 况

一年一度的软件工程最高水平学术会议——第十四届国际软件工程会议今年5月11—15日在澳大利亚墨尔本"世界议会中心"举行。出席会议的各国代表约五百余名。今年会议的主题是"可信赖的（trusted）计算系统"，丹麦Bjorner教授、美国Leveson博士及澳大利亚Lister教授作了主题发言。会上围绕软件度量、分析与测试、环境支持、软件评估、实时系统、形式方法、软件过程、工具、重用及展望等专题宣读了四十余篇论文。会前分八个题目进行综述讲座，这些题目是：软件可靠性工程、配置管理的环境支持、计算机安全性、需求及设计的CASE工具、总体质量管理及环境中的对象管理。会前联合国教科文组织安排了"亚太地区软件工程研讨会"，国家教委信息中心陈海东同志宣读了论文，介绍了我国软件工程技术的应用及研究情况，受到了好评。会议期间我们与丹麦Bjorner教授（现兼任联合国大学澳

则集。InH-PROLOG与改进型SPRF相比，InH-PROLOG允许注销剪枝规则，而改进型SPRF允许子句头排序。这两个特征均修剪了搜索空间，故选择合适系统依赖于在给定领域中采用何种剪枝更为有用。例如，在使用类PROLOG的子句深度优先搜索时，注销剪枝规则在二者中更有效，故InH-PROLOG作为程序设计语言更具吸引力。然而，在带完全搜索策略的定理证明环境中，改进型SPRF的较小的规则集则有更多的优越性。类似地，N-PROLOG可能是在另一领域中的合适选择，因为它有更一般的输入形式。然而，由于N-PROLOG本质上是一种直觉主义系统，经典的完全性仅能通过将其输入限制至一定形式或扩展该系统才能获得。

参考文献

[1] Clocksin, W.F. and Mellish, C.S., Programming in Prolog, 2nd ed., Springer-Verlag, Berlin, 1984

[2] Gabbay, D.M., N-Prolog: An Extension of Prolog with Hypothetical Implication, Part 2, J. Logic Programming 4: 251—283 (1985)

[3] Gabbay, D.M. and Reyle, U., N-Prolog: An Extension of Prolog with Hypothetical Implication, Part 1, J. Logic Programming 4: 319—355(1984)

[4] Loveland, D.W., Near-Horn Prolog, in: J. Lassez(ed.), Logic Programming: Proceedings of the Fourth International Conference, MIT Press, 1987, pp.456—469

[5] Plaisted, D., A Simplified Problem Reduction Format, Artif. Intell. 18, 227—261(1982)

[6] Reed, D.W., and Loveland, D.W., A Comparison of Three Prolog Extensions, J. Logic Programming 12, 25—50(1992)