Introduction to PyLog

Clarence Protin

November 8, 2021

Abstract

Introduction

As Voevodsky pointed out in a public lecture in Princeton, it is highly desirable to obtain a foundations of mathematics that will allow automatic verification of proofs. This task is usually carried out in the higher-order context of lambda calculus and type theory with the bonus of the constructive computational information furnished by the Curry-Howard isomorphism: contemporary examples are the promising formal mathematics projects employing Coq, Mizar, Isabelle and specially various approaches to implementing Voevodsky's Homotopy Type Theory such as Cubicaltt. PyLog is a computationally and philosophically alternative approach which aims at fulfilling a number of desiderata:

1. The formal environment must be easy, simple, intuitive and attractive to use for the average logician or mathematician.
2. The process of writing proofs (and the checking algorithm) should resemble as closely as possible the structure of actual mathematical proofs and their process of elaboration.
3. It should be first-order with all its type-free simplicity and versatility and yet not be necessarily based on ZFC.
4. It should be easy to combine different formalised theories.
5. It should be easy to write formalised proofs such that they can be easily checked either by a human or a machine; we should obtain a "perfect bridge" between programming and mathematics.
6. The difficulty of formalising and checking a given theory should not exceed mathematical difficulty of the theory involved.
7. Classical logic is to be seen as an extension of intuitionistic logic and the user is free to use the classical negation rule or not.

The key ingredients that I propose are:

- A linearised natural deduction for the predicate calculus extended with a Kelley-Morse style extension operator.
- A first-order approach to a "reverse mathematics" style formalisation of interesting fragments of mathematics.

The Logic of PyLog

PyLog is based on the natural deduction presentation of first-order predicate logic with equality endowed with rules for a class-forming operator $\{x : P(x)\}$ and a primitive binary predicate $\in$. Pylog also includes second-order variables allowing us to instantiate logical validities. The language of PyLog consists of finite sets of constants, (first-order) variables, second-order variables, function symbols of different arities $n > 0$, predicate symbols of different arities $n < 0$ and the special symbol $\perp$. Terms and formulas are defined by mutual recursion:

- A constant c is a term.
- A variable x is a term.
- If $t_1, ...t_n$ are terms and f is a n-ary function symbol then $f(t_1, ..., t_n)$ is a term.
- If A is a formula and x is a variable then $\{x : A\}$ is a term (called an *extension*)
- If P is a n-ary predicate symbol and $t_1, ...t_n$ are terms then $P(t_1, ..., t_n)$is a formula.
- If t and s are terms then $t = s$ is a formula (this is a particular case of the last condition).
- If $\mathfrak{A}$ is a second-order variable then it is a formula.
- If A and B are formulas then $A \vee B$, $A \ \& \ B$, $A \to B$ are formulas.
- If x is a variable and A is a formula then $\forall x.A$ and $\exists x.A$ are formulas.
- $\perp$ is a formula.

We define the set $FV(e)$ of *free variables* of an expression e (term or formula) as follows:

- $FV(c) = \emptyset$
- $FV(x) = \{x\}$
- $FV(f(t_1, ..., t_n)) = \bigcup_{i=1,...,n} FV(t_i)$
- $FV(P(t_1, ..., t_n)) = \bigcup_{i=1,...,n} FV(t_i)$
- $FV(\perp) = \emptyset$
- $FV(\mathfrak{A}) = \emptyset$
- $FV(A \vee B)$, $FV(A \ \& \ B)$ and $FV(A \to B)$ are equal to $FV(A) \cup FV(B)$
- $FV(\forall x.A)$, $FV(\exists x.A)$ and $FV(\{x : A\})$ are equal to $FV(A)/\{x\}$

As usual we consider expression *modulo* the renaming of quantified variables or variables within the scope of an extension: in any subexpression of the form $\forall x.A$, $\exists x.A$ or $\{x : A\}$ we may rename x and all free occurrences of x in A to a fresh variable y as long y does not occur within the scope of some quantifier $\forall y$, $\exists y$ or extension $\{y : ...\}$. When we write $A[t/x]$ we assume that the bound variables of A have been renamed so as to be distinct from $FV(t)$ (this is a slightly stronger condition than we actually need).

In PyLog proofs are always in the context of a *proof environment*. This consisting of:

- A list formulas called *axioms*

- A list of formulas called *assumed theorems*
- A list of *defining equations* for constants or function symbols of the form $c = A$ or $f(x_1, ..., x_n) = A$
- A list of *predicate definitions* consisting of triples $(P, (x_1, ..., x_n), A)$ defining n-ary predicate symbols P. Here $FV(A) \subseteq \{x_1, ..., x_n\}$. Triples are also denoted by $P(x_1, ..., x_n) \equiv A$.

All lists above may be empty. In PyLog the default proof environment consists of a single predicate definition for $Set(x)$ defined as $\exists y.x \in y$. The language is endowed further with the primitive binary predicate $\in$. There are no other functions, predicates or constants.

The proof system of PyLog is based on a linearised variant of natural deduction with conservative second-order order extension.

We first present the system in the standard form. We assume the reader is familiar with proof trees and the concept of dependency(the first chapters of [1] are sufficient). The proof system of PyLog consists of the following. We have *purely logical rules* which are the rules for *minimal predicate calculus* plus the intuitionistic and classical negation rules:

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

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

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

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

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

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

The proviso for ForallInt is that y cannot occur in any assumption on which A depends and the proviso for ExistsElim is that y does not occur in $\exists y.A$ or in C or on any hypothesis on which C depends other than $[A[y/x]]$. $\sim A$ is syntactic sugar for $A \to \bot$.

We have the *class rules*:

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

These rules express the *classification axiom scheme* of Kelley-Morse set theory such as formulated in the appendix of [2].

We have also the *equality rules*[1]:

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

[1]this is inspired by the treatment in [3]

where in EqualitySub A' is A when a specified number of occurrences of t are replace by s. EqualitySub is of fundamental importance in using defined constants and function symbols of the proof environment.

We have then our *second-order rule*:

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

where A' results from A by substituting all occurrences of $\mathfrak{A}$ in A by a formula B. The proviso is that $\mathfrak{A}$ does not occur in any hypothesis on which A depends and that no free variable in B becomes bound after the substitution.

Remark 0.1 This rule is to be understood as a combination of an invisible second-order generalisation of the variable $\mathfrak{A}$ followed by an instantiation by B.

The final set of rules concern how information in the proof environment is introduced into the proof.

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

The first three rules simply add the nth formula in the lists of axioms, assumed theorems and defining equations respectively. DefExp does the following. Assume we have a definition $P(x_1, ..., x_n) \equiv B$. Then DefExp replaces specified occurrences of subformulas of the form $P(t_1, ..., t_n)$ by $B[t_1/x_1, ..., t_n/x_n]$ (the resulting expression is denoted by A' in the rule). DefSub does the inverse of this. For chosen $t_1, ..., t_n$ we must specify the occurrences of expressions of the form $B[t_1/x_1, ..., t_n/x_n]$ in A which we wish to "collapse" into $P(t_1, ..., t_n)$[2].

When we wish to use defined functions or constants we first introduce the defining equalities into our proof by means o $\mathrm{DefEqInt}_n$ and then make use of EqSub. The above are the core rules of PyLog. We introduce the usual abbreviation $A \leftrightarrow B$ and so finally have two rules to toggle this notation:

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

Remark 0.2 Certain combinations of rules occur frequently and it is convenient to have dervived rules(or shortcuts) such as

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

for

[2]For example if we had the definition $Set(x) \leftrightarrow \exists y.x \in y$ and a line in our proof such as

$$1.Set(x)$$

then calling DefExp(n,"Set",[0]) would add the line

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

On the other hand if we had a line

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

then the command DefSub(3,"Set", ["x"],[0]) would yield

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

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

and two other obvious shortcuts EquivRight and EquivLeft. Also the important derived rule

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

for

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

Linearised Natural Deduction

A linear proof in PyLog (for a given proof environment) is a list of *proof elements*. Each proof element p is a triple (A, par, dis) where A is a formula and par and dis are lists of integers. If p occurs in position m (we say that m is p's number) and the list has length m then the elements of par (parents) must be strictly less than n and those dis(discharges) strictly larger than n. Rules are applied by adding a new proof element to the end of the list and possibly updating previous proof entries. Given a linear proof and an element p we can recursively backtrack the parents to obtain a *dependency tree* of proof-elements. The dependencies of p are obtained by taking the set of leaves of this tree are removing those proof-elements having dis containing a number which occurs in the dependency tree. It is also easy to see how given a proof tree we can obtain a linear proof. In PyLog rules have the general format

$$(Name, Parents, Parameters)$$

where Name is the name of the rule, Parents is the list of the number previous elements which the rule is applied to and Parameters can constain formulas, terms, variables, position lists, etc. In PyLog rules are entered as Python function. The arguments will specify all the required information in Parents and Parameters.

The PyLog command Qed(ForNum) checks if the formula has discharged all its assumptions.

It is helpful to look at a snippet in the definition of some classes:

```
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 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

   (...)
```

List of Core PyLog Rules

```
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 Commands

We have a list of commands for setting up the proof environment, that is, for introducing the axioms, assumed theorems, defined constants and symbols and defined predicates that will be used in the proof.

```
Hyp(Formula)

NewAx(Formula)

NewDef(PredicateName, ArgList, Formula)
```

```
AddConstants(NameList)

AddFunction(FunctionName, Arity, PrefixBool)

NewDefEq(EquationFormula)

AddTheorem(Formula)
```

Hyp introduces a formula as a hypothesis.

When defining functions with NeDefEq() we must first use AddFunction() specifying the name, arity and whether the function is to be displayed with prefix or infix notation (for binary functions). For constants we use AddConstant(). We also must take care that we have enough variables via the AddVariables(VarList) function.

Then we have a list of commands which displays information about the current proof and proof environment. ShowDefinitions() displays the defined predicates. ShowDefEquations() displays the defined constants and functions. ShowAxioms() displays the axioms. ShowTheorems() displays the assumed theorems which may be used in the proof (it is not advisable to alter this list during the proof). ShowProof() displays the current state of the proof and ShowLog() shows the list of previous succesful rule commands which constitute the proof. We also have a command Undo() which deletes the last element of the proof.

If a theorem has already been saved you can view the conclusion with ViewTheorem(Name) or add it directly to the proof environment with the LoadTheorem(Name) command - *provided that the required environment has been previously loaded.*

Using Pylog

In PyLog a *theory* is a directory whose files are *theorems*. A theorem consists of both a proof environment and a proof in this environment (either complete or incomplete). All theorems in a theory should ideally have the same proof environment. The theorem to be proved ideally should occur at the end of the proof and have been tested with the command Qed(Number). There should also be an "empty" theorem which is to be seen as the proof environment that must be loaded in order to start writing a new theorem. The command Load(Name) loads a proof environment or theorem and the command Save(Name) will save the current proof environment or theorem. The command ViewTheorem(Name) will not load anything but only display the last line of the proof. The command ViewTheory(DirName) will likewise display all the theorems in the directory.

To use Pylog Python 3.∗ is required. PyLog runs from a terminal through the Python CLI. Clone the repository[3] on GitHub, enter the folder, and enter

```
$ python -i proofenvironment.py

Welcome to PyLog 1.0

Natural Deduction Proof Assistant and Proof Checker

(c) 2020  C. Lewis Protin

>>>
```

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

In the PyLog folder we have the saved Kelley-Morse environment. We load this by Load("Kelley-Morse"). When a command is succesful PyLog will return True. We can now examine the axioms and definitions:

```
>>> 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}}
12. proj1(x) = ∩∩x
13. proj2(x) = (∩∪x ∪ (∪∪x ~ ∪∩x))
14. (a∘b) = {w: ∃x.∃y.∃z.((((x,y) ϵ a) & ((y,z) ϵ b)) & (w = (x,z)))}
15. (r)^-1 = {z: ∃x.∃y.(((x,y) ϵ r) & (z = (y,x)))}
16. domain(f) = {x: ∃y.((x,y) ϵ f)}
17. range(f) = {y: ∃x.((x,y) ϵ f)}
18. (f'x) = ∩{y: ((x,y) ϵ f)}
19. (x X y) = {z: ∃a.∃b.((z = (a,b)) & ((a ϵ x) & (b ϵ y)))}
20. func(x,y) = {f: (Function(f) & ((domain(f) = x) & (range(f) = y)))}
21. E = {z: ∃x.∃y.((z = (x,y)) & (x ϵ 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 ϵ y)
(x ⊂ y) <-> ∀z.((z ϵ x) -> (z ϵ y))
Relation(r) <-> ∀z.((z ϵ r) -> ∃x.∃y.(z = (x,y)))
Function(f) <-> (Relation(f) & ∀x.∀y.∀z.((((x,y) ϵ f) & ((x,z) ϵ f)) -> (y = z)))
Trans(r) <-> ∀x.∀y.∀z.((((x,y) ϵ r) & ((y,z) ϵ r)) -> ((x,z) ϵ r))
Connects(r,x) <-> ∀y.∀z.(((y ϵ x) & (z ϵ x)) -> ((y = z) v (((y,z) ϵ r) v ((z,y) ϵ r))))
Asymmetric(r,x) <-> ∀y.∀z.(((y ϵ x) & (z ϵ x)) -> (((y,z) ϵ r) -> ¬((z,y) ϵ r)))
First(r,x,z) <-> ((z ϵ x) & ∀y.((y ϵ x) -> ¬((y,z) ϵ 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 ϵ x)
```

```
 & (v ϵ y)) & ((u,v) ϵ r)) -> (u ϵ y)))
OrderPreserving(f,r,s) <-> ((Function(f) & (WellOrders(r,domain(f))
 & WellOrders(r,range(f)))) & ∀u.∀v.((((u ϵ domain(f))
 & (v ϵ domain(f))) & ((u,v) ϵ r)) -> (((f'u),(f'v)) ϵ r)))
1-to-1(f) <-> (Function(f) & Function((f)^-1))
Full(x) <-> ∀y.((y ϵ x) -> (y ⊂ x))
Ordinal(x) <-> (Full(x) & Connects(E,x))
Integer(x) <-> (Ordinal(x) & WellOrders((E)^-1,x))
Choice(f) <-> (Function(f) & ∀y.((y ϵ domain(f)) -> ((f'y) ϵ y)))
Equi(x,y) <-> ∃f.(1-to-1(f) & ((domain(f) = x) & (range(f) = y)))
Card(x) <-> (Ordinal(x) & ∀y.(((y ϵ x) & (y ϵ ord)) -> ¬Equi(y,x)))
TransIn(r,x) <-> ∀u.∀v.∀w.(((u ϵ x) & ((v ϵ x) & (w ϵ x))) ->
 ((((u,v) ϵ r) & ((v,w) ϵ r)) -> ((u,w) ϵ r)))
```

We can also check by ShowProof() that the proof is empty. By default expressions are displayed using pretty printing (Unicode character) which can use infix notation. The pretty printing can be changed via parser.prettyprint[FunctionNameString] = PrettyString. Expressions are entered in a strictly functional way (with the exception of logical connectives, extensions and quantifiers).

```
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)^-1
restrict(f,x)             (f|x)
int                        ω
```

Note that $\neg A$ is the pretty print for $A \to \bot$. When using the rules of Pylog we must think of $\neg A$ this way. Conjunction is usually entered in infix style (A &) but PyLog will create and group parenthesis to the right: thus (A & B & C) is interpreted as (A & (B & C)).

First Proof in Pylog

In this section we prove theorem 4 of [2] in the Kelley-Morse proof environment:

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

We give here the log of session proving the first half of the theorem.

```
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))")
True
>>> ShowProof()
0. z ϵ (x ∪ y) Hyp
>>> 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}}
12. proj1(x) = ∩∩x
13. proj2(x) = (∩∪x ∪ (∪∪x ~ ∪∩x))
14. (a∘b) = {w: ∃x.∃y.∃z.((((x,y) ϵ a) & ((y,z) ϵ b)) & (w = (x,z)))}
15. (r)^-1 = {z: ∃x.∃y.(((x,y) ϵ r) & (z = (y,x)))}
16. domain(f) = {x: ∃y.((x,y) ϵ f)}
17. range(f) = {y: ∃x.((x,y) ϵ f)}
18. (f'x) = ∩{y: ((x,y) ϵ f)}
19. (x X y) = {z: ∃a.∃b.((z = (a,b)) & ((a ϵ x) & (b ϵ y)))}
20. func(x,y) = {f: (Function(f) & ((domain(f) = x) & (range(f) = y)))}
21. E = {z: ∃x.∃y.((z = (x,y)) & (x ϵ 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)
True
>>> ShowProof()
0. z ϵ (x ∪ y) Hyp
1. (x ∪ y) = {z: ((z ϵ x) v (z ϵ y))} DefEqInt
>>> EqualitySub(0,1,[0])
True
```

```
>>> ShowProof()
0. z ϵ (x ∪ y) Hyp
1. (x ∪ y) = {z: ((z ϵ x) v (z ϵ y))} DefEqInt
2. z ϵ {z: ((z ϵ x) v (z ϵ y))} EqualitySub 0 1
>>> ClassElim(2)
True
>>> ShowProof()
0. z ϵ (x ∪ y) Hyp
1. (x ∪ y) = {z: ((z ϵ x) v (z ϵ y))} DefEqInt
2. z ϵ {z: ((z ϵ x) v (z ϵ y))} EqualitySub 0 1
3. Set(z) & ((z ϵ x) v (z ϵ y)) ClassElim 2
>>> AndElimR(3)
True
>>> ShowProof()
0. z ϵ (x ∪ y) Hyp
1. (x ∪ y) = {z: ((z ϵ x) v (z ϵ y))} DefEqInt
2. z ϵ {z: ((z ϵ x) v (z ϵ y))} EqualitySub 0 1
3. Set(z) & ((z ϵ x) v (z ϵ y)) ClassElim 2
4. (z ϵ x) v (z ϵ y) AndElimR 3
>>> ImpInt(4,0)
True
>>> ShowProof()
0. z ϵ (x ∪ y) Hyp
1. (x ∪ y) = {z: ((z ϵ x) v (z ϵ y))} DefEqInt
2. z ϵ {z: ((z ϵ x) v (z ϵ y))} EqualitySub 0 1
3. Set(z) & ((z ϵ x) v (z ϵ y)) ClassElim 2
4. (z ϵ x) v (z ϵ y) AndElimR 3
5. (z ϵ (x ∪ y)) -> ((z ϵ x) v (z ϵ y)) ImpInt 4
>>> Qed(5)
True
>>> ShowProof()
0. z ϵ (x ∪ y) Hyp
1. (x ∪ y) = {z: ((z ϵ x) v (z ϵ y))} DefEqInt
2. z ϵ {z: ((z ϵ x) v (z ϵ y))} EqualitySub 0 1
3. Set(z) & ((z ϵ x) v (z ϵ y)) ClassElim 2
4. (z ϵ x) v (z ϵ y) AndElimR 3
5. (z ϵ (x ∪ y)) -> ((z ϵ x) v (z ϵ y)) ImpInt 4 Qed
>>> Hyp("(Elem(z,x) v Elem(z,y))")
True
>>> ShowProof()
0. z ϵ (x ∪ y) Hyp
1. (x ∪ y) = {z: ((z ϵ x) v (z ϵ y))} DefEqInt
2. z ϵ {z: ((z ϵ x) v (z ϵ y))} EqualitySub 0 1
3. Set(z) & ((z ϵ x) v (z ϵ y)) ClassElim 2
4. (z ϵ x) v (z ϵ y) AndElimR 3
5. (z ϵ (x ∪ y)) -> ((z ϵ x) v (z ϵ y)) ImpInt 4 Qed
6. (z ϵ x) v (z ϵ y) Hyp
>>> Hyp("Elem(z,x)")
True
>>> ShowProof()
```

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

```
4. (z ϵ x) v (z ϵ y) AndElimR 3
5. (z ϵ (x ∪ y)) -> ((z ϵ x) v (z ϵ y)) ImpInt 4 Qed
6. (z ϵ x) v (z ϵ y) Hyp
7. z ϵ x Hyp
8. ∃x.(z ϵ x) ExistsInt 7
9. Set(z) DefSub 8
>>> Hyp("Elem(z,y)")
True
>>> ShowProof()
0. z ϵ (x ∪ y) Hyp
1. (x ∪ y) = {z: ((z ϵ x) v (z ϵ y))} DefEqInt
2. z ϵ {z: ((z ϵ x) v (z ϵ y))} EqualitySub 0 1
3. Set(z) & ((z ϵ x) v (z ϵ y)) ClassElim 2
4. (z ϵ x) v (z ϵ y) AndElimR 3
5. (z ϵ (x ∪ y)) -> ((z ϵ x) v (z ϵ y)) ImpInt 4 Qed
6. (z ϵ x) v (z ϵ y) Hyp
7. z ϵ x Hyp
8. ∃x.(z ϵ x) ExistsInt 7
9. Set(z) DefSub 8
10. z ϵ y Hyp
>>> ExistsInt(10, "y","x",[0])
True
>>> ShowProof()
0. z ϵ (x ∪ y) Hyp
1. (x ∪ y) = {z: ((z ϵ x) v (z ϵ y))} DefEqInt
2. z ϵ {z: ((z ϵ x) v (z ϵ y))} EqualitySub 0 1
3. Set(z) & ((z ϵ x) v (z ϵ y)) ClassElim 2
4. (z ϵ x) v (z ϵ y) AndElimR 3
5. (z ϵ (x ∪ y)) -> ((z ϵ x) v (z ϵ y)) ImpInt 4 Qed
6. (z ϵ x) v (z ϵ y) Hyp
7. z ϵ x Hyp
8. ∃x.(z ϵ x) ExistsInt 7
9. Set(z) DefSub 8
10. z ϵ y Hyp
11. ∃x.(z ϵ x) ExistsInt 10
>>> DefSub(11,"Set","z",[0])
True
>>> ShowProof()
0. z ϵ (x ∪ y) Hyp
1. (x ∪ y) = {z: ((z ϵ x) v (z ϵ y))} DefEqInt
2. z ϵ {z: ((z ϵ x) v (z ϵ y))} EqualitySub 0 1
3. Set(z) & ((z ϵ x) v (z ϵ y)) ClassElim 2
4. (z ϵ x) v (z ϵ y) AndElimR 3
5. (z ϵ (x ∪ y)) -> ((z ϵ x) v (z ϵ y)) ImpInt 4 Qed
6. (z ϵ x) v (z ϵ y) Hyp
7. z ϵ x Hyp
8. ∃x.(z ϵ x) ExistsInt 7
9. Set(z) DefSub 8
10. z ϵ y Hyp
11. ∃x.(z ϵ x) ExistsInt 10
```

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

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

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

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

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

The proof is fully codified by the log (and the proof environment):

```
>>> 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)
```

Similarly the second half of the conjunction is proven:

```
21. z ϵ (x ∩ y) Hyp
22. (x ∩ y) = {z: ((z ϵ x) & (z ϵ y))} DefEqInt
23. z ϵ {z: ((z ϵ x) & (z ϵ y))} EqualitySub 21 22
24. Set(z) & ((z ϵ x) & (z ϵ y)) ClassElim 23
25. (z ϵ x) & (z ϵ y) AndElimR 24
26. (z ϵ (x ∩ y)) -> ((z ϵ x) & (z ϵ y)) ImpInt 25 Qed
27. (z ϵ x) & (z ϵ y) Hyp
```

```
28. z ϵ x AndElimL 27
29. ∃x.(z ϵ x) ExistsInt 28
30. Set(z) DefSub 29
31. Set(z) & ((z ϵ x) & (z ϵ y)) AndInt 30 27
32. z ϵ {z: ((z ϵ x) & (z ϵ y))} ClassInt 31
33. {z: ((z ϵ x) & (z ϵ y))} = (x ∩ y) Symmetry 22
34. z ϵ (x ∩ y) EqualitySub 32 33
35. ((z ϵ x) & (z ϵ y)) -> (z ϵ (x ∩ y)) ImpInt 34 Qed
36. ((z ϵ (x ∩ y)) -> ((z ϵ x) & (z ϵ y))) & (((z ϵ x) & (z ϵ y)) -> (z ϵ (x ∩ y))) AndIn
37. (z ϵ (x ∩ y)) <-> ((z ϵ x) & (z ϵ y)) EquivConst  Qed
38. ((z ϵ (x ∪ y)) <-> ((z ϵ x) v (z ϵ y))) & ((z ϵ (x ∩ y)) <-> ((z ϵ x) & (z ϵ y))) And
```

with 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)
```

This theorem comes in the main directory of the PyLog repository and can be loaded via Load("Th4"). We also show the proof of the first half of the conjunction of theorem 5 of [2] which illustrates how we use previous theorems and apply an axiom:

```
0. z ϵ (x ∪ x) Hyp
1. ((z ϵ (x ∪ y)) <-> ((z ϵ x) v (z ϵ y))) & ((z ϵ (x ∩ y)) <-> ((z ϵ x) & (z ϵ y))) Theo
2. (z ϵ (x ∪ y)) <-> ((z ϵ x) v (z ϵ y)) AndElimL 1
3. ((z ϵ (x ∪ y)) -> ((z ϵ x) v (z ϵ y))) & (((z ϵ x) v (z ϵ y)) -> (z ϵ (x ∪ y))) EquivE
4. (z ϵ (x ∪ y)) -> ((z ϵ x) v (z ϵ y)) AndElimL 3
5. ∀y.((z ϵ (x ∪ y)) -> ((z ϵ x) v (z ϵ y))) ForallInt 4
6. (z ϵ (x ∪ x)) -> ((z ϵ x) v (z ϵ x)) ForallElim 5
7. (z ϵ x) v (z ϵ x) ImpElim 0 6
8. z ϵ x Hyp
9. z ϵ x Hyp
10. z ϵ x OrElim 7 8 8 9 9
11. (z ϵ (x ∪ x)) -> (z ϵ x) ImpInt 10 Qed
12. z ϵ x Hyp
13. (z ϵ x) v (z ϵ x) OrIntL 12
```

```
14. ((z ϵ x) v (z ϵ y)) -> (z ϵ (x ∪ y)) AndElimR 3
15. ∀y.(((z ϵ x) v (z ϵ y)) -> (z ϵ (x ∪ y))) ForallInt 14
16. ((z ϵ x) v (z ϵ x)) -> (z ϵ (x ∪ x)) ForallElim 15
17. z ϵ (x ∪ x) ImpElim 13 16
18. (z ϵ x) -> (z ϵ (x ∪ x)) ImpInt 17 Qed
19. ((z ϵ (x ∪ x)) -> (z ϵ x)) & ((z ϵ x) -> (z ϵ (x ∪ x))) AndInt 11 18
20. (z ϵ (x ∪ x)) <-> (z ϵ x) EquivConst
21. ∀z.((z ϵ (x ∪ x)) <-> (z ϵ x)) ForallInt 20 Qed
22. ∀x.∀y.((x = y) <-> ∀z.((z ϵ x) <-> (z ϵ y))) AxInt
23. ∀y.(((x ∪ x) = y) <-> ∀z.((z ϵ (x ∪ x)) <-> (z ϵ y))) ForallElim 22
24. ((x ∪ x) = x) <-> ∀z.((z ϵ (x ∪ x)) <-> (z ϵ x)) ForallElim 23
25. (((x ∪ x) = x) -> ∀z.((z ϵ (x ∪ x)) <-> (z ϵ x))) & (∀z.((z ϵ (x ∪ x)) <-> (z ϵ x))
26. ∀z.((z ϵ (x ∪ x)) <-> (z ϵ x)) -> ((x ∪ x) = x) AndElimR 25
27. (x ∪ x) = x ImpElim 21 26 Qed
```

the log is

```
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)
```

To prove Th6 and Th7 we could make use of PolySub and previously proven propositional validities.
For instance if a logical validity was previously proven

```
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)
```

Suppose we saved this theorem as "Log1". Then we can add this theorem to our environment and instantiate it via PolySub to, for instance:

```
>>> 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)")
True
>>> PolySub(1,"B","Elem(z,y)")
True
>>> ShowProof()
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)")
```

Kelley-Morse Set Theory Project

Our project is to have a complete verified formalisation of all the theorems of Set Theory in the Appendix of [2].

Proof of Law of Excluded Middle

The classical validity is included as a theorem because it is so convenient for classical reasoning. Here is a formal proof using the classical negation rule.

1. $\sim\sim X$ Hyp
2. $\sim X$ Hyp
3. $\perp$ ImpElim 2 1
4. X AbsC 3 2
5. $\sim\sim X \supset X$ ImpInt 4 1 Qed
6. X Hyp
7. $\sim X$ Hyp
8. $\perp$ ImpElim 6 7
9. $\sim\sim X$ ImpInt 8 7
10. $X \supset \sim\sim X$ ImpInt 9 6 Qed
11. $A \supset B$ Hyp
12. $\sim B$ Hyp
13. A Hyp
14. B ImpElim 13 11
15. $\perp$ ImpElim 14 12
16. $\sim A$ ImpInt 15 13
17. $\sim B \supset \sim A$ ImpInt 16 12
18. $(A \supset B) \supset (\sim B \supset \sim A)$ ImpInt 17 11 Qed
19. $\sim A \& A$ Hyp
20. A AndElimR 19
21. $\sim A$ AndElimL 19
22. $\perp$ IntElim 20 21
23. $\sim(\sim A \& A)$ ImpInt 22 19 Qed
24. $\sim(A \vee \sim A)$ Hyp
25. A Hyp
26. $A \vee \sim A$ OrIntL 25
27. $\perp$ ImpElim 26 24
28. $\sim A$ ImpInt 27 25
29. $\sim A$ Hyp
30. $A \vee \sim A$ OrIntR 29
31. $\perp$ ImpElim 30 24
32. $\sim\sim A$ ImpInt 31 29
33. $\sim\sim A \supset A$ PolySub X A
34. A ImpElim 32 33
35. $A \& \sim A$ AndInt 24 28
36. $\sim(A \vee \sim A) \supset (A \& \sim A)$ ImpInt 35 24 Qed
37. $(\sim(A \vee \sim A) \supset B) \supset (\sim B \supset \sim\sim(A \vee \sim A))$ PolySub 18 A $\sim(A \vee \sim A)$
38. $(\sim(A \vee \sim A) \supset (A \& \sim A)) \supset (\sim(A \& \sim A) \supset \sim\sim(A \vee \sim A))$ PolySub 18 B $(A \& \sim A)$
39. $\sim(A \& \sim A) \supset \sim\sim(A \vee \sim A)$ ImpElim 36 38
40. $\sim\sim(A \vee \sim A)$ ImpElim 23 39
41. $\sim\sim(A \vee \sim A) \supset (A \vee \sim A)$ Polysub 5 X $A \vee \sim A$
42. $A \vee \sim A$ ImpElim 40 41 Qed

We will also consider second-order predicate variables with fixed arity.

References

[1] D. Prawitz, Natural Deduction.

[2] J. Kelley, General Topology.

[3] A. Troelstra, Constructivism in Mathematics vol. I.