A Curry-Howard Perspective On Cyclic Proofs: Case for Support

James Brotherston (PI) and Amina Doumane (RCI)
Dept. of Computer Science, University College London

Abstract

1 Previous Research Track Record

James Brotherston is a Reader in Logic and Computation in the Dept. of Computing at University College London (UCL), where he is a member of the Programming Principles, Logic and Verification (PPLV) research group. Brotherston holds a PhD from the School of Informatics, University of Edinburgh 2006. He was a postdoctoral researcher at Imperial College London from 2006–11 before taking up a lectureship at Queen Mary University of London, moving to UCL in 2012. He has held as PI grants worth more than £1M in total to date, including an EPSRC Postdoctoral Fellowship (2008–11) and an EPSRC Career Acceleration Fellowship (2011–16). He is the author of 29 conference and journal papers, with an h-index of 17.

Brotherston's main research interests lie in mathematical logic and its application to computer science, particularly in automated program verification. He is especially well known for his pioneering work on *cyclic proof* (see e.g. [**?**, **?**, **?**]) and also for his work in *separation logic* (see e.g. [**?**, **?**]).

Amina Doumane is a Post-doc in the LIP Computer Science laboratory at Ecole Normale Supérieur (ENS) Lyon, where she is a member of the Proofs and Languages (PLUME) team, headed by Professor Russ Harmer. Before joining the PLUME team, Doumane has defended her PhD at University Paris Diderot, under the Supervision of Pierre-Louis Curien, David Baelde and Alexis Saurin.

The main thematic axis of Doumane's research is proof theory and its relations with computer science. More precisely, she is interested in the proof theory of logics with fixed points and its applications to programming languages and verification. Her work on the normalization of circular proofs [**?**] lays the theoretical foundations to develop typed programming languages based on circular proofs. She also gave a new constructive proof of completeness for the linear-time mu-calculus [**?**, **?**]. The underlying algorithm to this proof can be implemented to produce certificates supporting the decision of model-checkers. This last result won the Kleene award for the best student paper at LICS 2017.

Relevant previous work. The investigators have contributed significantly to ...

Host organisation. The UCL Dept. of Computer Science is well known as a leading centre for computer science research in the UK, and was ranked first place of 89 universities for Computer Science in the 2014 REF. The PPLV group, headed by Prof. David Pym, consists of six full-time faculty members — Pym, Brotherston, professors Alexandra Silva and Robin Hirsch, and lecturers Ilya Sergey and Fabio Zanasi — plus a substantial number of postdocs, PhD students and part-time faculty. The group enjoys close links with Facebook (through Peter O'Hearn, part-time professor and Engineering Manager at Facebook) and Amazon (through Byron Cook, part-time professor and Head of AWS Security Automated Reasoning Group at Amazon) as well as the recently founded Turing Institute (through David Pym, ATI University Liaison Director for UCL) and numerous collaborative links with researchers at other universities.

1 Proposed Research and its Context

1.1 Proposition of a high level view of the project

- Logic has been succesfully applied in many domains of computer science. In particular in the domains of **verification** and **programming laguages**:
  - In verification: formulas-as-specifications, proofs-as-certificates.
  - In programming languages: formulas-as-types, proofs-as-programs
- Fixed points logics are crucial in both verification and programming languages:
  - In verification: we can express a wide range of properties using fixed points.
  - In PL: program with (co-)inductive data types
- Two main families of proof systems for logics with fixed points
  - Finite proofs with explicite (co-)induction rules. Widely used but with many drawbacks.
  - Circular proofs. Not very well-explored yet very promising.
- **Our goal** is to develop:
  - Theorem provers (allowing certificate production)
  - Typed programming languages

  based on circular proofs.

1.2 Background

Good methods for reasoning with general inductive predicates have the potential to greatly boost the automation and coverage of such automatic verification tools.

However, the difficulty in reasoning with general inductive definitions — a fundamental activity in both mathematics and computer science — is well known, perhaps even notorious. Automating the process of mathematical reasoning using induction and recursion principles, generally known as *inductive theorem proving*, has been a topic of active interest in computer science and AI research since at least the late 1970s, with the development of specialised theorem provers such as NqThm [**?**]. Since its inception, the default approach to automated inductive theorem proving has been *explicit induction*: the direct application of proof rules or axioms capturing valid induction principles for the structures under consideration. The best known explicit-induction provers in the last 20 years include ACL2 [**?**], Inka [**?**] and Oyster-Clam [**?**], alongside more recent examples such as IsaPlanner [**?**] and Zeno [**?**]. However, reasoning using explicit induction rules is known to be extremely difficult (for an overview see [**?**]). One of the biggest problems is that in an explicit induction proof, the structure(s) over which induction is to take place, the form of the induction schema and the induction hypotheses must all be chosen at the *beginning* of the proof, before it is apparent which choices are likely to be fruitful.

An alternative to proof by explicit induction is provided by instead working in the *infinite descent* style as proposed formally by Fermat [**?**] in which, roughly speaking, the use of explicit induction is eschewed in favour of a combination of simple *case analysis* and a post-facto search for justification of the use of hypotheses by finding well-founded orderings. Wirth [**?**] describes an imagined mathematician working in this way as follows:

> *He starts with the conjecture and simplifies it by case analysis. When he realizes that the current goal becomes similar to an instance of the conjecture, he applies the instantiated conjecture just like a lemma, but keeps in mind that he has actually applied an induction hypothesis. Finally, he searches for some well-founded ordering in which all the instances of the conjecture he has applied as induction hypotheses are smaller than the original conjecture.*

This approach has the advantage for proof search that the crucial choices of induction structures, schema and hypotheses are *delayed until as late as possible in the proof.*

Cyclic proof systems essentially formalise this concept: they drop explicit induction rules in favour of much simpler case analysis rules, but compen-

sate by allowing for more complexity in the proof structure. Specifically, a cyclic proof is a derivation tree whose leaves may be "back-linked" to another matching node in the proof (see Figure **??**), subject to a (decidable) global soundness condition ensuring that all infinite paths in this structure correspond to sound arguments by infinite descent.

Cyclic proof systems seem to have first arisen in computer science as tableaux for the propositional modal μ-calculus [**?**]. Since then, cyclic proof systems have been proposed for a number of applications, including: first-order μ-calculus [**?**]; verifying properties of concurrent processes [**?**]; first-order logic with inductive definitions [**?**], bunched logic [**?**]; and termination of pointer programs [**?**]. For some time, the development of these formal proof systems was not matched by the development of automated tools implementing them, most likely owing to the technical difficulties of dealing with cyclic proof structures subject to a global soundness condition. However, in the last few years a small number of tools based on cyclic proof have appeared, including the interactive QuodLibet prover [**?**], and provers geared to limited fragments of separation logic [**?**, **?**]. Most recently, a prototype theorem prover, Cyclist, has been developed that implements both the generic notion of cyclic proof due to Brotherston and its instantiation to many of the aforementioned concrete settings, including the abduction of inductive predicates for program termination [**?**, **?**]. This prover exhibits promising automated performance on examples that ordinarily require complicated induction schemes, which are problematic for traditional explicit-induction theorem provers.

The Curry-Howard correspondence. Traditionally, proofs are a tool used to establish the validity of mathematical statements. Proofs have been generally compared at an aestethic/concision level, but were not considered as a proper object of study. The focus was in provability not in the proofs themselves. This situation have changed in the 60's, where we descovered a deep link between proofs and programs. This link is known as the *Curry-Howard Correspondence*. [Recall this correspondence.]

Over time, the meaning of the Curry-Howard correspondence has evolved from the simple observation of an isomorphism between an existing proof system and an existing programming language, to something more "productive". Indeed, one can either start from a typed programming language and extract its isomorphic proof system which may have an interesting logical meaning. Conversely, one could start from a known proof system and try to understand its computational content in order to design its corresponding programming language, this is for instance how a number of calculi were born [**?**, **?**, **?**].

Curry-Howard was also productive in renewing questions related to the semantics of logic, putting an emphasis on the semantics of proofs, in contrast to tranditional thruth or provability semantics.

Fixed points logics are those logics containing two special connectives: the connective μ and its dual ν. A formula of the form $\mu X.F$ (resp. $\nu X.F$) can be understood as the "least (resp. greatest) fixed points of the operator $X \mapsto F(X)$".

Least and greatest fixed points allow to treat in a direct, generic and intuitive way data (natural numbers, lists, etc) and co-data (streams, infinite trees, etc). The least fixed point operator μ allows to define finite data types such as natural numbers, lists, finite trees, etc. Dually, the greatest fixed point operator allow to define infinite data types (called co-data), such as streams, infinite trees, etc.

Proof systems for logics with fixed points. There are two main families of proof systems with fixed points: finitary proof systems with explicit rules of (co)induction and infinitary proof systems.

- Finitary proof systems. In these proof systems, the induction principle is reflected by an explicit rule, called **Park's rule**. Let us recall that the induction principle is based on the characterization of the least fixed point of an operator $X \mapsto F(X)$, $\mu X.F$, as its least pre-fixed point. This means that:

  i) $\mu X.F$ is a pre-fixed point, *i.e.*, $F(\mu X.F) \leq \mu X.F$.

  ii) $\mu X.F$ is smaller than any other pre-fixed point S of F:

  $$F(S) \leq S \Rightarrow \mu X.F \leq S.$$

  These two points yield respectively the following two rules for the μ connective:

  $$\frac{\Delta \vdash F[\mu X.F/X], \Gamma}{\Delta \vdash \mu X.F, \Gamma}\ (\mu_r) \qquad \frac{F[S/X] \vdash S}{\mu X.F \vdash S}\ (\mu_l)$$

Dually, the coinduction principle, based on the characterization of the greatest fixed point of an operator as its greatest post-fixed point, yields the following two rules:

$$\frac{S \vdash F[S/X]}{S \vdash \nu X.F}\ (\nu_{\mathsf{r}}) \qquad \frac{\Delta, F[\nu X.F/X] \vdash \Gamma}{\Delta, \nu X.F \vdash \Gamma}\ (\nu_{\mathsf{l}})$$

The problem of these rules is that, in general, the cut rule is not admissible in the proof systems using them. To recover cut-elimination, the following variant of the rules (μ_{l}) and (ν_{r}), containing a "hidden cut", are usually used instead:

$$\frac{\Gamma \vdash S, \Delta \qquad S \vdash F[S/X]}{\Gamma \vdash \nu X.F, \Delta}\ (\nu)$$

$$\frac{F[S/X] \vdash S \qquad \Gamma, S \vdash \Delta}{\Gamma, \mu X.F \vdash \Delta}\ (\mu_{\mathsf{r}})$$

The fact that the cut rule is unavoidable (either in an explicit or a hidden way) means that proof systems with explicit rules of induction are fundamentally not suited to proof-search, and that there is very little we can do to fix that. An other drawback of proofs using explicit induction is that their computational meaning is usually not explicit: it is hard to tell what such proofs compute, when two proofs compute the same function, etc.

- Infinitary proof systems. These systems are obtained by using, instead of the Park's rules (ν_{r}) and (μ_{l}), the following unfolding rules:

$$\frac{\Gamma \vdash F[\nu X.F/X], \Delta}{\Gamma \vdash \nu X.F, \Delta}\ (\nu_{\mathsf{r}}) \qquad \frac{\Gamma, F[\mu X.F] \vdash \Delta}{\Gamma, \mu X.F \vdash \Delta}\ (\mu_{\mathsf{l}})$$

and allowing infinite derivation trees, called **pre-proofs**. Clearly, these rules do not reflect the difference in nature between μ and ν. More importantly, these pre-proofs are unsound, since one can derive the empty sequent as follows:

$$\dfrac{\dfrac{\dfrac{\vdots}{\vdash \mu X.X}\ (\mu_{\mathsf{r}})}{\vdash \mu X.X}\ (\mu_{\mathsf{r}}) \qquad \dfrac{\dfrac{\vdots}{\mu X.X \vdash}\ (\mu_{\mathsf{l}})}{\mu X.X \vdash}\ (\mu_{\mathsf{l}})}{\vdash}\ (\mathsf{Cut})$$

To get a proof system which is sound, we declare a pre-proof to be a proof if and only if it satisfies the **validity condition**. This condition says, roughly speaking, that in every infinite branch there should be either a least fixed point unfolded infinitely often in the left or a greatest fixed unfolded infinitely often in the right.

There is a natural restriction to infinitary proofs given by regular proofs, that is, proofs that have only finitely many sub-trees. These proofs are called **circular proofs**, since they can be represented as finite trees with loops. This restricted **cyclic proof system** is more suited for a computer science use, as its proofs can be finitely represented and manipulated.

Infinitary and circular proof systems have existed for a long time, but in the shadows, and more as technical tools than as proper proof systems. Indeed, their infinitary nature makes them suitable intermediary objects between the syntax and the semantics, so that we usually find them in completeness proofs. This is the case, for instance, with the μ-calculus **refutations** of Niwinski and Walukiewicz [**?**], which are used as an intermediary proof system in Walukiewicz's proof of completeness of the μ-calculus with respect to Kozen's axiomatization [**?**].

In the last decades, infinitary proof systems have started to come out from the shadows and have begun to be considered as proper proofs (Dam and Gurov [**?**], Dax *et al.* [**?**]. Santocanale [**?**]. Brotherston [], QuodLibet [**?**], ...)

Despite their compelling interest, few proof-theoretical investigations have been done about infinitary proof theory (cut-elimination, focalization, ...).

1.3 Research hypothesis and objectives

1.4 National importance

1.5 Programme and methodology

1.5.1 Task A: Circular functional programs

- **Long term objective:** Design typed functional programming languages for (co)-inductive data-types based on circular type systems.

The use of circular proofs as type systems represents a promising new method of writing (co)inductive programs, at least from the three following perspectives:

1. The programs are easier to write, more user-friendly. Indeed, there are no invariants to guess.
2. It is known that every finitary proof can be transformed into a circular one, but the converse is still an open question (cf Task B). Circular proofs may then contain "more proofs" and using circular type systems may allow us to write more programs.
3. Another interesting feature of the infinitary proofs is that their computational meaning is more explicit compared to finitary proof systems. [Add an example in the background section to argument that?]

- **Short/Mid-term objective:** Before realizing this long term objective, one needs to ensure that the circular type sytems enjoy the usual fundamental properties, normalization for instance.

  A step in this direction has been done by the second author and *al.*, who showed cut-elimination for circular proofs in the setting of sequent calculus []. It has been shown initially for multiplicative addivite linear-logic MALL, but have been extended to other logics (LL, LK, LJ).

  The translation between sequent calculus and natural deduction is usually immediate in the finitary case. In the infinitary and the circular setting, this is not the case anymore.

  Since circular typing systems are usually in correspondence with the formalism of natural deduction rather than with sequent calculus one, showing their normalization is not a corrolary of the second author's result. Thus we have for this direction two short/mid-term objectives:

  1. Define properly a validity condition for natural deduction circular proofs.
  2. Show normalization.

[We started working on that with David Baelde, Alexis Saurin and Guilhem Jaber, can we explicitely cite them as collaborators for this work direction?]

1.5.2 Task B: The expressive power of circular proofs

The question of relating the finitary proof systems with explicit (co)induction rules and the circular proof systems is a hard question.

It has been already set as a conjecture in similar settings, for instance by Brotherston and Simpson [**?**] in the framework of classical first-order logic with definitions.

We can ask this question at two levels: at the level of **provability** and at the level of **computational power**. Before discussing these two work directions, let us set up some notations.

Notation 1. Let S be a proof system for a logic *without* fixed points. We denote by $\mu\mathsf{S}$ the finitary proof system obtained from S by adding the explicite (co)induction rules.

We denote by $\mu\mathsf{S}^\infty$ the infinitary proof system obtained from S by adding the unfolding rules for the fixed points and considering as proofs the infinite derivations satisfying the validity condition.

We denote by $\mu\mathsf{S}^\omega$ the circular proof system obtained by restricting the proofs of $\mu\mathsf{S}^\infty$ to the regular ones.

For this task, we have two main research directions:

- **Relating the finitary and the circular proof systems at the level of provability:** For a given proof system S, does the proof systems $\mu\mathsf{S}$ and $\mu\mathsf{S}^\omega$ prove the same theorems?

  In [**?**], this question has been partially answered by showing that every provable sequent in $\mu\mathsf{S}$ is also provable in $\mu\mathsf{S}^\omega$. The converse in more difficult, and [**?**] provide only a sufficient condition on the proof system $\mu\mathsf{S}$ and $\mu\mathsf{S}^\omega$ (the invariant property) and a condition on the circular proofs to be translated (the translatability criterion) which guarantees that they can be indeed transformed into finitary proofs.

  If the answer to this problem happens to be positive, we see two ways to proceed:

– Either by finding a semantics of formulas, common to both $\mu\mathsf{S}$ and $\mu\mathsf{S}^\omega$. The idea is to show soundness for $\mu\mathsf{S}^\omega$ (that is, if a formula is provable in $\mu\mathsf{S}$ then it is valid *w.r.t.* this semantics) and completeness for $\mu\mathsf{S}$ (that is, if a formula is valid then it is provable in $\mu\mathsf{S}^\omega$). This is the approach taken by Brotherstone and Simpson [**?**].

– Or by using combinatorial techniques, that is by showing a translation procedure that transforms $\mu\mathsf{S}^\omega$ proofs into $\mu\mathsf{S}$ ones. This is the approach adopted in [**?**]. It has the advantage of being constructive by essence.

Recently, the conjecture of Brotherston and Simpson has been been answered by Berard and Tatsuta [**?**] and by Alex Simpson [**?**].

- **Relating the finitary and the circular proof systems at the level of computational power:** For a given proof system S, do the proof systems $\mu\mathsf{S}$ and $\mu\mathsf{S}^\omega$ denote the same set of programs.

  If the answer happens to be positive, a way to proceed is by using **proof semantics**. For that, we have to find a domain of interpretation for proofs, common to both $\mu\mathsf{S}$ and $\mu\mathsf{S}^\omega$. If we denote by $[\![\pi]\!]$, the interpretation of a $\mu\mathsf{S}$ or a $\mu\mathsf{S}^\omega$ proof π, this problem can be reformulated as follows: For every proof Π of $\mu\mathsf{S}^\omega$ (resp. $\mu\mathsf{S}$), is there a $\mu\mathsf{S}$ (resp. $\mu\mathsf{S}^\omega$) proof π such that $[\![\Pi]\!] = [\![\pi]\!]$?

  An example of such proof semantics is computational ludics, which was used by the second author to compare the $\mu\mathsf{MALLP}$ and $\mu\mathsf{MALLP}^\omega$, the finitary and the circular extensions of polarized linear logic MALLP.

(C): *Cyclic proof for mixed induction and coinduction.*

Having previously developed cyclic proof systems for (program) logics with inductive constructs, it is of clear interest to to extend our methodology to logics featuring coinductive structures, e.g., bisimilarity relations as widely used in concurrency theory. Our first step will be to formulate and analyse a cyclic proof system for coinduction. Since coinductive proof often takes a circular form based on a simple guardedness condition, it would also be interesting to compare our proof machinery with these established forms. Having already considered cyclic proof for induction, we believe dualising the concepts to coinduction should be fairly straightforward, so the obvious next step is to consider logics featuring mixed inductive and coinductive structures. The more complex soundness condition likely to be needed in the presence of mixed induction and coinduction, taking into account the possible nesting of least and greatest fixed points, is suggested by the work of Niwinski and Walukiewicz on refutations in the μ-calculus [**?**]. Having formulated proof systems for mixed induction and coinduction, the secondary research goals are analogous to those previously achieved for the preceding systems for induction: the translation of proofs featuring explicit uses of induction and coinduction into cyclic proofs; and appropriate completeness results for the associated infinitary systems. (We suspect that completeness will be non-trivial to establish, since in the case of Niwinski and Walukiewicz [**?**], the set-theoretic property of Borel determinacy is required.)

(D): *Cyclic proof in other program logics.*

Finally, it is also of interest to consider the potential for cyclic proof in the setting of other program logics than the frameworks considered in the strands above. For example, one could consider frameworks in which the syntax of the programming language under consideration is separated from an overlying logic, e.g. Plotkin's PCF language [**?**] with LCF as the logic, or the pure-functional fragment of ML with an appropriate logic. In such settings, cyclicity together with an appropriate guardedness condition would presumably replace or complement the usual structural induction principles for the inductive datatypes of the language.

One could also investigate frameworks in which the programming language and logic are combined in a dependent type theory. Cyclic proof within such a framework would very likely take the form of a construct for general recursion with appropriate restrictions on its application, which would replace the usual fixed recursion constructors for inductive types within the theory. Correctness would be ensured by proving a restricted normalization result sufficient to ensure logical consistency, and also by proving that closed terms of inductive type reduce to canonical form (i.e. the type theory correctly

models datatypes).