Introduction to PyLog

Clarence Protin

November 4, 2021

Abstract

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$^+$, which is work-in-progress, involves extending PyLog with parametric-style second-order features allowing us to instantiate logical validities. The language of PyLog consists of finite sets of constants, 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.
- 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:

- $FV(c) = \emptyset$
- $FV(x) = \{x\}$
- $FV(f(t_1, ..., t_n)) = \bigcup_{i=1,...,n} FV(t_i)$
- $FV(P(t_1, ..., t_n)) = \bigcup_{i=1,...,n} FV(t_i)$
- $FV(\perp) = \emptyset$
- $FV(A \vee B)$, $FV(A \ \& \ B)$ 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. We first present the system in the standard form. We assume the reader is familiar with proof trees and the concept of dependency. 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:

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

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

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

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

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

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

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*:

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

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.

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

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

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 $P(t_1, ..., t_n)$[1].

When we wish to use defined functions or constants we first introduce the defining equalities into our proof by means o 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:

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

Remark 0.1 Certain combinations of rules occur frequently and it is convenient to have dervived rules(or shortcuts) such as

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

for

[1]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.

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)



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

AxInt(number)

TheoremInt(number)

DefEqInt(number)

DefExp(ForNum, predicateName, PositionList)
```

```
DefSub(ForNum, predicateName, ArgList, PositionList)


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

Qed(ForNum)

EquivConst(ForNum,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.

```
NewAx(Formula)

NewDef(PredicateName, ArgList, Formula)

AddConstants(NameList)

AddFunction(FunctionName, Arity, PrefixBool)

NewDefEq(EquationFormula)

AddTheorem(Formula)
```

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. We also have a command Undo() which deletes the last element of the proof.

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 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

>>>
```

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. (ab) = {w: ∃x.∃y.∃z.((((x,y) ϵ a) & ((y,z) ϵ b)) & (w = (x,z)))}
15. (r) = {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)))
Full(x) <-> ∀y.((y ϵ x) -> (y ⊂ x))
Ordinal(x) <-> (Full(x) & Connects(E,x))
Integer(x) <-> (Ordinal(x) & WellOrders((E),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
===================================

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)`

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)).

References

[1] D. Prawitz, Natural Deduction.

[2] J. Kelley, General Topology.

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