Pi: an Interactive Derivation Editor for the Calculus of Partial Inductive Definitions

Lars-Henrik Eriksson

Swedish Institute of Computer Science[1]
Box 1263, S-164 28 KISTA, SWEDEN
Phone: +46 8 752 15 09
E-mail: lhe@sics.se

Abstract. Pi is a system for the interactive construction and editing of formal derivations in the calculus of finitary partial inductive definitions. This calculus can be used as a logical framework where object logics are specified, turning Pi into a derivation system for a particular object logic. Noteworthy features of Pi include: a graphic user interface where derivations are presented in tree form, direct manipulation of the derivation tree structure by selection using a mouse, and the ability to edit existing derivations by cutting and pasting as well as by changing the formulae occurring in a derivation. A simple facility for automatic theorem proving has been designed.

1. Introduction

Pi is a program for the interactive construction of formal derivations, being developed at the Swedish Institute of Computer Science. The main design goals for the Pi system are:

- To support the calculus of finitary partial inductive definitions – a formal system that can be used as a logical framework.
- To provide a highly user-friendly user interface.
- To provide functions for editing existing derivations, in order to facilitate interactive development of derivations.

The program is intended to serve the double purpose of being a vehicle for experimentation with the use of the calculus of finitary partial inductive definitions as a logical framework and as a tool for interactively proving theorems in object logics. As a tool, it has primarily been used for formal verification. Since automatic theorem proving facilities have not yet been implemented only smaller proofs have been done so far – up to a few hundred inferences – although work on substantially larger proofs is in progress.

Much of the inspiration for the design of the Pi user interface comes from the IPE (Interactive Proof Editor) system from the University of Edinburgh [6].

The author's Ph.D thesis [1] gives a thorough description of the calculus of finitary partial inductive definitions, of using it as a logical framework and of the principles of the Pi system. [2] summarises the calculus and describes its use as a logical framework.

Pi is written in SICStus Prolog and runs under X windows. It consists of about 5000 lines of Prolog code. The program is perhaps also of interest as a "production-size" Prolog program which makes virtually no use of meta-logical predicates and which uses cut only in situations where it is safe from a declarative point of view. The program is available from the author.

[1] This paper was written while the author was visiting the Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Germany.

2. The Meta Logic

The meta logic of the Pi system is the calculus of *finitary partial inductive definitions* [1,2] – a finitary variant of partial inductive definitions [3]. This calculus is a quite general logical formalism, where inference rules are parametrised by a finitary partial inductive definition (or simply a *definition*). In contrast to ordinary inductive definitions, the operator associated with a partial inductive definition may be non-monotonic [3].

Briefly, the calculus can be described as an intuitionistic sequent calculus including only connectives and inference rules for conjunction, implication, and universal quantification. The calculus also includes the usual structural inference rules – except cut. (Although cut is sometimes an admissible rule of the calculus.) A definition comprises a set of *clauses*, each of which "defines" some atomic formula in terms of an arbitrary formula of the calculus.

Additionally, there are two inference rules that depends on a particular definition. The inference rule for applying the definition to the succedent of a sequent ("definition-right") basically infers the succedent formula from its definition. The inference rule for applying the definition to the antecedent of a sequent ("definition-left"), implements the dual principle of *definitional reflection* [7].

The parametrisation of the inference rules makes the calculus suitable as a logical framework. A logic can be represented by providing a particular definition, which in a sense makes the inference rules of the calculus behave as inference rules of the represented (*object*) logic. The logics that have actually been represented so far include ordinary first-order logic, several modal logics, three-valued logic, naive set theory and the type system of the Edinburgh logical framework. To distinguish the connectives of the meta-logic from the corresponding connectives of object logics, they are not written with the usual symbols.

For representing first-order logic, the missing connectives (disjunction, negation, and existential quantification) can all be defined using a suitable finitary partial inductive definition. With that definition, the two definition inference rules will behave as the sequent calculus rules for these logical connectives. (In practise, conjunction, implication and universal quantification are defined anew as synonyms to the meta level connectives, to provide a complete set of connectives and inference rules at the object level.)

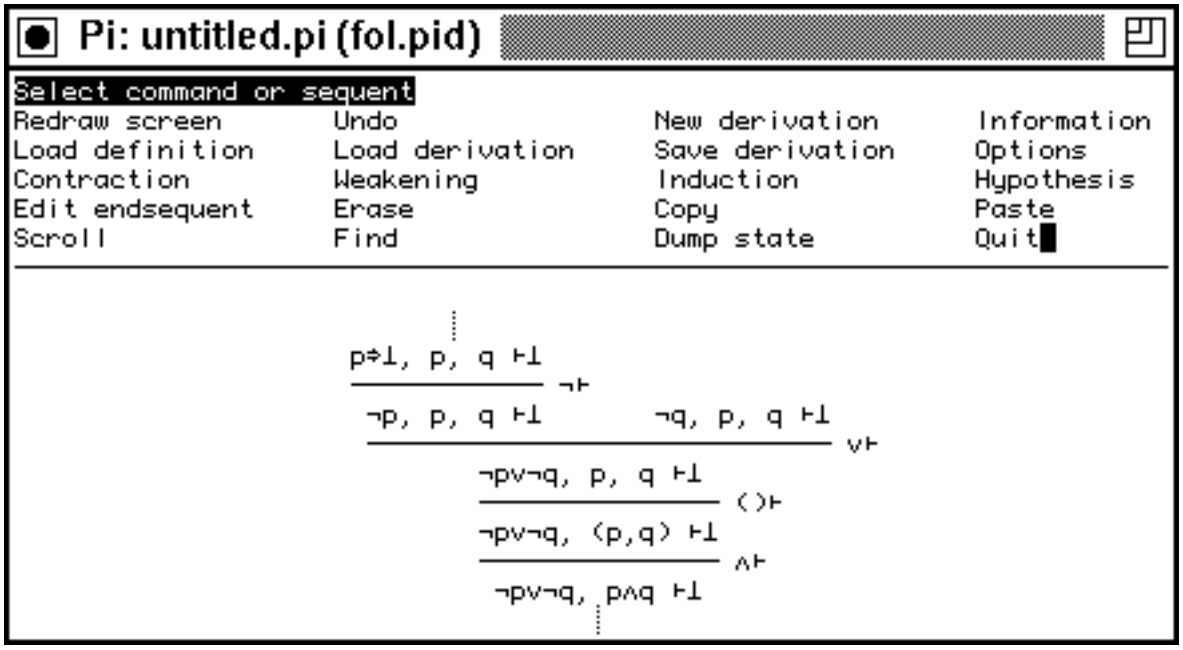
To provide a sufficiently expressive language for the representation of object logics, the atomic formulae of the calculus are terms of typed lambda calculus with a type system based on the simple theory of types, similar to that of λProlog [5]. The formulation of the definition inference rules requires full unification of higher-order terms – pre-unification is not enough. In practise, though, higher-order pattern unification [4] is usually sufficient.

3. The User Interface

To use Pi, the user must prepare a file with the particular partial inductive definition to be used. This file also includes relevant syntax and type declarations. In order to display formulae of an object logic as close as possible to that of the standard formulation of that logic, Pi provides a flexible syntax and can display several special symbols commonly used in logic and computer science. After starting the Pi session, the definition file is loaded and its contents determines the behaviour of the definition inference rules as well as the way formulae are parsed and printed. In the examples of the rest of this paper, we assume that a definition of first order logic has been loaded.

The construction of the derivation of a sequent proceeds in a top-down, goal-oriented fashion. The user specifies the sequent to be derived – the *endsequent* – using a built-in text editor. This creates a derivation tree with the endsequent as its only node. The user then requests the insertion of an inference step to derive this sequent. That inference step will be added to the derivation tree, having (in general) some premises which must also be derived. The user then adds inferences steps to derive each premise, and so on until no premises remain. The user is not bound to any specific order of deriving open premises, but can do this in any order desired.

Interaction with Pi takes place using a window, with a menu or text entry area at the top and the rest of the window displaying (part of) the derivation in progress. The following picture shows a typical situation[2] :



This window shows a derivation tree. Pi will attempt to arrange the derivation within the window in such a way as to make as much as possible of the derivation fit in the space available. In this case the full derivation tree is too large to fit on the screen, so dotted lines show where the derivation continues, above or below the visible part. By selecting one of the dotted lines with the mouse, the display is scrolled in the corresponding direction.

To present derivations entirely within the object logic, is it possible to suppress the display of those parts of a derivation dealing with the meta-logical connectives.

Commands are given by selecting items in the menu at the top using a mouse or other pointing device. Logical (non-structural) inference steps are added to the derivation tree by selecting the appropriate formula of a goal sequent. The form of the selected formula uniquely determines the inference rule to be inserted. E.g. selecting the formula $\neg q$ in the sequent $\neg q, p, q \vdash \bot$ causes an inference step for negation in the antecedent to be inserted.

In some cases, the user must choose between variants of a rule, or provide additional information – such as the formula to be substituted for the quantified variable when using the rule for universal quantification in the antecedent. Choices are presented in the menu area and selected using the mouse. Formulae are entered using the built-in text editor.

[2] Except that the size of the window is not typical, but quite small in order to fit on the page

4. Derivation Editing

A central feature of Pi is the ability to edit derivations. By "derivation editing" we mean the process of modifying an existing derivation (except by simply adding new inference steps to leaves of the derivation tree or by removing entire subderivations). There are four editing operations available to the user:

- Copying and pasting subderivations
- Changing the endsequent of the whole derivation
- Inserting an inference step into the middle of the derivation tree
- Changing an existing inference step in the middle of the derivation tree

The derivation editing functions are implemented using two basic operations: changing the derivation tree structure and deriving a sequent using the same inference steps as those of a given derivation. The latter operation can also be regarded as changing the endsequent of the given derivation and propagating the changes throughout the entire derivation.

Although conceptually simple, there are two practical difficulties with the latter operation. One difficulty is that it might be impossible to derive the new sequent using the given derivation. At some place an invalid inference step might result. Pi solves this problem by inserting an "error pseudo-inference" step at the place of the invalid inference step. It serves the function of keeping the previous subderivation so that the user can do any modifications necessary to rectify the error. In most cases Pi will notice when the user has resolved the problem and automatically remove the error step.

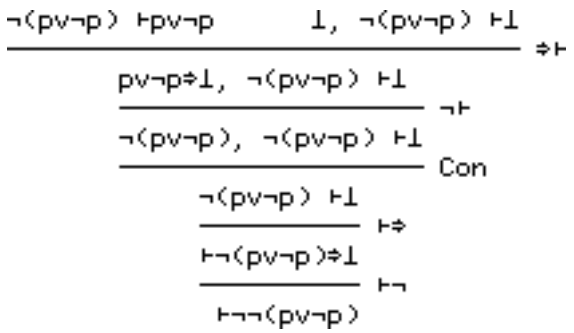
The other difficulty is to know which antecedent formulae of the new sequent that should be used with each inference step. Pi attempts to resolve this problem by heuristically matching the antecedent formulae of the new sequent with those of the old endsequent of the derivation. Presently, the heuristic will first match identical formulae, then formulae having the same type and outermost constructor. This matching is generally transparent to the user, but might sometimes not give the result the user might have expected.

Suppose the new sequent is $a, f(y,z), r \Rightarrow s \vdash c$ and the old endsequent of the derivation is $r \Rightarrow s, d, f(y,g(3)), a \Rightarrow b \vdash c$. The third antecedent formula of the former sequent will match the first antecedent formula of the latter sequent, since they are identical. The second formula of the former sequent will match the third formula of the latter sequent, since they have the same outermost function symbol.

For an example of an editing operation – the insertion of a step into a derivation – suppose that we want to derive $\vdash \neg\neg(p \vee \neg p)$ in intuitionistic logic. Forgetting that a contraction is needed, we create the following (incomplete) derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash p \vee \neg p} \qquad \dfrac{}{\bot \vdash \bot}}{p \vee \neg p \Rightarrow \bot \vdash \bot}\,\Rightarrow\vdash}{\neg(p \vee \neg p) \vdash \bot}\,\neg\vdash}{\vdash \neg(p \vee \neg p) \Rightarrow \bot}\,\vdash\Rightarrow}{\vdash \neg\neg(p \vee \neg p)}\,\vdash\neg$$

At this point we realise that a contraction step is needed at the sequent $\neg(p \vee \neg p) \vdash \bot$. By asking Pi to insert a contraction of the antecedent formula into the derivation, the derivation is changed to:



5. Automatic Theorem Proving

Although Pi is intended for interactive theorem proving, simple facilities for automatic theorem proving can be of great help to the user by automatically performing trivial derivation construction tasks. At the time of this writing, such facilities have been designed but not yet implemented.

The automatic theorem proving facilities consist of two parts. The first part is the possibility of using incomplete derivations as derived inference rules. This is by itself not automatic theorem proving, since no search is involved. However, the used of derived rules can greatly speed up the derivation construction process.

The second part is a mechanism that takes a set of derived rules and attempts to apply each in turn to a sequent to be derived. If some derived rule is applicable, it is inserted into the derivation. The process is then repeated in a depth-first fashion on the premises of the newly inserted rule. This is an automatic theorem proving process, although quite simple.

For the purposes of assisting interactive work, we expect these facilities to be sufficient. They also have the advantage that it is easy to construct the sets of derived rules.

6. References

1. Eriksson, L.-H.: Finitary Partial Inductive Definitions and General Logic, Ph.D. thesis, Department of Computer and Systems Sciences, Royal Institute of Technology, Stockholm 1993.
2. Eriksson, L.-H.: Finitary Partial Inductive Definitions as a General Logic, In: Dyckhoff, R. (ed.), Extensions to Logic Programming – ELP 93, Springer Lecture Notes in Artificial Intelligence, 1994.
3. Hallnäs, L.: Partial Inductive Definitions, Theoretical Computer Science, vol. 87, no. 1, 1991.
4. Miller, D.: A Logic Programming Language with Lambda-Abstraction, Function Variables and Simple Unification, In: Schroeder-Heister, P. (ed.), Extensions of Logic Programming, Springer Lecture Notes in Artificial Intelligence 475, 1991.
5. Nadathur, G. and Pfenning, F.: Types in Higher-Order Logic Programming, In: Pfenning, F. (ed.), Types in Logic Programming, The MIT Press, 1992.
6. Ritchie, B.: The Design and Implementation of an Interactive Proof Editor, Ph.D. thesis CST-57-88, Department of Computer Science, University of Edinburgh, 1988.
7. Schroeder-Heister, P.: Rules of Definitional Reflection, Proceedings of the Eighth Annual IEEE Symposium of Logic in Computer Science (LICS), pp. 222 – 232, 1993.