PyLog简介

Clarence Protin

2023年6月6日

arXiv:2304.02074v2 [math.LO] 5 Jun 2023

摘要

PyLog 是一款精简的实验性证明助手，基于线性化自然演绎，适用于扩展了概括算子的直觉主义和经典一阶逻辑。PyLog 作为一种工具颇具价值，可与其他更复杂的证明助手及形式化数学项目（如 Coq 及其相关项目）配合使用。基于依赖类型论的证明助手与 PyLog 所采用的方法截然不同，却又通过 Curry-Howard对应有着深刻联系。Coq 的策略系统采用自上而下的证明方法（通过回溯规则，寻找一个满足给定类型的项，并自动完成类型判定和类型推断）；而 PyLog 的经典方法则遵循数学证明通常的书写方式。

PyLog 应沿着 Coq 的路线进一步发展，尤其是通过引入大量“微自动化”和一款出色的集成开发环境。

引言

正如 Voevodsky 在普林斯顿的一场公开讲座中指出的那样，我们非常希望建立一种数学基础，使证明能够得到自动验证。标准方法采用直觉主义高阶逻辑或各种强大的依赖类型理论，这些理论还具有一个额外优势：Curry-Howard同构能够提供构造性计算信息。值得注意的例子包括基于 Coq 和 Agda 的形式化数学项目，以及基于同伦类型论的项目。PyLog 是一种在计算和哲学层面上另辟蹊径的方法，旨在满足若干 desiderata：

1. 用户环境应当易用、简洁、直观且富有吸引力，让逻辑学家或数学家使用起来得心应手；同时对程序员透明，便于其访问数据结构和算法，以支持未来的开发与应用。2. 编写证明的过程（以及检查算法）应当令人愉快，并在结构上贴近实际的数学实践，或者至少遵循《数学原理》（1910年）所确立的范式。3. 系统应采用一阶逻辑（并提供较弱的“参数化”二阶扩展），兼具无类型体系的简洁性与灵活性。4. 不同形式化理论应易于组合，定理也应易于组织到各个理论中。5. 形式化证明应便于人工或机器检查。6. 形式化和检查某一理论的难度不应超过该理论本身的数学难度。

7. 经典逻辑应被视为直觉主义逻辑的一种扩展，用户可以自行决定是否使用经典否定规则。

我提出的关键要素如下：

- 带等号的谓词演算（以及某种较弱的二阶扩展）的线性化自然演绎，并加入一个 Kelley-Morse 风格的扩展算子。
- 它可以轻松地形式化凯利-莫尔斯集合论，但并不局限于此。

PyLog 的逻辑

PyLog 基于带等号的一阶谓词逻辑的自然演绎表述，并为类形成算子 $\{x : P(x)\}$ 和原始二元谓词 $\in$添加了规则。PyLog 还包含二阶变量，使我们能够实例化逻辑有效式。PyLog 的语言由有限个常量、（一阶）变量、二阶变量、不同元数的函数符号 $n > 0$、不同元数的谓词符号$n < 0$ 以及特殊符号 $\bot$组成。项和公式通过互递归定义：

- 常量 c 是一个项。
- 变量 x 是一个项。
- 如果 $t_1, ...t_n$ 是项，且 f 是一个 n元函数符号，那么 $f(t_1, ..., t_n)$ 是一个项。
- 如果 A 是一个公式，且 x 是一个变量，那么 $\{x: A\}$ 是一个项（称为扩展）
- 如果 P是一个 n元谓词符号，且 $t_1, ...t_n$ 是项，那么 $P(t_1, ..., t_n)$是一个公式。
- 如果 t 和 s 是项，那么 $t = s$ 是一个公式（这是上一条条件的一个特例）。
- 如果 $\mathfrak{A}$ 是一个二阶变量，那么它就是一个公式。
- 如果 A 和 B 是公式，那么 $A \vee B$、 $A \,\&\, B$、 $A \to B$ 也是公式。
- 如果 x 是变量，且 A是公式，那么 $\forall x.A$ 和 $\exists x.A$ 也是公式。
- $\bot$ 是公式。

我们定义 表达式 e（项或公式）的自由变量集合 $FV(e)$ 如下 ：

- $FV(c) = \emptyset$
- $FV(x) = \{x\}$
- $FV(f(t_1, ..., t_n)) = \bigcup_{i=1,...,n} FV(t_i)$
- $FV(P(t_1, ..., t_n)) = \bigcup_{i=1,...,n} FV(t_i)$
- $FV(\bot) = \emptyset$
- $FV(\mathfrak{A}) = \emptyset$
- $FV(A \vee B)$, $FV(A \,\&\, B)$ and $FV(A \to B)$ 与 $FV(A) \cup FV(B)$ 相等
- $FV(\forall x.A)$、 $FV(\exists x.A)$ 和 $FV(\{x : A\})$ 等于 $FV(A)/\{x\}$

通常，我们将表达式视为在重命名量化变量或扩展作用域内的变量后仍相同：对于形如 $\forall x.A$, $\exists x.A$ or $\{x\colon A\}$ 的任意子表达式，只要 y 不在某个量词 $\forall y$, $\exists y$ 或扩展 $\{y:\ ...\}$的作用域内，我们就可以将 x 以及 A 中 x 的所有自由出现重命名为一个新的变量 y。当我们写作 $A[t/x]$ 时，我们假定 A 的约束变量已重命名，使其与 $FV(t)$中的变量不同（这比实际所需的条件略强）。

在 PyL og 中，证明始终处于证明环境的上下文中。该环境由以下内容组成 of:

- 一个称为公理的公式列表
- 一个称为假定定理的公式列表
- 一个常量或函数符号的定义方程列表，形式为 $c = A$或$f(x_1, ..., x_n) = A$
- 一个谓词定义列表，由三元组 $(P, (x_1, ..., x_n)\ A)$ 组成，用于定义 n 元谓词符号 P。此处 $FV(A) \subseteq \{x_1, ..., x_n\}$。三元组也记作 $P(x_1, ..., x_n) \equiv A$。

以上所有列表都可以为空。在 PyLog 中，默认的证明环境由一个谓词定义构成，即将 $Set(x)$ 定义为 $\exists y.x \in y$。此外，语言还包含原始二元谓词 $\in$。除此之外，没有其他函数、谓词或常量。

PyLog 的证明系统基于自然演绎的一种线性化变体，并采用保守的二阶阶次扩展。

我们首先以标准形式介绍该系统。我们假定读者熟悉证明树和依赖关系的概念（阅读 [3] 的前几章即可）。PyLog 的证明系统由以下内容构成。我们有纯逻辑规则，即最小谓词演算的规则，以及直觉主义和经典否定规则：

$$\frac{A \qquad B}{A\ \&\ B}\ \text{AndInt} \qquad \frac{A\ \&\ B}{A}\ \text{AndElimL} \qquad \frac{A\ \&\ B}{B}\ \text{AndElimR}$$

$$\frac{\begin{array}{c}[A]\\ B\end{array}}{A \to B}\ \text{ImpInt} \qquad \frac{A \qquad A \to B}{B}\ \text{ImpElim}$$

$$\frac{A}{B \vee A}\ \text{OrIntL} \qquad \frac{A}{A \vee B}\ \text{OrIntR} \qquad \frac{A \vee B \qquad \begin{array}{c}[A]\\ C\end{array} \qquad \begin{array}{c}[B]\\ C\end{array}}{C}\ \text{OrElim}$$

$$\frac{A}{\forall x.A[x/y]}\ \text{ForallInt}_y \qquad \frac{\forall x.A}{A[t/x]}\ \text{ForallElim}$$

$$\frac{A[t/x]}{\exists x.A}\ \text{ExistsInt} \qquad \frac{\exists x.A \qquad \begin{array}{c}[A[y/x]]\\ C\end{array}}{C}\ \text{ExistsElim}$$

$$\frac{\bot}{A}\ \text{Abs}_i \qquad \frac{\begin{array}{c}[\sim A]\\ \bot\end{array}}{A}\ \text{Abs}_c$$

ForallInt 的限制条件是：y 不能出现在 A 所依赖的任何假设中；ExistsElim 的限制条件是：y 不得出现在 $\exists y.A$ 或 C 中，也不得出现在 C 所依赖的任何假设中（$[A[y/x]]$除外）。$\sim A$ 是 $A \to \bot$的语法糖。

类规则如下：

$$\frac{t \in \{x : A\}}{Set(t)\ \&\ A[t/x]}\ \text{ClassElim} \qquad \frac{Set(t)\ \&\ A[t/x]}{t \in \{x : A(x)\}}\ \text{ClassInt}_x$$

这些规则表达了凯利-莫尔斯集合论的分类公理模式，正如 [4] 附录中所表述的那样。

我们还有等同性规则[1]:

$$\frac{}{t = t}\ \text{Identity} \qquad \frac{s = t}{t = s}\ \text{Symmetry} \qquad \frac{A \qquad t = s}{A'}\ \text{EqualitySub}$$

其中，在 EqualitySub 中， A' 是 A，即将 t 的指定数量的出现替换为 s。EqualitySub 在使用证明环境中的定义常量和函数符号时具有根本重要性。

于是我们得到二阶规则:

$$\frac{A}{A'}\ \text{PolySub}_{\mathfrak{A}}$$

其中， A' 是将 A 中 $\mathfrak{A}$ 的所有出现替换为公式 B后，由 A 得到的。限制条件是， $\mathfrak{A}$ 不能出现在 A 所依赖的任何假设中，并且替换后， B 中的任何自由变量都不能变成约束变量。

注释 0.1 该规则应理解为先对变量 $\mathfrak{A}$ 进行不可见的二阶概括，再通过 B进行实例化。

最后一组规则涉及如何将证明环境中的信息引入证明中。

$$\frac{}{A}\ \text{AxInt}_n \qquad \frac{}{A}\ \text{TheoremInt}_n \qquad \frac{}{t = s}\ \text{DefEqInt}_n \qquad \frac{A}{A'}\ \text{DefExp} \qquad \frac{A}{A'}\ \text{DefSub}$$

前三条规则分别只是将公理、假定定理和定义方程列表中的第 n 个公式添加进来。DefExp 的作用如下。假设我们有一个定义$P(x_1, ..., x_n) \equiv B$。那么，DefExp 会将指定位置上形如$P(t_1, ..., t_n)$的子公式替换为 $B[t_1/x_1, ..., t_n/x_n]$ （所得表达式在规则中记作 A'）。DefSub 则执行相反的操作。对于选定的 $t_1, ..., t_n$，我们必须指定 A 中形如 $B[t_1/x_1, ..., t_n/x_n]$ 的表达式出现的位置，并将其“折叠”为 $P(t_1, ..., t_n)$[2]。

当我们要使用定义函数或常量时，首先通过 DefEqInt_n 将定义等式引入证明中，然后使用 EqSub。以上是 PyLog 的核心规则。我们引入常用的缩写 $A \leftrightarrow B$，因此最终有两条规则可用于切换这种记法:

[1]这种做法受到 [5] 中处理方式的启发。例如，若我们有定义 $Set(x) \leftrightarrow \exists y.x \in y$，并且证明中有这样一行:

$$1.Set(x)$$

那么调用 DefExp(n,”集合”,[0]) 就会添加这一行

$$2.\exists y.x \in y$$

另一方面，如果我们有一行

$$3.\exists y.x \in y\ \&\ \exists z.y \in z$$

那么，命令 DefSub(3,”集合”，[”x”],[0]) 将会得到

$$4.Set(x)\ \&\ \exists z.y \in z$$

$$\frac{A \to B \ \& \ B \to A}{A \leftrightarrow B}\ \text{EquivConst} \qquad \frac{A \leftrightarrow B}{A \to B \ \& \ B \to A}\ \text{EquivExp}$$

备注 0.2 某些规则组合经常出现，因此使用派生规则（或快捷规则）会很方便，例如

$$\frac{A \to B \qquad B \to A}{A \leftrightarrow B}\ \text{EquivJoin}$$

for

$$\frac{\dfrac{A \to B \qquad B \to A}{A \to B \ \& \ B \to A}\ \text{AndInt}}{A \leftrightarrow B}\ \text{EquivCont}$$

以及另外两个显而易见的快捷规则 EquivRight 和 EquivLeft。还有重要的派生规则

$$\frac{A}{A[t/y]}\ \text{FreeSub}_{y,t}$$

for

$$\frac{\dfrac{A}{\forall x.A[x/y]}\ \text{ForallInt}_y}{A[t/x]}\ \text{ForallElim}$$

线性化自然演绎

在 PyLog 中，给定一个证明环境，线性证明是由证明元素组成的列表。每个证明元素 p 都是一个三元组（A, par, dis），其中 A 是一个公式，而 par 和 dis 是整数列表。如果 p 出现在位置 m （我们称 m 为 p的编号），且该列表的长度为 m，那么 par （父节点）中的元素必须严格小于 n，而 dis（消解项）中的元素必须严格大于 n。应用规则时，会在列表末尾添加一个新的证明元素，并可能更新先前的证明条目。给定一个线性证明和一个元素 p，我们可以递归回溯其父节点，从而得到一个由证明元素构成的依赖树。通过取这棵树的叶节点集合，并移除那些 p中包含依赖树中某个编号的证明元素，即可得到 p的依赖关系。我们也很容易看出，如何从一棵证明树得到一个线性证明。在 PyLog 中，规则的一般形式如下

$$(Name, Parents, Parameters)$$

其中 Name 是规则的名称，Parents 是该规则所应用到的前面元素的编号列表，Parameters 可以包含公式、项、变量、位置列表等。在 PyLog 中，规则以 Python 函数的形式输入。参数将指定 Parents 和 Parameters 中所需的全部信息。

PyLog 命令 Qed(ForNum) 用于检查公式是否已解除其所有假设。查看一些类定义中的代码片段会有所帮助:

```
class ProofElement:
 def __init__(self,name,dependencies,parameters,discharging,formula):
  self.name = name
  self.dependencies = dependencies
  self.parameters = parameters
```

```
  self.discharging = discharging
  self.formula = formula
  self.dischargedby = []
  self.pos = 0
  self.qed=False
  self.comment=""

class ProofEnvironment:
 def __init__(self,proof,name):
  self.proof = proof
  self.name = name
  self.definitions = {}
  self.definitionequations = []
  self.axioms = []
  self.theorems = []
  self.log = []

 def CheckRange(self, dependencies):
  for dep in dependencies:
   if dep > len(self.proof):
    return False
   return True

 def GetTree(self,proofelement):
  out = [proofelement.pos-1]
  for dep in proofelement.dependencies:
    out = out + self.GetTree(self.proof[dep])
    return out

 def GetHyp(self,proofelement):
   if proofelement.name =="Hyp":
    return [proofelement.pos-1]
   out = []
   for dep in proofelement.dependencies:
     out = out + self.GetHyp(self.proof[dep])
   return out

 def CheckDischargedBy(self, hyp, proofelem):
   if len(self.proof[hyp].dischargedby) ==0:
     return False
   for h-1 in self.proof[hyp].dischargedby:
     if h in self.GetTree(self.proof[proofelem]):
       return True
     return False

 def GetHypDep(self,proofelement):
   aux = []
   for  h in self.GetHyp(proofelement):
     if len(Intersect([x-1 for x in self.proof[h].dischargedby],
      self.GetTree(proofelement))) == 0:
```

```
        aux.append(h)
    return aux

    (...)
```

PyLog核心规则列表

```
Logical Rules
=============

AndInt(ForNum, ForNum)

AndElimL(ForNum)

AndElimR(ForNum)

ImpInt(ForNum, DisNum)

ImpElim(ForNum,ImpNum)

OrIntL(ForNum,formula)

OrIntR(ForNum,formula)

OrElim(OrForNum, LeftHypNum, LeftConNum, RightHypNum, RightConNum)

ForallInt(ForNum, VarName, newVarName)

ForallElim(ForNum, term)

ExistsInt(ForNum, term, newVarName, PositionList)

ExistsElim(ExistsForNum, InstForNum,ConForNum, instVariable)

AbsI(BotForNum)

AbsC(NegForNum, BotConNum)


Class Rules
=========

ClassElim(MemForNum)

ClassInt(ForNum, newVarName)



Equality Rules
==============
```

```
Identity(term)

Symmetry(EqForNum)

EqualitySub(ForNum, EqForNum, PositionList)




Second-Order Rule
=================

PolySub(ForNum, SecondOrderVarName, formula)


Proof Environment Rules
=======================

AxInt(number)

TheoremInt(number)

DefEqInt(number)

DefExp(ForNum, predicateName, PositionList)

DefSub(ForNum, predicateName, ArgList, PositionList)


Other Rules
==========

Qed(ForNum)

EquivConst(ForNum)

EquivExp(ForNum)

EquivLeft(ForNum)

EquivRight(ForNum)

FreeSub(ForNum, VarName, Term)
```

PyLog 命令

我们有一组用于设置证明环境的命令，即用于引入证明中将使用的公理、假定定理、定义常量和符号，以及定义谓词的命令。

```
Hyp(Formula)

NewAx(Formula)

AddPredicate(PredicateName, Arity, PrefixBook)

NewDef(PredicateName, ArgList, Formula)

AddConstants(NameList)

AddFunction(FunctionName, Arity, PrefixBool)

NewDefEq(EquationFormula)

AddTheorem(Formula)
```

Hyp 用于引入一个公式作为假设。

使用 NewDefEq() 定义函数时，必须先使用 AddFunction() 指定函数名称、元数，以及该函数是否采用前缀表示法或中缀表示法（二元函数适用）。传递给 NewDefEq 的参数不能包含最外层括号。例如：

```
NewDefEq("rus = extension z. neg Elem(z,z)")
```

对于常量，我们使用 AddConstant()。还必须确保通过 AddVariables(VarList) 函数提供足够的变量。

接下来是一组命令，用于显示当前证明和证明环境的信息。ShowDefinitions() 显示已定义的谓词。ShowDefEquations() 显示已定义的常量和函数。ShowAxioms() 显示公理。ShowThe- orems() 显示可在证明中使用的假定定理（不建议在证明过程中修改此列表）。ShowProof() 显示当前证明状态，ShowLog() 则显示构成该证明的先前成功规则命令列表。我们还有一个 Undo() 命令，用于删除证明中的最后一个元素。Hypotheses(n) 显示公式 n 所依赖的假设。

如果某个定理已保存，您可以使用 ViewTheorem(Name) 查看其结论，或使用 LoadTheorem(Name) 命令将其直接添加到证明环境中，前提是所需的环境已预先加载。

使用 PyLog

在 PyLog 中，理论是一个目录，其中的文件都是定理。一个定理由证明环境和该环境中的证明组成（完整或未完成）。理论中的所有定理

理想情况下都应使用相同的证明环境。待证明的定理理想情况下应出现在证明的末尾，并且已经使用命令 Qed(Number) 进行测试。此外，还应有一个“空”定理，它被视为开始编写新定理时必须加载的证明环境。命令 Load(Name) 用于加载证明环境或定理，命令 Save(Name) 用于保存当前的证明环境或定理。命令 ViewTheorem(Name) 不会加载任何内容，只会显示证明的最后一行。类似地，命令 ViewTheory(DirName) 会显示该目录中的所有定理。

要使用 Pylog，必须安装 Python 3。∗ PyLog 通过 Python 命令行界面在终端中运行。在 GitHub 上克隆代码仓库[3]，进入该文件夹，然后输入

```
$ python -i proofenvironment.py

Welcome to PyLog 1.0

Natural Deduction Proof Assistant and Proof Checker

(c) 2020  C. Lewis Protin

>>>
```

我们的项目是对 [4]附录中的集合论所有定理进行完整验证的形式化。

在 PyLog 文件夹中，我们保存了 Kelley-Morse 环境。使用 Load(”Kelley- Morse”) 加载该环境。命令成功执行时，PyLog 会返回 True。现在我们可以查看公理和定义：

```
>>> ShowAxioms()
0. ∀x.∀y.((x = y) <-> ∀z.((z ϵ x) <-> (z ϵ y)))
1. Set(x) -> ∃y.(Set(y) & ∀z.((z ⊂ x) -> (z ϵ y)))
2. (Set(x) & Set(y)) -> Set((x ∪ y))
3. (Function(f) & Set(domain(f))) -> Set(range(f))
4. Set(x) -> Set(∪x)
5. ¬(x = 0) -> ∃y.((y ϵ x) & ((y ∩ x) = 0))
6. ∃y.((Set(y) & (0 ϵ y)) & ∀x.((x ϵ y) -> (suc x ϵ y)))
7. ∃f.(Choice(f) & (domain(f) = (U ~ {0})))
>>> ShowDefEquations()
0. (x ∪ y) = {z: ((z ϵ x) v (z ϵ y))}
1. (x ∩ y) = {z: ((z ϵ x) & (z ϵ y))}
2. ~x = {y: ¬(y ϵ x)}
3. (x ~ y) = (x ∩ ~y)
4. 0 = {x: ¬(x = x)}
5. U = {x: (x = x)}
6. ∪x = {z: ∃y.((y ϵ x) & (z ϵ y))}
7. ∩x = {z: ∀y.((y ϵ x) -> (z ϵ y))}
8. Px = {y: (y ⊂ x)}
9. {x} = {z: ((z ϵ U) -> (z = x))}
10. {x,y} = ({x} ∪ {y})
11. (x,y) = {x,{x,y}}
```

[3]https://github.com/owl77/PyLog

```
12. proj1(x) = ∩∩x
13. proj2(x) = (∩∪x ∪ (∪∪x ~ ∪∩x))
14. (a∘b) = {w: ∃x.∃y.∃z.((((x,y) ∈ a) & ((y,z) ∈ b)) & (w = (x,z)))}
15. (r)^-11 = {z: ∃x.∃y.(((x,y) ∈ r) & (z = (y,x)))}
16. domain(f) = {x: ∃y.((x,y) ∈ f)}
17. range(f) = {y: ∃x.((x,y) ∈ f)}
18. (f'x) = ∩{y: ((x,y) ∈ f)}
19. (x X y) = {z: ∃a.∃b.((z = (a,b)) & ((a ∈ x) & (b ∈ y)))}
20. func(x,y) = {f: (Function(f) & ((domain(f) = x) & (range(f) = y)))}
21. E = {z: ∃x.∃y.((z = (x,y)) & (x ∈ y))}
22. ord = {x: Ordinal(x)}
23. suc x = (x ∪ {x})
24. (f|x) = (f ∩ (x X U))
25. ω = {x: Integer(x)}
>>> ShowDefinitions()
Set(x) <-> ∃y.(x ∈ y)
(x ⊂ y) <-> ∀z.((z ∈ x) -> (z ∈ y))
Relation(r) <-> ∀z.((z ∈ r) -> ∃x.∃y.(z = (x,y)))
Function(f) <-> (Relation(f) & ∀x.∀y.∀z.((((x,y) ∈ f) & ((x,z) ∈ f)) -> (y = z)))
Trans(r) <-> ∀x.∀y.∀z.((((x,y) ∈ r) & ((y,z) ∈ r)) -> ((x,z) ∈ r))
Connects(r,x) <-> ∀y.∀z.(((y ∈ x) & (z ∈ x)) -> ((y = z) v (((y,z) ∈ r) v ((z,y) ∈ r))))
Asymmetric(r,x) <-> ∀y.∀z.(((y ∈ x) & (z ∈ x)) -> (((y,z) ∈ r) -> ¬((z,y) ∈ r)))
First(r,x,z) <-> ((z ∈ x) & ∀y.((y ∈ x) -> ¬((y,z) ∈ r)))
WellOrders(r,x) <-> (Connects(r,x) & ∀y.(((y ⊂ x) & ¬(y = 0)) -> ∃z.First(r,y,z)))
Section(r,x,y) <-> (((y ⊂ x) & WellOrders(r,x)) & ∀u.∀v.((((u ∈ x)
 & (v ∈ y)) & ((u,v) ∈ r)) -> (u ∈ y)))
OrderPreserving(f,r,s) <-> ((Function(f) & (WellOrders(r,domain(f))
 & WellOrders(r,range(f)))) & ∀u.∀v.((((u ∈ domain(f))
 & (v ∈ domain(f))) & ((u,v) ∈ r)) -> (((f'u),(f'v)) ∈ r)))
1-to-1(f) <-> (Function(f) & Function((f)^-11))
Full(x) <-> ∀y.((y ∈ x) -> (y ⊂ x))
Ordinal(x) <-> (Full(x) & Connects(E,x))
Integer(x) <-> (Ordinal(x) & WellOrders((E)^-11,x))
Choice(f) <-> (Function(f) & ∀y.((y ∈ domain(f)) -> ((f'y) ∈ y)))
Equi(x,y) <-> ∃f.(1-to-1(f) & ((domain(f) = x) & (range(f) = y)))
Card(x) <-> (Ordinal(x) & ∀y.(((y ∈ x) & (y ∈ ord)) -> ¬Equi(y,x)))
TransIn(r,x) <-> ∀u.∀v.∀w.(((u ∈ x) & ((v ∈ x) & (w ∈ x))) ->
 ((((u,v) ∈ r) & ((v,w) ∈ r)) -> ((u,w) ∈ r)))
```

我们还可以通过 ShowProof() 检查证明是否为空。默认情况下，表达式会以美化打印的方式显示（使用 Unicode 字符），并且可以采用中缀表示法。可以通过 parser.prettyprint[FunctionNameString] = PrettyString 更改美化打印方式。输入表达式时必须严格采用函数式方式（逻辑连接词、扩展和量词除外）。

```
Input and default "pretty" display
==================================
neg A                          ¬A
bigunion(x)                    ∪x
bigintersection                ∩x
union(x,y)                    (x ∪ y)
```

```
intersection(x,y)        (x ∩ y)
extension x. A           {x: A}
forall x. A              ∀x. A
exists x. A              ∃x. A
Elem(x,y)               (x ϵ y)
app(f,x)                 (f'x)
pair(x,y)                {x,y}
singleton(x)             {x}
orderedpair(x,y)         (x,y)
prod(x,y)                (x X y)
complement1(x)             ~x
complement2(x)           (x ~ y)
parts(x)                 Px
comp(a,b)                (a∘b)
inv(r)                    (r)^-11
restrict(f,x)             (f|x)
int                        ω
```

请注意，$\neg A$ 是 $A \to \bot$的美观打印。使用 PyLog 的规则时，我们必须以这种方式理解 $\neg A$ 。合取 & 通常以中缀形式输入（ A & B ），但 PyLog 会自动在右侧添加并分组括号：因此，(A & B & C) 会被解释为 (A & (B & C))。

PyLog 中的第一个证明

本节将在 Kelley-Morse 证明环境中证明 [4]的定理 4:

```
((z ϵ (x ∪ y)) <-> ((z ϵ x) v (z ϵ y))) & ((z ϵ (x ∩ y)) <-> ((z ϵ x) & (z ϵ y)))
```

我 们在此详细介绍一个证明该定理前半部分的会话 .

```
Welcome to PyLog 1.0

Natural Deduction Proof Assistant and Proof Checker

(c) 2020  C. Lewis Protin

>>> Load("Kelley-Morse")
True
>>> ShowProof()
>>> Hyp("Elem(z,union(x,y))")
0. z ϵ (x ∪ y) Hyp
True
>>> ShowDefEquations()
0. (x ∪ y) = {z: ((z ϵ x) v (z ϵ y))}
1. (x ∩ y) = {z: ((z ϵ x) & (z ϵ y))}
2. ~x = {y: ¬(y ϵ x)}
3. (x ~ y) = (x ∩ ~y)
4. 0 = {x: ¬(x = x)}
5. U = {x: (x = x)}
6. ∪x = {z: ∃y.((y ϵ x) & (z ϵ y))}
```

```
7. ∩x = {z: ∀y.((y ∈ x) -> (z ∈ y))}
8. Px = {y: (y ⊂ x)}
9. {x} = {z: ((z ∈ U) -> (z = x))}
10. {x,y} = ({x} ∪ {y})
11. (x,y) = {x,{x,y}}
12. proj1(x) = ∩∩x
13. proj2(x) = (∩∪x ∪ (∪∪x ~ ∪∩x))
14. (a∘b) = {w: ∃x.∃y.∃z.((((x,y) ∈ a) & ((y,z) ∈ b)) & (w = (x,z)))}
15. (r)^-11 = {z: ∃x.∃y.(((x,y) ∈ r) & (z = (y,x)))}
16. domain(f) = {x: ∃y.((x,y) ∈ f)}
17. range(f) = {y: ∃x.((x,y) ∈ f)}
18. (f'x) = ∩{y: ((x,y) ∈ f)}
19. (x X y) = {z: ∃a.∃b.((z = (a,b)) & ((a ∈ x) & (b ∈ y)))}
20. func(x,y) = {f: (Function(f) & ((domain(f) = x) & (range(f) = y)))}
21. E = {z: ∃x.∃y.((z = (x,y)) & (x ∈ y))}
22. ord = {x: Ordinal(x)}
23. suc x = (x ∪ {x})
24. (f|x) = (f ∩ (x X U))
25. ω = {x: Integer(x)}
>>> DefEqInt(0)
0. z ∈ (x ∪ y) Hyp
1. (x ∪ y) = {z: ((z ∈ x) v (z ∈ y))} DefEqInt
True
>>> EqualitySub(0,1,[0])
0. z ∈ (x ∪ y) Hyp
1. (x ∪ y) = {z: ((z ∈ x) v (z ∈ y))} DefEqInt
2. z ∈ {z: ((z ∈ x) v (z ∈ y))} EqualitySub 0 1
True
>>> ClassElim(2)
0. z ∈ (x ∪ y) Hyp
1. (x ∪ y) = {z: ((z ∈ x) v (z ∈ y))} DefEqInt
2. z ∈ {z: ((z ∈ x) v (z ∈ y))} EqualitySub 0 1
3. Set(z) & ((z ∈ x) v (z ∈ y)) ClassElim 2
True
>>> AndElimR(3)
0. z ∈ (x ∪ y) Hyp
1. (x ∪ y) = {z: ((z ∈ x) v (z ∈ y))} DefEqInt
2. z ∈ {z: ((z ∈ x) v (z ∈ y))} EqualitySub 0 1
3. Set(z) & ((z ∈ x) v (z ∈ y)) ClassElim 2
4. (z ∈ x) v (z ∈ y) AndElimR 3
True
>>> ImpInt(4,0)
0. z ∈ (x ∪ y) Hyp
1. (x ∪ y) = {z: ((z ∈ x) v (z ∈ y))} DefEqInt
2. z ∈ {z: ((z ∈ x) v (z ∈ y))} EqualitySub 0 1
3. Set(z) & ((z ∈ x) v (z ∈ y)) ClassElim 2
4. (z ∈ x) v (z ∈ y) AndElimR 3
5. (z ∈ (x ∪ y)) -> ((z ∈ x) v (z ∈ y)) ImpInt 4
True
>>> Qed(5)
```

```
(...)
5. (z ∈ (x ∪ y)) -> ((z ∈ x) v (z ∈ y)) ImpInt 4 Qed
>>> Hyp("(Elem(z,x) v Elem(z,y))")
0. z ∈ (x ∪ y) Hyp
1. (x ∪ y) = {z: ((z ∈ x) v (z ∈ y))} DefEqInt
2. z ∈ {z: ((z ∈ x) v (z ∈ y))} EqualitySub 0 1
3. Set(z) & ((z ∈ x) v (z ∈ y)) ClassElim 2
4. (z ∈ x) v (z ∈ y) AndElimR 3
5. (z ∈ (x ∪ y)) -> ((z ∈ x) v (z ∈ y)) ImpInt 4 Qed
6. (z ∈ x) v (z ∈ y) Hyp
True
>>> Hyp("Elem(z,x)")
0. z ∈ (x ∪ y) Hyp
1. (x ∪ y) = {z: ((z ∈ x) v (z ∈ y))} DefEqInt
2. z ∈ {z: ((z ∈ x) v (z ∈ y))} EqualitySub 0 1
3. Set(z) & ((z ∈ x) v (z ∈ y)) ClassElim 2
4. (z ∈ x) v (z ∈ y) AndElimR 3
5. (z ∈ (x ∪ y)) -> ((z ∈ x) v (z ∈ y)) ImpInt 4 Qed
6. (z ∈ x) v (z ∈ y) Hyp
7. z ∈ x Hyp
True
>>> ExistsInt(7,"x","x",[0])
0. z ∈ (x ∪ y) Hyp
1. (x ∪ y) = {z: ((z ∈ x) v (z ∈ y))} DefEqInt
2. z ∈ {z: ((z ∈ x) v (z ∈ y))} EqualitySub 0 1
3. Set(z) & ((z ∈ x) v (z ∈ y)) ClassElim 2
4. (z ∈ x) v (z ∈ y) AndElimR 3
5. (z ∈ (x ∪ y)) -> ((z ∈ x) v (z ∈ y)) ImpInt 4 Qed
6. (z ∈ x) v (z ∈ y) Hyp
7. z ∈ x Hyp
8. ∃x.(z ∈ x) ExistsInt 7
True
>>> ShowDefinitions()
Set(x) <-> ∃y.(x ∈ y)
(x ⊂ y) <-> ∀z.((z ∈ x) -> (z ∈ y))
Relation(r) <-> ∀z.((z ∈ r) -> ∃x.∃y.(z = (x,y)))
Function(f) <-> (Relation(f) & ∀x.∀y.∀z.((((x,y) ∈ f) & ((x,z) ∈ f)) -> (y = z)))
Trans(r) <-> ∀x.∀y.∀z.((((x,y) ∈ r) & ((y,z) ∈ r)) -> ((x,z) ∈ r))
Connects(r,x) <-> ∀y.∀z.(((y ∈ x) & (z ∈ x)) -> ((y = z) v (((y z) ∈ r) v ((z,y) ∈ r))))
Asymmetric(r,x) <-> ∀y.∀z.(((y ∈ x) & (z ∈ x)) -> (((y,z) ∈ r) -> ¬((z,y) ∈ r)))
First(r,x,z) <-> ((z ∈ x) & ∀y.((y ∈ x) -> ¬((y,z) ∈ r)))
WellOrders(r,x) <-> (Connects(r,x) & ∀y.(((y ⊂ x) & ¬(y = 0)) -> ∃z.First(r,y,z)))
Section(r,x,y) <-> (((y ⊂ x) & WellOrders(r,x)) & ∀u.∀v.((((u ∈ x)
& (v ∈ y)) & ((u,v) ∈ r)) -> (u ∈ y)))
OrderPreserving(f,r,s) <-> ((Function(f) & (WellOrders(r,domain(f))
 & WellOrders(r,range(f)))) & ∀u.∀v.((((u ∈ domain(f)) & (v ∈ domain(f))) & ((u,v) ∈ r))
1-to-1(f) <-> (Function(f) & Function((f)^-11))
Full(x) <-> ∀y.((y ∈ x) -> (y ⊂ x))
Ordinal(x) <-> (Full(x) & Connects(E,x))
Integer(x) <-> (Ordinal(x) & WellOrders((E)^-11,x))
```

```
Choice(f) <-> (Function(f) & ∀y.((y ∈ domain(f)) -> ((f'y) ∈ y)))
Equi(x,y) <-> ∃f.(1-to-1(f) & ((domain(f) = x) & (range(f) = y)))
Card(x) <-> (Ordinal(x) & ∀y.(((y ∈ x) & (y ∈ ord)) -> ¬Equi(y,x)))
TransIn(r,x) <-> ∀u.∀v.∀w.(((u ∈ x) & ((v ∈ x) & (w ∈ x))) -> ((((u,v) ∈ r)
& ((v,w) ∈ r)) -> ((u,w) ∈ r)))
>>> DefSub(8,"Set","z",[0])
0. z ∈ (x ∪ y) Hyp
1. (x ∪ y) = {z: ((z ∈ x) v (z ∈ y))} DefEqInt
2. z ∈ {z: ((z ∈ x) v (z ∈ y))} EqualitySub 0 1
3. Set(z) & ((z ∈ x) v (z ∈ y)) ClassElim 2
4. (z ∈ x) v (z ∈ y) AndElimR 3
5. (z ∈ (x ∪ y)) -> ((z ∈ x) v (z ∈ y)) ImpInt 4 Qed
6. (z ∈ x) v (z ∈ y) Hyp
7. z ∈ x Hyp
8. ∃x.(z ∈ x) ExistsInt 7
9. Set(z) DefSub 8
True
>>> Hyp("Elem(z,y)")
0. z ∈ (x ∪ y) Hyp
1. (x ∪ y) = {z: ((z ∈ x) v (z ∈ y))} DefEqInt
2. z ∈ {z: ((z ∈ x) v (z ∈ y))} EqualitySub 0 1
3. Set(z) & ((z ∈ x) v (z ∈ y)) ClassElim 2
4. (z ∈ x) v (z ∈ y) AndElimR 3
5. (z ∈ (x ∪ y)) -> ((z ∈ x) v (z ∈ y)) ImpInt 4 Qed
6. (z ∈ x) v (z ∈ y) Hyp
7. z ∈ x Hyp
8. ∃x.(z ∈ x) ExistsInt 7
9. Set(z) DefSub 8
10. z ∈ y Hyp
True
>>> ExistsInt(10, "y","x",[0])
0. z ∈ (x ∪ y) Hyp
1. (x ∪ y) = {z: ((z ∈ x) v (z ∈ y))} DefEqInt
2. z ∈ {z: ((z ∈ x) v (z ∈ y))} EqualitySub 0 1
3. Set(z) & ((z ∈ x) v (z ∈ y)) ClassElim 2
4. (z ∈ x) v (z ∈ y) AndElimR 3
5. (z ∈ (x ∪ y)) -> ((z ∈ x) v (z ∈ y)) ImpInt 4 Qed
6. (z ∈ x) v (z ∈ y) Hyp
7. z ∈ x Hyp
8. ∃x.(z ∈ x) ExistsInt 7
9. Set(z) DefSub 8
10. z ∈ y Hyp
11. ∃x.(z ∈ x) ExistsInt 10
True
>>> DefSub(11,"Set","z",[0])
0. z ∈ (x ∪ y) Hyp
1. (x ∪ y) = {z: ((z ∈ x) v (z ∈ y))} DefEqInt
2. z ∈ {z: ((z ∈ x) v (z ∈ y))} EqualitySub 0 1
3. Set(z) & ((z ∈ x) v (z ∈ y)) ClassElim 2
4. (z ∈ x) v (z ∈ y) AndElimR 3
```

```
5. (z ∈ (x ∪ y)) -> ((z ∈ x) v (z ∈ y)) ImpInt 4 Qed
6. (z ∈ x) v (z ∈ y) Hyp
7. z ∈ x Hyp
8. ∃x.(z ∈ x) ExistsInt 7
9. Set(z) DefSub 8
10. z ∈ y Hyp
11. ∃x.(z ∈ x) ExistsInt 10
12. Set(z) DefSub 11
True
>>> OrElim(6,7,9,10,12)
0. z ∈ (x ∪ y) Hyp
1. (x ∪ y) = {z: ((z ∈ x) v (z ∈ y))} DefEqInt
2. z ∈ {z: ((z ∈ x) v (z ∈ y))} EqualitySub 0 1
3. Set(z) & ((z ∈ x) v (z ∈ y)) ClassElim 2
4. (z ∈ x) v (z ∈ y) AndElimR 3
5. (z ∈ (x ∪ y)) -> ((z ∈ x) v (z ∈ y)) ImpInt 4 Qed
6. (z ∈ x) v (z ∈ y) Hyp
7. z ∈ x Hyp
8. ∃x.(z ∈ x) ExistsInt 7
9. Set(z) DefSub 8
10. z ∈ y Hyp
11. ∃x.(z ∈ x) ExistsInt 10
12. Set(z) DefSub 11
13. Set(z) OrElim 6 7 9 10 12
True
>>> AndInt(13,6)
0. z ∈ (x ∪ y) Hyp
1. (x ∪ y) = {z: ((z ∈ x) v (z ∈ y))} DefEqInt
2. z ∈ {z: ((z ∈ x) v (z ∈ y))} EqualitySub 0 1
3. Set(z) & ((z ∈ x) v (z ∈ y)) ClassElim 2
4. (z ∈ x) v (z ∈ y) AndElimR 3
5. (z ∈ (x ∪ y)) -> ((z ∈ x) v (z ∈ y)) ImpInt 4 Qed
6. (z ∈ x) v (z ∈ y) Hyp
7. z ∈ x Hyp
8. ∃x.(z ∈ x) ExistsInt 7
9. Set(z) DefSub 8
10. z ∈ y Hyp
11. ∃x.(z ∈ x) ExistsInt 10
12. Set(z) DefSub 11
13. Set(z) OrElim 6 7 9 10 12
14. Set(z) & ((z ∈ x) v (z ∈ y)) AndInt 13 6
True
>>> ClassInt(14,"z")
0. z ∈ (x ∪ y) Hyp
1. (x ∪ y) = {z: ((z ∈ x) v (z ∈ y))} DefEqInt
2. z ∈ {z: ((z ∈ x) v (z ∈ y))} EqualitySub 0 1
3. Set(z) & ((z ∈ x) v (z ∈ y)) ClassElim 2
4. (z ∈ x) v (z ∈ y) AndElimR 3
5. (z ∈ (x ∪ y)) -> ((z ∈ x) v (z ∈ y)) ImpInt 4 Qed
6. (z ∈ x) v (z ∈ y) Hyp
```

```
7. z ∈ x Hyp
8. ∃x.(z ∈ x) ExistsInt 7
9. Set(z) DefSub 8
10. z ∈ y Hyp
11. ∃x.(z ∈ x) ExistsInt 10
12. Set(z) DefSub 11
13. Set(z) OrElim 6 7 9 10 12
14. Set(z) & ((z ∈ x) v (z ∈ y)) AndInt 13 6
15. z ∈ {z: ((z ∈ x) v (z ∈ y))} ClassInt 14
True
>>> Symmetry(1)
0. z ∈ (x ∪ y) Hyp
1. (x ∪ y) = {z: ((z ∈ x) v (z ∈ y))} DefEqInt
2. z ∈ {z: ((z ∈ x) v (z ∈ y))} EqualitySub 0 1
3. Set(z) & ((z ∈ x) v (z ∈ y)) ClassElim 2
4. (z ∈ x) v (z ∈ y) AndElimR 3
5. (z ∈ (x ∪ y)) -> ((z ∈ x) v (z ∈ y)) ImpInt 4 Qed
6. (z ∈ x) v (z ∈ y) Hyp
7. z ∈ x Hyp
8. ∃x.(z ∈ x) ExistsInt 7
9. Set(z) DefSub 8
10. z ∈ y Hyp
11. ∃x.(z ∈ x) ExistsInt 10
12. Set(z) DefSub 11
13. Set(z) OrElim 6 7 9 10 12
14. Set(z) & ((z ∈ x) v (z ∈ y)) AndInt 13 6
15. z ∈ {z: ((z ∈ x) v (z ∈ y))} ClassInt 14
16. {z: ((z ∈ x) v (z ∈ y))} = (x ∪ y) Symmetry 1
True
>>> EqualitySub(15,16,[0])
0. z ∈ (x ∪ y) Hyp
1. (x ∪ y) = {z: ((z ∈ x) v (z ∈ y))} DefEqInt
2. z ∈ {z: ((z ∈ x) v (z ∈ y))} EqualitySub 0 1
3. Set(z) & ((z ∈ x) v (z ∈ y)) ClassElim 2
4. (z ∈ x) v (z ∈ y) AndElimR 3
5. (z ∈ (x ∪ y)) -> ((z ∈ x) v (z ∈ y)) ImpInt 4 Qed
6. (z ∈ x) v (z ∈ y) Hyp
7. z ∈ x Hyp
8. ∃x.(z ∈ x) ExistsInt 7
9. Set(z) DefSub 8
10. z ∈ y Hyp
11. ∃x.(z ∈ x) ExistsInt 10
12. Set(z) DefSub 11
13. Set(z) OrElim 6 7 9 10 12
14. Set(z) & ((z ∈ x) v (z ∈ y)) AndInt 13 6
15. z ∈ {z: ((z ∈ x) v (z ∈ y))} ClassInt 14
16. {z: ((z ∈ x) v (z ∈ y))} = (x ∪ y) Symmetry 1
17. z ∈ (x ∪ y) EqualitySub 15 16
True
>>> ImpInt(17,6)
```

```
0. z ∈ (x ∪ y) Hyp
1. (x ∪ y) = {z: ((z ∈ x) v (z ∈ y))} DefEqInt
2. z ∈ {z: ((z ∈ x) v (z ∈ y))} EqualitySub 0 1
3. Set(z) & ((z ∈ x) v (z ∈ y)) ClassElim 2
4. (z ∈ x) v (z ∈ y) AndElimR 3
5. (z ∈ (x ∪ y)) -> ((z ∈ x) v (z ∈ y)) ImpInt 4 Qed
6. (z ∈ x) v (z ∈ y) Hyp
7. z ∈ x Hyp
8. ∃x.(z ∈ x) ExistsInt 7
9. Set(z) DefSub 8
10. z ∈ y Hyp
11. ∃x.(z ∈ x) ExistsInt 10
12. Set(z) DefSub 11
13. Set(z) OrElim 6 7 9 10 12
14. Set(z) & ((z ∈ x) v (z ∈ y)) AndInt 13 6
15. z ∈ {z: ((z ∈ x) v (z ∈ y))} ClassInt 14
16. {z: ((z ∈ x) v (z ∈ y))} = (x ∪ y) Symmetry 1
17. z ∈ (x ∪ y) EqualitySub 15 16
18. ((z ∈ x) v (z ∈ y)) -> (z ∈ (x ∪ y)) ImpInt 17
True
>>> Qed(18)
True
>>> ShowProof()
(...)
18. ((z ∈ x) v (z ∈ y)) -> (z ∈ (x ∪ y)) ImpInt 17 Qed
>>> AndInt(5,18)
0. z ∈ (x ∪ y) Hyp
1. (x ∪ y) = {z: ((z ∈ x) v (z ∈ y))} DefEqInt
2. z ∈ {z: ((z ∈ x) v (z ∈ y))} EqualitySub 0 1
3. Set(z) & ((z ∈ x) v (z ∈ y)) ClassElim 2
4. (z ∈ x) v (z ∈ y) AndElimR 3
5. (z ∈ (x ∪ y)) -> ((z ∈ x) v (z ∈ y)) ImpInt 4 Qed
6. (z ∈ x) v (z ∈ y) Hyp
7. z ∈ x Hyp
8. ∃x.(z ∈ x) ExistsInt 7
9. Set(z) DefSub 8
10. z ∈ y Hyp
11. ∃x.(z ∈ x) ExistsInt 10
12. Set(z) DefSub 11
13. Set(z) OrElim 6 7 9 10 12
14. Set(z) & ((z ∈ x) v (z ∈ y)) AndInt 13 6
15. z ∈ {z: ((z ∈ x) v (z ∈ y))} ClassInt 14
16. {z: ((z ∈ x) v (z ∈ y))} = (x ∪ y) Symmetry 1
17. z ∈ (x ∪ y) EqualitySub 15 16
18. ((z ∈ x) v (z ∈ y)) -> (z ∈ (x ∪ y)) ImpInt 17 Qed
19. ((z ∈ (x ∪ y)) -> ((z ∈ x) v (z ∈ y))) &
(((z ∈ x) v (z ∈ y)) -> (z ∈ (x ∪ y))) AndInt 5 18
True
>>> EquivConst(19)
0. z ∈ (x ∪ y) Hyp
```

```
1. (x ∪ y) = {z: ((z ∈ x) v (z ∈ y))} DefEqInt
2. z ∈ {z: ((z ∈ x) v (z ∈ y))} EqualitySub 0 1
3. Set(z) & ((z ∈ x) v (z ∈ y)) ClassElim 2
4. (z ∈ x) v (z ∈ y) AndElimR 3
5. (z ∈ (x ∪ y)) -> ((z ∈ x) v (z ∈ y)) ImpInt 4 Qed
6. (z ∈ x) v (z ∈ y) Hyp
7. z ∈ x Hyp
8. ∃x.(z ∈ x) ExistsInt 7
9. Set(z) DefSub 8
10. z ∈ y Hyp
11. ∃x.(z ∈ x) ExistsInt 10
12. Set(z) DefSub 11
13. Set(z) OrElim 6 7 9 10 12
14. Set(z) & ((z ∈ x) v (z ∈ y)) AndInt 13 6
15. z ∈ {z: ((z ∈ x) v (z ∈ y))} ClassInt 14
16. {z: ((z ∈ x) v (z ∈ y))} = (x ∪ y) Symmetry 1
17. z ∈ (x ∪ y) EqualitySub 15 16
18. ((z ∈ x) v (z ∈ y)) -> (z ∈ (x ∪ y)) ImpInt 17 Qed
19. ((z ∈ (x ∪ y)) -> ((z ∈ x) v (z ∈ y))) &
 (((z ∈ x) v (z ∈ y)) -> (z ∈ (x ∪ y))) AndInt 5 18
20. (z ∈ (x ∪ y)) <-> ((z ∈ x) v (z ∈ y)) EquivConst
True
>>> Qed(20)
(...)
20. (z ∈ (x ∪ y)) <-> ((z ∈ x) v (z ∈ y)) EquivConst  Qed
>>> Save("Th4")
True
```

证明已由日志（以及证明环境）完整编码:

```
>>> ShowLog()
0. Hyp("Elem(z, union(x,y))")
1. DefEqInt(0)
2. EqualitySub(0,1,[0])
3. ClassElim(2)
4. AndElimR(3)
5. ImpInt(4,0)
6. Hyp("(Elem(z,x) v Elem(z,y))")
7. Hyp("Elem(z,x)")
8. ExistsInt(7,"x","x",[0])
9. DefSub(8,"Set",["z"],[0])
10. Hyp("Elem(z,y)")
11. ExistsInt(10,"y","y",[0])
12. DefSub(11,"Set",["z"],[0])
13. OrElim(6,7,9,10,12)
14. AndInt(13,6)
15. ClassInt(14,"z")
16. Symmetry(1)
17. EqualitySub(15,16,[0])
18. ImpInt(17,6)
19. AndInt(5,18)
```

```
20. EquivConst(19)
```

命令 GenerateProof() 会删除证明，然后根据 Proof.log 字段重新生成证明，并对最后一行运行 Qed()。这样就可以对给定环境中的任何证明日志进行形式化检查。该日志包含生成证明所需的全部信息。命令 UsedTheorems() 会列出证明中使用的其他定理。

类似地，合取的后半部分得证：

```
21. z ∈ (x ∩ y) Hyp
22. (x ∩ y) = {z: ((z ∈ x) & (z ∈ y))} DefEqInt
23. z ∈ {z: ((z ∈ x) & (z ∈ y))} EqualitySub 21 22
24. Set(z) & ((z ∈ x) & (z ∈ y)) ClassElim 23
25. (z ∈ x) & (z ∈ y) AndElimR 24
26. (z ∈ (x ∩ y)) -> ((z ∈ x) & (z ∈ y)) ImpInt 25 Qed
27. (z ∈ x) & (z ∈ y) Hyp
28. z ∈ x AndElimL 27
29. ∃x.(z ∈ x) ExistsInt 28
30. Set(z) DefSub 29
31. Set(z) & ((z ∈ x) & (z ∈ y)) AndInt 30 27
32. z ∈ {z: ((z ∈ x) & (z ∈ y))} ClassInt 31
33. {z: ((z ∈ x) & (z ∈ y))} = (x ∩ y) Symmetry 22
34. z ∈ (x ∩ y) EqualitySub 32 33
35. ((z ∈ x) & (z ∈ y)) -> (z ∈ (x ∩ y)) ImpInt 34 Qed
36. ((z ∈ (x ∩ y)) -> ((z ∈ x) & (z ∈ y))) & (((z ∈ x) & (z ∈ y)) -> (z ∈ (x ∩ y)))
 AndInt 26 35
37. (z ∈ (x ∩ y)) <-> ((z ∈ x) & (z ∈ y)) EquivConst  Qed
38. ((z ∈ (x ∪ y)) <-> ((z ∈ x) v (z ∈ y))) & ((z ∈ (x ∩ y)) <-> ((z ∈ x) & (z ∈ y)))
 AndInt 20 37
```

使用 log

```
21. Hyp("Elem(z, intersection(x,y))")
22. DefEqInt(1)
23. EqualitySub(21,22,[0])
24. ClassElim(23)
25. AndElimR(24)
26. ImpInt(25,21)
27. Hyp("(Elem(z,x) & Elem(z,y))")
28. AndElimL(27)
29. ExistsInt(28,"x","x",[0])
30. DefSub(29,"Set",["z"],[0])
31. AndInt(30,27)
32. ClassInt(31,"z")
33. Symmetry(22)
34. EqualitySub(32,33,[0])
35. ImpInt(34,27)
36. AndInt(26,35)
37. EquivConst(36)
38. AndInt(20,37)
```

该定理位于 PyLog 仓库的主目录中，可以通过 Load("Th4") 加载。我们还展示了 [4]定理 5 的合取的前半部分的证明，以说明如何使用先前的定理并应用公理：

```
0. z ∈ (x ∪ x) Hyp
1. ((z ∈ (x ∪ y)) <-> ((z ∈ x) v (z ∈ y))) & ((z ∈ (x ∩ y)) <-> ((z ∈ x) & (z ∈ y))) Theo
2. (z ∈ (x ∪ y)) <-> ((z ∈ x) v (z ∈ y)) AndElimL 1
3. ((z ∈ (x ∪ y)) -> ((z ∈ x) v (z ∈ y))) & (((z ∈ x) v (z ∈ y)) -> (z ∈ (x ∪ y))) Equivl
4. (z ∈ (x ∪ y)) -> ((z ∈ x) v (z ∈ y)) AndElimL 3
5. ∀y.((z ∈ (x ∪ y)) -> ((z ∈ x) v (z ∈ y))) ForallInt 4
6. (z ∈ (x ∪ x)) -> ((z ∈ x) v (z ∈ x)) ForallElim 5
7. (z ∈ x) v (z ∈ x) ImpElim 0 6
8. z ∈ x Hyp
9. z ∈ x Hyp
10. z ∈ x OrElim 7 8 8 9 9
11. (z ∈ (x ∪ x)) -> (z ∈ x) ImpInt 10 Qed
12. z ∈ x Hyp
13. (z ∈ x) v (z ∈ x) OrIntL 12
14. ((z ∈ x) v (z ∈ y)) -> (z ∈ (x ∪ y)) AndElimR 3
15. ∀y.(((z ∈ x) v (z ∈ y)) -> (z ∈ (x ∪ y))) ForallInt 14
16. ((z ∈ x) v (z ∈ x)) -> (z ∈ (x ∪ x)) ForallElim 15
17. z ∈ (x ∪ x) ImpElim 13 16
18. (z ∈ x) -> (z ∈ (x ∪ x)) ImpInt 17 Qed
19. ((z ∈ (x ∪ x)) -> (z ∈ x)) & ((z ∈ x) -> (z ∈ (x ∪ x))) AndInt 11 18
20. (z ∈ (x ∪ x)) <-> (z ∈ x) EquivConst
21. ∀z.((z ∈ (x ∪ x)) <-> (z ∈ x)) ForallInt 20 Qed
22. ∀x.∀y.((x = y) <-> ∀z.((z ∈ x) <-> (z ∈ y))) AxInt
23. ∀y.(((x ∪ x) = y) <-> ∀z.((z ∈ (x ∪ x)) <-> (z ∈ y))) ForallElim 22
24. ((x ∪ x) = x) <-> ∀z.((z ∈ (x ∪ x)) <-> (z ∈ x)) ForallElim 23
25. (((x ∪ x) = x) -> ∀z.((z ∈ (x ∪ x)) <-> (z ∈ x))) & (∀z.((z ∈ (x ∪ x)) <-> (z ∈ x))
26. ∀z.((z ∈ (x ∪ x)) <-> (z ∈ x)) -> ((x ∪ x) = x) AndElimR 25
27. (x ∪ x) = x ImpElim 21 26 Qed
```

日志是

```
0. Hyp("Elem(z,union(x,x))")
1. TheoremInt(1)
2. AndElimL(1)
3. EquivExp(2)
4. AndElimL(3)
5. ForallInt(4,"y","y")
6. ForallElim(5,"x")
7. ImpElim(0,6)
8. Hyp("Elem(z,x)")
9. Hyp("Elem(z,x)")
10. OrElim(7,8,8,9,9)
11. ImpInt(10,0)
12. Hyp("Elem(z,x)")
13. OrIntL(12,"Elem(z,x)")
14. AndElimR(3)
15. ForallInt(14,"y","y")
16. ForallElim(15,"x")
17. ImpElim(13,16)
18. ImpInt(17,12)
19. AndInt(11,18)
```

```
20. EquivConst(19)
21. ForallInt(20,"z","z")
22. AxInt(0)
23. ForallElim(22,"union(x,x)")
24. ForallElim(23,"x")
25. EquivExp(24)
26. AndElimR(25)
27. ImpElim(21,26)
```

要证明 Th6 和 Th7，我们可以利用 PolySub 和先前证明的命题有效式。

例如，如果某个逻辑有效式已被证明

```
0. A v B Hyp
1. A Hyp
2. B v A OrIntL 1
3. B Hyp
4. B v A OrIntR 3
5. B v A OrElim 0 1 2 3 4
6. (A v B) -> (B v A) ImpInt 5 Qed
```

```
0. Hyp("(A v B)")
1. Hyp("A")
2. OrIntL(1,"B")
3. Hyp("B")
4. OrIntR(3,"A")
5. OrElim(0,1,2,3,4)
6. ImpInt(5,0)
```

假设我们将这个定理保存为"Log1"。然后，我们可以将该定理添加到我们的环境中，并通过 PolySub 将其具体化，例如：

```
>>> Load("Kelley-Morse")
True
>>>
>>>
>>> ViewTheorem("Log1")
'Log1 : (A v B) -> (B v A)'
>>> LoadTheorem("Log1")
True
>>> ShowTheorems()
0. A v ¬A
1. (A v B) -> (B v A)
>>> TheoremInt(1)
True
>>> ShowProof()
0. (A v B) -> (B v A) TheoremInt
>>> PolySub(0,"A","Elem(z,x)")
(...)
>>> PolySub(1,"B","Elem(z,y)")
0. (A v B) -> (B v A) TheoremInt
```

```
1. ((z ϵ x) v B) -> (B v (z ϵ x)) PolySub 0
2. ((z ϵ x) v (z ϵ y)) -> ((z ϵ y) v (z ϵ x)) PolySub 1
>>> ShowLog()
0. TheoremInt(1)
1. PolySub(0,"A","Elem(z,x)")
2. PolySub(1,"B","Elem(z,y)")
```

添加定理时，务必确保这些定理使用相同的环境保存。逻辑有效式是个例外。

如果某个谓词属于语言，但尚未定义，那么可以按照经典意义将其视为一个二阶变量（即具有固定的元数），并使用 PredSub 命令。这有助于实现归纳等公理模式，也便于复用一阶有效式（多态变量无法做到这一点）。

```
>>> Load("Kelley-Morse")
True
>>> AddPredicate("Foo",1,True)
True
>>> Hyp("Foo(x)")
0. Foo(x)  Hyp
True
>>> PredSub(0,"Foo",["x"], "neg Set(x)", [0])
0. Foo(x)  Hyp
1. ¬Set(x)  PredSub 0
True
>>> PredSub(1,"Set",["x"], "neg Set(x)", [0])
Predicate is defined.
```

要检验一个理论（即一组定理），可以运行 CheckTheory 函数。例如，检验凯利-莫尔斯集合论中定理 55 及之前的定理:

```
>>> CheckTheory(["Th4","Th5","Th6","Th7","Th8","Th11","Th12","Th14","Th16","Th17","Th19",
"Th20","Th21","Th24","Th26","Th27","Th28","Th29","Th30","Th31","Th32","Th33", "Th34","Th35", "Th37
"Th38","Th39","Th41", "Th42","Th43","Th44","Th46","Th47","Th49","Th50","Th53","Th54","Th55", "T
```

这将根据日志重新生成每个证明。如果最后一行是 Qed，则检查成功。

排中律的证明

这里我们给出一个用 PyLog（使用 AbsC）证明经典有效式 $Av\neg A$ 的证明。为此，我们使用三个有效式:

```
>>> ShowTheorems()
0. D <-> ¬¬D
1. (A -> B) -> (¬B -> ¬A)
2. ¬(A v B) <-> (¬A & ¬B)
```

这些定理的证明很容易。只有定理 0 用到了 AbsC。下面是以“ExcludedMiddle”为名保存的证明。

```
>>> ShowProof()
0. ¬(A v B) <-> (¬A & ¬B) TheoremInt
1. ¬(A v ¬A) <-> (¬A & ¬¬A) PolySub 0
2. (¬(A v ¬A) -> (¬A & ¬¬A)) & ((¬A & ¬¬A) -> ¬(A v ¬A)) EquivExp
3. D <-> ¬¬D TheoremInt
4. (D -> ¬¬D) & (¬¬D -> D) EquivExp
5. ¬A & ¬¬A Hyp
6. ¬A AndElimL 5
7. ¬¬A AndElimR 5
8. ¬¬D -> D AndElimR 4
9. ¬¬A -> A PolySub 8
10. A ImpElim 7 9
11. ¬A & A AndInt 6 10
12. (¬A & ¬¬A) -> (¬A & A) ImpInt 11 Qed
13. ¬(A v ¬A) -> (¬A & ¬¬A) AndElimL 2
14. ¬(A v ¬A) Hyp
15. ¬A & ¬¬A ImpElim 14 13
16. ¬A & A ImpElim 15 12
17. ¬(A v ¬A) -> (¬A & A) ImpInt 16 Qed
18. (A -> B) -> (¬B -> ¬A) TheoremInt
19. (¬(A v ¬A) -> B) -> (¬B -> ¬¬(A v ¬A)) PolySub 18
20. (¬(A v ¬A) -> (¬A & A)) -> (¬(¬A & A) -> ¬¬(A v ¬A)) PolySub 19
21. ¬(¬A & A) -> ¬¬(A v ¬A) ImpElim 17 20
22. ¬¬(A v ¬A) -> (A v ¬A) PolySub 8
23. ¬(¬A & A) Hyp
24. ¬¬(A v ¬A) ImpElim 23 21
25. A v ¬A ImpElim 24 22
26. ¬(¬A & A) -> (A v ¬A) ImpInt 25 Qed
27. ¬A & A Hyp
28. ¬A AndElimL 27
29. A AndElimR 27
30. _|_ ImpElim 29 28
31. ¬(¬A & A) ImpInt 30 Qed
32. A v ¬A ImpElim 31 26 Qed
>>> ShowLog()
0. TheoremInt(2)
1. PolySub(0,"B","neg A")
2. EquivExp(1)
3. TheoremInt(0)
4. EquivExp(3)
5. Hyp("(neg A & neg neg A)")
6. AndElimL(5)
7. AndElimR(5)
8. AndElimR(4)
9. PolySub(8,"D","A")
10. ImpElim(7,9)
11. AndInt(6,10)
12. ImpInt(11,5)
```

```
13. AndElimL(2)
14. Hyp("neg (A v neg A)")
15. ImpElim(14,13)
16. ImpElim(15,12)
17. ImpInt(16,14)
18. TheoremInt(1)
19. PolySub(18,"A","neg (A v neg A)")
20. PolySub(19,"B","(neg A & A)")
21. ImpElim(17,20)
22. PolySub(8,"D","(A v neg A)")
23. Hyp("neg (neg A & A)")
24. ImpElim(23,21)
25. ImpElim(24,22)
26. ImpInt(25,23)
27. Hyp("(neg A & A)")
28. AndElimL(27)
29. AndElimR(27)
30. ImpElim(29,28)
31. ImpInt(30,27)
32. ImpElim(31,26)
```

Gentzen 自动定理证明器

PyLog 中包含一个简单的算法，用于寻找直觉主义命题演算中命题有效式的证明。Smullyan [6] 曾指出，经典命题演算的语义树对应于“上下颠倒”的相继式演算证明。这是自动证明生成的一个范例。直觉主义命题演算（IPC）的无切相继式演算，更具体地说，是限于 IPC 的 Gentzen 系统 G3i（其中通常的结构规则被“吸收”），是这种对应关系的显著例子。事实上，众所周知，将限于命题公式的 G3i“反转”，即可为 IPC [5][4.2.6]提供一个判定过程。对于命题逻辑中的 G3i，我们有以下规则：

$$\frac{}{P,\Gamma \Rightarrow P}\,\mathrm{Ax} \qquad \frac{}{\bot,\Gamma \Rightarrow A}\,\mathrm{L}\bot$$

$$\frac{A,B,\Gamma \Rightarrow C}{A \wedge B,\Gamma \Rightarrow C}\,\mathrm{L}\wedge \qquad \frac{\Gamma \Rightarrow A \qquad \Gamma \Rightarrow B}{\Gamma \Rightarrow A \wedge B}\,\mathrm{R}\wedge$$

$$\frac{A,\Gamma \Rightarrow C \qquad B,\Gamma \Rightarrow C}{A \vee B,\Gamma \Rightarrow C}\,\mathrm{L}\vee \qquad \frac{\Gamma \Rightarrow A_i}{\Gamma \Rightarrow A_0 \vee A_1}\,\mathrm{R}\vee, i = 0,1$$

$$\frac{A \to B,\Gamma \Rightarrow A \qquad B,\Gamma \Rightarrow C}{A \to B,\Gamma \Rightarrow C}\,\mathrm{L}\to \qquad \frac{A,\Gamma \Rightarrow B}{\Gamma \Rightarrow A \to B}\,\mathrm{R}\to$$

这些规则要么没有前提（公理），要么有一个或两个前提。要将证明上下颠倒，显然必须以某种方式“反转”这些规则。对于这类系统中的一个特殊类别（即长度保持且解析的系统，这与子公式性质相对应），我们可以进行“反转”，得到所谓的约化系统，它更便于用于生成证明。证明可视为一系列获胜的线性行动；这些行动作用于有状态系统后，会产生空集。

我们的算法改编自对 [5][4.2.6] 的证明。基本对象是序列列表对象 S，它包含一个相继式列表（s_1, ..., s_n）以及一个 State（m, h），其中包含

已应用于该对象的规则历史 m，以及这些规则生成的所有新相继式的累积集合 m（这样可以避免循环）。规则 r 是一种应用，它接收一个序列列表 S，并生成一个新的序列列表 S'。通过选取一个特定序列 s_k、将其移除，并可能生成新的相继式，再将这些相继式加入移除了 s_k 的原列表中，从而实现序列列表的变换。该规则会添加到 h，新相继式（如果有）则会添加到 m。为了证明公式 f，我们从一个序列对象 $(\Rightarrow f), (\emptyset, ())$ 开始，也就是说，其中只有一个相继式。我们要找到一系列规则 $r_1, ..., r_n$，按顺序应用后，得到形如 $((), (m, h))$ 的序列对象。要理解我们对命题逻辑 G3i 的逆变换，最好的方法是查看 PyLog 主文件夹中 gentzen.py 文件里的源代码。下面来看 Sequent 和 SequentList 类：

```
class Sequent:
  def __init__(self, head,body):
   self.head = head
   self.body = body

(...)

class SequentList:
 def __init__(self,formstring):
  form = astop.NegationExpand(parser.Formula(tokenizer.Tokenize(formstring)))
  seq = Sequent(form,[])
  self.sequentlist = [seq]
  self.memory = []
  self.proof = []

(...)
```

以下是一个规则示例：

```
def rand(seq,n):
 if  n < len(seq.sequentlist)  and seq.sequentlist[n].head.name=="constructor":
  if seq.sequentlist[n].head.operator.name=="&":
   aux1 = seq.sequentlist[:n]
   aux2 = seq.sequentlist[n+1:]
   aux3 = seq.sequentlist[n].body
   left = seq.sequentlist[n].head.left
   right = seq.sequentlist[n].head.right
   new1 = Sequent(left,aux3)
   new2 = Sequent(right, aux3)
   if not SeqInc(new1, seq.memory) and not SeqInc(new2,seq.memory):
    seq.sequentlist = aux1 + [new1,new2] + aux2
    seq.memory.append(new1)
    seq.memory.append(new2)
    seq.proof.append("rand(" + str(n) +")")
   return seq
 return False
```

共有九条规则：rand、ror1、ror2、rimp、land、lor、limp、labs、ax。

此 genzen.py 程序使用与 PyLog 相同的句法引擎，公式的输入方式也相同。请注意，它只能找到直觉主义命题逻辑有效式的证明。

用于寻找证明的算法是一种简单的穷举搜索，并会避免循环。搜索可能需要一些时间。

```
$ python3.9 -i gentzen.py
>>> Auto("( neg (A v B) -> (neg A & neg B))")
>>> Prove()1. rimp(0)2. rand(0)
3. rimp(0)4. imp(0,1)5. ror1(0)
6. ax(0,0)   A, ¬(A v B) => A
7. ax(0,1)   A, _|_ => _|_
8. rimp(0)9. limp(0,1)
10. ror2(0)11. ax(0,0)   B,
 ¬(A v B) => B12. ax(0,1)   B,
 _|_ => _|_
```

```
0.  B, ¬(A v B) => B   Ax01.  B,
 _|_ => _|_   Ax12.  B,
 ¬(A v B) => A v B   Ror2  03.  B,
 ¬(A v B) => _|_   Limp1  2 14.  A,
 ¬(A v B) => A   Ax05.  A, _|_ => _|_   Ax1
6.  A, ¬(A v B) => A v B   Ror1  47.  A,
 ¬(A v B) => _|_   Limp1  6 5
8.  ¬(A v B) => ¬B   Rimp  3
9.  ¬(A v B) => ¬A   Rimp  7
10.  ¬(A v B) => ¬A & ¬B   Rand  9 8
11.  => ¬(A v B) -> (¬A & ¬B)   Rimp  10
```

如上所示，还有一种算法可以根据证明重构完整相继式证明树的线性版本。观察这个线性相继式证明，很容易得到对应的线性自然演绎证明。这将在未来版本的 PyLog 中实现。得到证明序列后，也可以手动测试：依次将规则应用于初始相继式列表对象（即对象 State）。

```
>>> State.display()
0.  => ¬(A v B) -> (¬A & ¬B)
>>> rimp(State,0)
<__main__.SequentList object at 0x109a69070>
>>> State.display()
0.  ¬(A v B) => ¬A & ¬B
>>> rand(State,0)
<__main__.SequentList object at 0x109a69070>
>>> State.display()
0.  ¬(A v B) => ¬A
1.  ¬(A v B) => ¬B
>>> rimp(State,0)
<__main__.SequentList object at 0x109a69070>
```

```
>>> State.display()0.  A, ¬(A v B) => _|_
1.  ¬(A v B) => ¬B>>> limp(State,0,1)
<__main__.SequentList object at 0x109a69070>
>>> State.display()0.  A, ¬(A v B) => A v B
1.  A, _|_ => _|_2.  ¬(A v B) => ¬B
```

等等。未来我们将实现另一种算法，将线性相继式证明树转换为 PyLog 线性自然演绎证明。

形式化范畴论

```
>>> ShowAxioms()
0. ([f: A → B|C] & [g: B → D|C]) -> [(g∘f): A → D|C]
1. A:C -> [id(A): A → A|C]
2. (A:C & [f: A → B|C]) -> (((id(A)∘f) = f) & ((f∘id(A)) = f))
3. ([f: A → B|C] & ([g: B → D|C] & [h: D → E|C])) -> ((h∘(g∘f)) = ((h∘g)∘f))
4. A:C -> Cat(C)
5. [f: A → B|C] -> (A:C & B:C)
6. (Cat(A) & Cat(B)) <-> Cat(func(A,B))
7. F:func(A,B) -> FA:B
8. (F:func(A,B) & [f: a → b|A]) -> [Ff: Fa → Fb|B]
9. (F:func(A,B) & a:A) -> (Fid(A) = id(FA))
10. (F:func(A,B) & ([f: a → b|C] & [g: b → c|C])) -> (F(g∘f) = (Fg∘Ff))
11. ([e: F → G|func(A,B)] & a:A) -> [nat(e,a): Fa → Ga|B]
12. ([e: F → G|func(A,B)] & [f: a → b|A]) -> ((Gf∘nat(e,a)) = (nat(e,b)∘Ff))


>>> ShowDefinitions()
Terminal(A,C) <-> (A:C & ∀B.(B:C -> ∃¹f.[f: B → A|C]))
Isomorphism(f,A,B,C) <-> ([f: A → B|C] & ∃g.([g: B → A|C] & (((g∘f) = id(A)) & ((f∘g)
Iso(A,B,C) <-> ∃f.Isomorphism(f,A,B,C)

>>> ShowProof()
0. Terminal(T,C)  Hyp
1. Terminal(S,C)  Hyp
2. T:C & ∀B.(B:C -> ∃¹f.[f: B → T|C])  DefExp 0
3. S:C & ∀B.(B:C -> ∃¹f.[f: B → S|C])  DefExp 1
4. T:C  AndElimL 2
5. S:C  AndElimL 3
6. ∀B.(B:C -> ∃¹f.[f: B → T|C])  AndElimR 2
7. ∀B.(B:C -> ∃¹f.[f: B → S|C])  AndElimR 3
8. T:C -> ∃¹f.[f: T → T|C]  ForallElim 6
9. S:C -> ∃¹f.[f: S → S|C]  ForallElim 7
10. A:C -> [id(A): A → A|C]  AxInt
11. ∀A.(A:C -> [id(A): A → A|C])  ForallInt 10
12. T:C -> [id(T): T → T|C]  ForallElim 11
```

13. ∀A.(A:C -> [id(A): A → A|C]) ForallInt 10
14. S:C -> [id(S): S → S|C] ForallElim 13
15. [id(T): T → T|C] ImpElim 4 12
16. [id(S): S → S|C] ImpElim 5 14
17. $\exists^1$f.[f: T → T|C] ImpElim 4 8
18. $\exists^1$f.[f: S → S|C] ImpElim 5 9
19. ∃f.([f: T → T|C] & ∀l.([l: T → T|C] -> (l = f))) UniqueElim 17
20. ∃f.([f: S → S|C] & ∀m.([m: S → S|C] -> (m = f))) UniqueElim 18
21. [j: T → T|C] & ∀l.([l: T → T|C] -> (l = j)) Hyp
22. [k: S → S|C] & ∀m.([m: S → S|C] -> (m = k)) Hyp
23. ∀l.([l: T → T|C] -> (l = j)) AndElimR 21
24. ∀m.([m: S → S|C] -> (m = k)) AndElimR 22
25. [j: T → T|C] AndElimL 21
26. [k: S → S|C] AndElimL 22
27. S:C -> $\exists^1$f.[f: S → T|C] ForallElim 6
28. T:C -> $\exists^1$f.[f: T → S|C] ForallElim 7
29. $\exists^1$f.[f: S → T|C] ImpElim 5 27
30. $\exists^1$f.[f: T → S|C] ImpElim 4 28
31. ∃f.([f: S → T|C] & ∀r.([r: S → T|C] -> (r = f))) UniqueElim 29
32. ∃f.([f: T → S|C] & ∀s.([s: T → S|C] -> (s = f))) UniqueElim 30
33. [u: S → T|C] & ∀r.([r: S → T|C] -> (r = u)) Hyp
34. [v: T → S|C] & ∀s.([s: T → S|C] -> (s = v)) Hyp
35. [u: S → T|C] AndElimL 33
36. [v: T → S|C] AndElimL 34
37. ([f: A → B|C] & [g: B → D|C]) -> [(g∘f): A → D|C] AxInt
38. ∀A.(([f: A → B|C] & [g: B → D|C]) -> [(g∘f): A → D|C]) ForallInt 37
39. ([f: T → B|C] & [g: B → D|C]) -> [(g∘f): T → D|C] ForallElim 38
40. ∀B.(([f: T → B|C] & [g: B → D|C]) -> [(g∘f): T → D|C]) ForallInt 39
41. ([f: T → S|C] & [g: S → D|C]) -> [(g∘f): T → D|C] ForallElim 40
42. ∀f.(([f: T → S|C] & [g: S → D|C]) -> [(g∘f): T → D|C]) ForallInt 41
43. ([v: T → S|C] & [g: S → D|C]) -> [(g∘v): T → D|C] ForallElim 42
44. ∀g.(([v: T → S|C] & [g: S → D|C]) -> [(g∘v): T → D|C]) ForallInt 43
45. ([v: T → S|C] & [u: S → D|C]) -> [(u∘v): T → D|C] ForallElim 44
46. ∀D.(([v: T → S|C] & [u: S → D|C]) -> [(u∘v): T → D|C]) ForallInt 45
47. ([v: T → S|C] & [u: S → T|C]) -> [(u∘v): T → T|C] ForallElim 46
48. [v: T → S|C] & [u: S → T|C] AndInt 36 35
49. [(u∘v): T → T|C] ImpElim 48 47
50. ∀A.(([f: A → B|C] & [g: B → D|C]) -> [(g∘f): A → D|C]) ForallInt 37
51. ([f: S → B|C] & [g: B → D|C]) -> [(g∘f): S → D|C] ForallElim 50
52. ∀B.(([f: S → B|C] & [g: B → D|C]) -> [(g∘f): S → D|C]) ForallInt 51
53. ([f: S → T|C] & [g: T → D|C]) -> [(g∘f): S → D|C] ForallElim 52
54. ∀D.(([f: S → T|C] & [g: T → D|C]) -> [(g∘f): S → D|C]) ForallInt 53
55. ([f: S → T|C] & [g: T → S|C]) -> [(g∘f): S → S|C] ForallElim 54
56. ∀f.(([f: S → T|C] & [g: T → S|C]) -> [(g∘f): S → S|C]) ForallInt 55
57. ([u: S → T|C] & [g: T → S|C]) -> [(g∘u): S → S|C] ForallElim 56
58. ∀g.(([u: S → T|C] & [g: T → S|C]) -> [(g∘u): S → S|C]) ForallInt 57
59. ([u: S → T|C] & [v: T → S|C]) -> [(v∘u): S → S|C] ForallElim 58
60. [u: S → T|C] & [v: T → S|C] AndInt 35 36
61. [(v∘u): S → S|C] ImpElim 60 59
62. [id(T): T → T|C] -> (id(T) = j) ForallElim 23

```
63. id(T) = j  ImpElim 15 62
64. [(u∘v): T → T|C] -> ((u∘v) = j)  ForallElim 23
65. (u∘v) = j  ImpElim 49 64
66. [id(S): S → S|C] -> (id(S) = k)  ForallElim 24
67. id(S) = k  ImpElim 16 66
68. [(v∘u): S → S|C] -> ((v∘u) = k)  ForallElim 24
69. (v∘u) = k  ImpElim 61 68
70. j = id(T)  Symmetry 63
71. k = id(S)  Symmetry 67
72. (u∘v) = id(T)  EqualitySub 65 70
73. (v∘u) = id(S)  EqualitySub 69 71
74. ((u∘v) = id(T)) & ((v∘u) = id(S))  AndInt 72 73
75. [u: S → T|C] & (((u∘v) = id(T)) & ((v∘u) = id(S)))  AndInt 35 74
76. ∃h.([h: S → T|C] & (((h∘v) = id(T)) & ((v∘h) = id(S))))  ExistsInt 75
77. [v: T → S|C] & ∃h.([h: S → T|C] & (((h∘v) = id(T)) & ((v∘h) = id(S))))  AndInt 36
78. Isomorphism(v,T,S,C)  DefSub 77
79. ∃f.Isomorphism(f,T,S,C)  ExistsInt 78
80. Iso(T,S,C)  DefSub 79
81. Iso(T,S,C)  ExistsElim 19 21 80
82. Iso(T,S,C)  ExistsElim 20 22 81
83. Iso(T,S,C)  ExistsElim 31 33 82
84. Iso(T,S,C)  ExistsElim 32 34 83
85. Terminal(S,C) -> Iso(T,S,C)  ImpInt 84
86. Terminal(T,C) -> (Terminal(S,C) -> Iso(T,S,C))  ImpInt 85 Qed
```

超越类比原理

处理涉及代数结构的商的形式证明很繁琐。我们考虑一个类 U，其中包含给定结构的所有不同模型（或同构类的代表元）。我们为作用于整个 $\bigcup U$ 的运算 f_1、...、f_n 固定一个公理系统；通过限制这些运算，它们作用于 U 的所有元素 M，并分别对应于这些结构中的各种运算。这类似于 Weil 为代数几何奠定的基础：在其中，所有有理函数域都被视为某个大域的子域。商（例如群或环的商）不是根据结构来定义的，而是根据满同态 f：$A \to B$ 以及定义域 A 的给定子结构（正规子群、理想）来定义的。

二阶算术

```
The project is to formalise Stephen G. Simpson's book Subsystems of
Second-Order Arithmetic. The axioms of Z2 are interpreted as follows:

0. Nat(x) v Set(x)
1. Nat(x) -> ¬Set(x)
2. Nat(0) & Nat(1)
3. Nat(n) -> ¬((n + 1) = 0)
4. (Nat(n) & Nat(m)) -> (((m + 1) = (n + 1)) -> (m = n))
5. Nat(m) -> ((m + 0) = m)
```

```
6. (Nat(n) & Nat(m)) -> ((m + (n + 1)) = ((m + n) + 1))
7. Nat(m) -> ((m * 0) = 0)
8. (Nat(n) & Nat(m)) -> ((m * (n + 1)) = ((m * n) + m))
9. Nat(m) -> ¬(m < 0)
10. (Nat(n) & Nat(m)) -> ((m < (n + 1)) <-> ((m < n) v (m = n)))
11. (x ϵ X) -> (Nat(x) & Set(X))
12. ((0 ϵ X) & ∀n.((n ϵ X) -> ((n + 1) ϵ X))) -> ∀n.(n ϵ X)

n ϵ { x: form(x)}
-----------------
Nat(n) & form(n)


Nat(n) & form(n)
-----------------
n ϵ { x: form(x)}

The comprehension scheme is converted into the two rules above which are similar
to ClassElim and ClassInt used to formalise Kelley-Morse
set theory. Example of a simple proof:

0. 1 = 0  Hyp
1. Nat(0) & Nat(1)  AxInt
2. 0 = 0  Identity
3. Nat(0)  AndElimL 1
4. Nat(n) -> ¬((n + 1) = 0)  AxInt
5. ∀n.(Nat(n) -> ¬((n + 1) = 0))  ForallInt 4
6. Nat(0) -> ¬((0 + 1) = 0)  ForallElim 5
7. ¬((0 + 1) = 0)  ImpElim 3 6
8. ¬((0 + 0) = 0)  EqualitySub 7 0
9. Nat(m) -> ((m + 0) = m)  AxInt
10. ∀m.(Nat(m) -> ((m + 0) = m))  ForallInt 9
11. Nat(0) -> ((0 + 0) = 0)  ForallElim 10
12. (0 + 0) = 0  ImpElim 3 11
13. ¬(0 = 0)  EqualitySub 8 12
14. _|_  ImpElim 2 13
15. ¬(1 = 0)  ImpInt 14 Qed
```

欧几里得几何

有关欧几里得第一定理在 PyLog 中的形式化，请参见 [2]。

参考文献

[1] Protin, C. L.（2022）。欧几里得逻辑。http://dx.doi.org/10.13140/RG.2.2.15673.65120。

[2]Protin, C. L.（2022）。PyLog 中的凯利-莫尔斯集合论。
http://dx.doi.org/10.13140/RG.2.2.13288.14081。

[3] D. Prawitz, 《自然演绎》, Dover。[4] J. Kelley, 《一般拓扑学》。[5] A. Troelstra, 《数学中的构造主义》第一卷。[6] Raymond M. Smullyan, 《一阶逻辑》, Dover, 1995年。[7] J.-Y. Girard, 《证明与类型》, 剑桥大学出版社, 1989年。[8] A.S. Troelstra、H. Schwichtenberg, 《基础证明论》, 剑桥理论计算机科学丛书, 第二版, 2000年。