# Boosting Automated Verification Using Cyclic Proof: Case for Support

J. Brotherston (PI), B. Cook (CI), N. Gorogiannis (RCI)

## Abstract

Automatic verification tools based on *separation logic* have recently enabled the verification of code bases that scale into the millions of lines. Such analyses rely on the use of *inductive predicates* to describe data structures held in memory. However, such predicates are currently hard-coded into the analysis, which means that the analysis must fail when encountering an unknown data structure. This results in reduced program coverage and increased rates of false negatives. Thus, methods for reasoning with *general* inductively defined predicates could greatly enhance the state of the art.

*Cyclic proof*, in essence, implements reasoning by *infinite descent* à la Fermat for general inductive definitions. In contrast to traditional proofs by explicit induction, which force the prover to select the induction schema and hypotheses at the very beginning of a proof, cyclic proof allows these difficult decisions to be postponed until exploration of the proof search space makes suitable choices more evident. This makes cyclic proof an attractive method for automatic proof search.

The main contention of this proposal is that cyclic proof techniques can add inductive reasoning capability to the many components of a program analysis (theorem proving, abduction, frame-inference, abstraction) and thus can significantly extend the reach of current verification methods.

## 1 Previous Research Track Record

**James Brotherston** is a Lecturer and EPSRC Career Acceleration Fellow in the Dept. of Computing at University College London, where he is a member of the Programming Principles, Logic and Verification (PPLV) research group headed by Professor Peter O'Hearn. Brotherston holds a PhD from the School of Informatics, University of Edinburgh 2006. He held an RA position (2006–08) followed by an EPSRC Postdoctoral Fellowship (2008–11) at Imperial College London before joining O'Hearn's group at Queen Mary in Dec'11, which migrated to UCL in early 2012. He is the author of 17 published conference and journal papers.

Brotherston's main research interests lie in mathematical logic, especially proof theory, and its application to program verification and other problems in computer science. For example, his work with Kanovich [BK10] developed undecidability results for separation logic demonstrating concrete limits on what can be achieved with automated verification tools. Brotherston has also been strongly influential in the development of *cyclic proof* for logics with inductive definitions, providing the necessary foundations in his PhD work under Alex Simpson [Bro05, BS07] and later working on extensions to other settings, notably Hoare-style systems for program termination in separation logic, with Bornat and Calcagno [BBC08]. Very recently, Brotherston and Gorogiannis have explored the exciting possibility of using such a cyclic proof framework to infer or *abduce* the correct inductive predicates needed for a successful termination analysis [BG12]. Brotherston has also worked on prototype automated theorem proving tools employing cyclic proof architectures, most notably the CYCLIST theorem prover developed with Gorogiannis and Petersen [BGP12].

**Byron Cook** is Professor of Computer Science in the PPLV group at UCL and a Principal Researcher at Microsoft Research Cambridge. Cook's research interests include formal methods, logic and programming languages. He also has interest and expertise in the areas of hardware and systems, based on his previous employment in Intel's microprocessor design division and in Microsoft's Windows kernel team. To date Cook has worked actively in the areas of: functional programming; hardware modelling and design; SAT-solving; symbolic model checking for finite-state systems; decision procedures; automatic program verification and analysis; and the analysis of biological systems.

Cook's research in automatic program verification [CK11, GCPV09, CKV11] has gained significant attention (e.g. a substantial publication record, numerous keynote speaker invitations, and press hits in Scientific American, Science, Vogue, Financial Times, Economist, and Wired). He is particularly well known for his work on automatic methods for proving program termination and the associated Terminator prover [CPR06]. This work represents a breakthrough, challenging the prevailing opinion in computer science that automatic termination proving is impossible. Byron is also well known for his contributions to SLAM, which is often credited for the recent revival of automatic program verification research, and SLAyer, which is the best-in-class program verification tool supporting programs that make use of the heap [BCI11].

**Nikos Gorogiannis** is a Research Assistant in the PPLV group at UCL, presently employed on the "Resource Reasoning" EPSRC project held by jointly by O'Hearn at UCL and Gardner at Imperial. He took up his current position at Queen Mary in 2010 and moved to UCL along with O'Hearn's group in 2012. Previously he was an RA in the Dept. of Computer Science at UCL (2007–2009) where he worked with Prof. Anthony Hunter on knowledge representation and AI. Gorogiannis holds a PhD in Computer Science from the University of Birmingham (2006), supervised by Prof. Mark Ryan. He is the author of 11 conference and journal papers.

The main thematic axis of Gorogiannis' research is knowledge transfer between and within the areas of automated reasoning, AI and verification, with an emphasis placed on applications and implementations. Recently he has worked with Brotherston on the Cyclist cyclic theorem prover [BGP12], of which he is the main designer and implementor, and on the automatic abduction of inductively defined termination preconditions [BG12]. His previous work on verification includes complexity results on abducing program specifications in separation logic [GKO11]; formalising the relationship between specification updates and property satisfaction [GR07]; and using model checking for belief revision and merging [GH08], including an application for circuit fault-diagnosis. His work on AI includes the exploration of the relationship between classical logic and argumentation [GH11]; establishing the complexity of argumentation games [GH10] and implementing algorithms for belief merging [GH08].

**Host organisation.** The project is to take place within the PPLV group in the Dept. of Computer Science at UCL. The department is well known as a leading centre for computer science research in the UK, and its recent investment in the creation of the new PPLV group highlights its commitment to research in theoretical areas of computer science and their applications, especially verification.

The PPLV group currently consists of Profs. Cook and O'Hearn, three lecturers including Brotherston and Juan Navarro Perez (an expert in automatic theorem proving), three RAs including Gorogiannis, and a small number of PhD students. Having been established only recently (summer 2012), the group is now actively seeking to expand its roster of RAs and PhD students. The group also enjoys close links with researchers at Imperial (Gardner, Maffeis), Queen Mary (Distefano, Robinson), Sussex (Reus) and Microsoft Research Cambridge (Berdine, Parkinson). Regular, informal research meetings involving researchers from all these institutions help to foster a vibrant and lively atmosphere, and encourage the exchange of ideas.

## Selected publications by the team

[Bro05] Brotherston, J.: Cyclic proofs for first-order logic with inductive definitions. In: Proceedings of TABLEAUX-14. pp. 78–92. (2005)

[BBC08] Brotherston, J., Bornat, R., Calcagno, C.: Cyclic proofs of program termination in separation logic. In: Proceedings of POPL-35. pp. 101–112. (2008)

[BCI11] Berdine, J., Cook, B., Ishtiaq, S.: SLAyer: memory safety for systems-level code. In: Proceedings of CAV-23. pp. 178–183. (2011)

[BG12] Brotherston, J., Gorogiannis, N.: Cyclic abduction of inductive termination preconditions. Submitted. (2012)

[BGP12] Brotherston, J., Gorogiannis, N., Petersen, R.L.: A generic cyclic theorem prover. In: Proceedings of APLAS-10, to appear. (2012)

[BK10] Brotherston, J., Kanovich, M.: Undecidability of propositional separation logic and its neighbours. In: Proceedings of LICS-25. pp. 137–146. (2010)

[BS07] Brotherston, J., Simpson, A.: Complete sequent calculi for induction and infinite descent. In: Proceedings of LICS-22. pp. 51–60. (2007)

[CK11] Cook, B., Koskinen, E.: Making prophecies with decision predicates. In: Proceedings of POPL-38. pp. 399–410. (2011)

[CKV11] Cook, B., Koskinen, E., Vardi, M.: Temporal property verification as a program analysis task. In: Proceedings of CAV-23. pp. 333–348. (2011)

[CPR06] Cook, B., Podelski, A., Rybalchenko, A.: Termination proofs for systems code. In: Proceedings of PLDI. pp. 415–426. (2006)

[GH08] Gorogiannis, N., Hunter, A.: Implementing semantic merging operators using binary decision diagrams. Int. J. Approx. Reasoning 49(1), pp. 234–251. (2008)

[GH10] Hirsch, R., Gorogiannis, N.: The complexity of the warranted formula problem in propositional argumentation. J. Log. Comp. 20(2), pp. 481–499. (2010)

[GKO11] Gorogiannis, N., Kanovich, M., O'Hearn, P.W.: The complexity of abduction for separated heap abstractions. In: Proceedings of SAS-18. pp. 25–42. (2011)

[GR07] Gorogiannis, N., Ryan, M.: Minimal refinements of specifications in modal and temporal logics. Formal Aspects of Computing 19(4), pp. 417–444. (2007)

[GH11] Gorogiannis, N., Hunter, A.: Instantiating Abstract Argumentation with Classical Logic Arguments: Postulates and Properties. Artificial Intelligence Journal 175(9–10), 1479–1497 (June 2011)

[GCPV09] Gotsman, A., Cook, B., Parkinson, M., Vafeiadis, V.: Proving that non-blocking algorithms don't block. In: Proceedings of POPL-36. pp. 16–28. ACM (2009)

## 2: Proposed Research and its Context

**2.1 Background.** In the last decade, automatic verification for heap-manipulating programs based on separation logic, has made great strides, allowing the verification of large industrial code bases where only programs in the tens or hundreds of lines were previously verifiable. Crucially, separation logic allows for truly *compositional* program analysis, where the result of a large interprocedural analysis is determined by the results of the analysis on the individual procedures. The major automated tools based on separation logic to date include SMALLFOOT [3], SPACEINVADER [11] and ABDUCTOR [9]; these have been applied to increasingly large code bases, with a constantly decreasing annotation burden. However, one significant limitation of these automatic analyses is the use of fixed, hard-coded inductive predicates to describe heap data structures manipulated by programs, such as lists or trees. This means the analysis must fail or ask for assistance when it encounters data structures of a kind not described by these hard-coded predicates. Thus, 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 [4]. 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 [15], INKA [1] and OYSTER-CLAM [8], alongside more recent examples such as IsaPlanner [13] and Zeno [21]. However, reasoning using explicit induction rules is known to be extremely difficult (for an overview see [7]). 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 [17] 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 [23] 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 compensate 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 1), 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 $\mu$-calculus [22]. Since then, cyclic proof systems have been proposed for a number of applications, including: first-order $\mu$-calculus [10]; verifying properties of concurrent processes [20]; first-order logic with inductive definitions [Bro05], bunched logic [5]; and termination of pointer programs [BBC08]. For some time, the development of these formal proof systems was not matched by the development of automated tools implementing

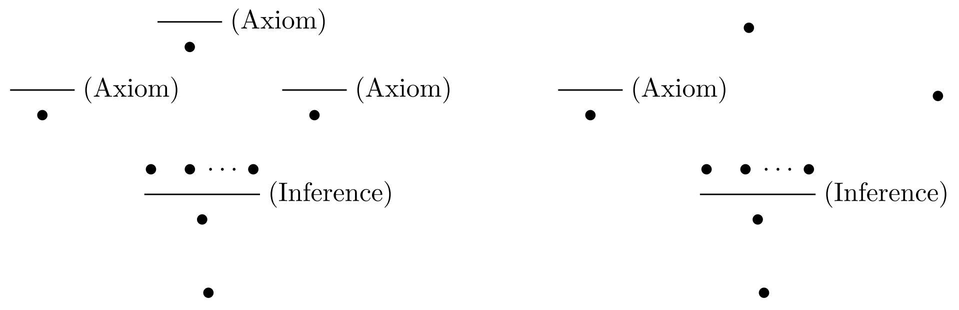

Figure 1: Left: a typical proof structured as a finite tree, with the parent-child relation between nodes (●) given by a set of inference rules. Right: a typical *cyclic pre-proof*, structured as a tree proof with "back-links" between nodes (shown as arrows).

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 [2], and provers geared to limited fragments of separation logic [18, 6]. 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 [BGP12, BG12]. This prover exhibits promising automated performance on examples that ordinarily require complicated induction schemes, which are problematic for traditional explicit-induction theorem provers.

**2.2 Research hypothesis and objectives.** Based upon the research into cyclic proof techniques to date on the one hand [BBC08, BGP12, BG12], and on interprocedural separation logic program analysis on the other [9, 11, 12, 24], we advance the following hypothesis in this proposal:

*firstly, that by integrating cyclic proof together with such powerful features of mature explicit-induction theorem provers as lemma application and advanced generalisation, a greater degree of automation in inductive proof search is likely to be achievable than by explicit induction alone; and, secondly, cyclic proof techniques can add inductive reasoning capability to the many components of an interprocedural separation logic program analysis (theorem proving, abduction, frame inference, abstraction) and thus significantly extend the coverage and automation of current verification methods.*

Accordingly, the main objective of this project is to realise the potential of cyclic proof for program analysis, and for theorem proving activities in support of such analyses. This will be achieved by building cyclic proof frameworks for the various components of a separation logic program analysis that extend and improve upon those already developed, while simultaneously focusing on the automatic proof search capability of a software tool which implements these frameworks. Then, we propose to integrate the individual cyclic provers into an interprocedural program analysis employing an overarching framework similar to that employed by current state-of-the-art analysers.

*Timeliness.* We note that there is a significant technical entry barrier to any type of mechanised theorem proving based upon cyclic proof. First, due to the requirement to form back-links in the proof, the prover must take a global view of proofs, rather than one localised to the current subgoal as in most conventional theorem provers. Second, one must implement the $\omega$-regular global soundness condition on pre-proofs qualifying them as *bona fide* proofs, which requires a model checker or similar technology. This rather daunting technical barrier goes some way to explaining the relative dearth of automatic cyclic theorem provers despite the theoretical foundations of such provers having been relatively well established for several years. However, we recently overcame these difficulties with the development of Cyclist [BGP12, BG12], which is, we believe, the first general automatic cyclic theorem prover in existence. We believe that now is thus the ideal time to capitalise on this significant implementation effort, and bring cyclic proof techniques into the mainstream of automated theorem proving.

**2.3 National importance.** This proposal is about using cyclic proof to develop new program analyses and to boost the automation of existing ones. Thus it falls within EPSRC's "Verification and Correctness" research area, which has been identified as an investment priority. Obtaining mathematical assurances that computer programs do not go wrong, e.g. by failing to terminate (the "endless hourglass") or by running out of memory, is valuable to society at large for programs in everyday use, such as device drivers; and it is quite literally of life-or-death importance in the case of safety-critical software. For example, there was a well known case in the 1980s in which Therac-25 radiation therapy machines administered massive radiation overdoses to patients due in part to a software bug [16].

Program verification is one of the areas of computer science of which the UK has traditionally been acknowledged as a centre of world-class research, and there is currently a concentration of expertise in this area in the south of England; we mention in particular the groups at Imperial, UCL, Queen Mary, Oxford and Cambridge, as well as the Microsoft research laboratory in Cambridge. Inductive theorem proving was a similarly important research area in the UK in the 1980s and 90s, with groups in Edinburgh and Cambridge being particularly active, but appears to be have been slightly on the wane since then, despite its key importance in program verification. We believe that the present proposal will help to strengthen the UK's pre-eminent position in program verification research, and provide a much-needed shot in the arm (both intellectually and in terms of resources) to research into automated inductive theorem proving.

**2.4 Programme and methodology.** This is a proposal to employ Gorogiannis and a second RA to carry out the research programme outlined below, under the supervision of Brotherston and Cook. We have organised the programme into five main themes, each described below.

***Theme A: Inductive entailments.*** The main focus of this theme is on automatically establishing entailments between formulas of separation logic with inductive predicates, as is needed in, e.g., the large-scale shape analyses employed by the Space Invader [24] and SLAyer [BCI11] tools at procedure call sites. Interestingly, we know of no explicit-induction theorem provers for separation logic in the literature, perhaps because the induction hypotheses needed even for simple proofs require the use of the multiplicative implication $-\!\!*$ [5], which is not included in the fragment of separation logic used by most existing verification tools.

At the time of writing, the Cyclist prover and previous theorem provers in the literature [6, 18] perform reasonably well on relatively simple examples, but fail to prove more complex examples, such as the equivalence of two definitions of a linked list, one where pointers are added from the left, and the other from the right. Of course, there can be no general solution to such problems for well known computability reasons, but the history of inductive theorem proving suggests some heuristic approaches that are likely to be of assistance. We believe that the two most important such approaches are: (a) the automatic *generalisation* of conjectures from failed proof attempts; and (b) the application of *lemmas* harvested from a library of previously proven theorems (perhaps discovered using machine learning algorithms). We also expect that performance might be further enhanced by moving from soundness checking based on Büchi automata to checking based on graph-based conditions which are often simpler in practice [14].

A feature of great importance for separation logic entailment provers used by program analyses is *frame inference*. This ability allows a prover to establish that the current symbolic state amounts to the precondition of the next executable command plus some left-over resource (or *frame*), which is essential in allowing the successful application of the *frame rule* of separation logic [19]. Frame inference has been incorporated into existing tools like Space Invader [24] but only for a fixed set of inductive definitions. Performing cyclic proof for entailments enhanced with frame inference for arbitrary inductive predicates represents a non-trivial extension of current theory, and would would greatly enhance the ability of a prover like Cyclist to function as a drop-in replacement prover for existing separation logic verification tools.

A secondary direction of this research theme, not directly related to separation logic analyses, is inductive theorem proving in the setting of first-order logic with inductively defined predicates and/or datatypes. This setting has been the historical focus of research in traditional explicit-induction theorem proving, and is also suited to expressing the correctness of functional programs. The Cyclist prover performs well on small examples requiring complicated induction schemes, which are highly problematic for proof search based on explicit induction. But, unfortunately, it currently lacks a facility for function definition over datatypes, which is essential for expressing most of the examples in existing test suites (e.g., theorems of Peano arithmetic). Implementing these missing features would enable serious comparisons to be made between cyclic theorem provers and explicit-induction theorem provers like IsaPlanner [13] and Zeno [21]. An intriguing alternative to implementing such schemes in Cyclist would be to alter the internal architecture of a mainstream theorem prover like Coq or Isabelle to use cyclic proof rather than explicit induction. This could have a huge impact, but would be a very risky undertaking since it would involve "rewiring" a major code base at a fundamental level. We would consider very carefully after an initial experimentation phase whether such an approach might be viable.

***Theme B: Cyclic proofs of program specifications.*** The Cyclist tool presently implements a version of the cyclic proof system for checking termination of imperative programs provided in [BBC08] (where preconditions are again written in separation logic). We believe this prototype can serve as a platform for building a cyclic prover capable of verifying arbitrary separation logic program specifications. This theme aims at making the necessary extensions to the system of [BBC08] and its associated implementation.

Firstly, we will extend the system of [BBC08] to work with postconditions as well as preconditions. Standard symbolic execution methods can normally approximate the strongest postcondition arising from running a program under a given precondition, but only for a fixed set of inductive definitions. Proposing a general abstraction method for arbitrary inductive predicates would enable us to extend Cyclist with postconditions, and constitute a non-trivial advance of the state of the art in its own right. Here, frame inference capability developed as part of Theme A would be crucial for computing postconditions. Also, the system includes memory safety in its notion of termination, but does not consider the case of memory safety only, which is more often of practical interest (e.g. for device drivers that are not intended to terminate). We believe that the system can be altered to establish memory safety properties by appropriately relaxing the global soundness condition on cyclic proofs.

Secondly, we will extend the programs handled by the prover to include procedure calls. There are standard ways of treating procedure calls in Hoare-style proof systems but, for obvious reasons, they require postconditions of procedures to be established, and they also require entailments to be established at procedure call sites. Since our pre- and postconditions will contain inductive predicates, this extension requires (in general) the integration of our program analysis with an inductive theorem prover of the type considered in Theme A above.

In many cases, the discovery of successful termination or safety proofs will again require the use of generalisation and/or lemma application mechanisms, of the type discussed above in the context of pure entailments.

***Theme C: Abduction of inductively defined predicates for program analysis.*** A problem of key importance in shape analysis, and program analysis in general, is in coming up with the definitions of the inductive predicates needed for a successful analysis. Typically, such predicates are either hard-coded into the analysis or must be provided to it by the user, which means that the analysis must either fail or ask the user for advice when it encounters data structures that are not described by the provided definitions. Brotherston and Gorogiannis recently described an exciting approach termed *cyclic abduction* to the automatic discovery of such predicates for termination analysis, based upon searching for a cyclic termination proof and abducing definitional clauses on the fly as necessary [BG12]. Their technique has once again been implemented in Cyclist, and the aim of this research theme is to overcome the current limitations of the technique and its implementation to extend the set of programs for which the right definitions can be inferred.

Firstly, we point out that the abduction mechanism, because it is based on proof search, is confined by the parameters of the cyclic proof system itself. Thus extending cyclic abduction to consider safety properties, postconditions, and programs with procedure calls depends on corresponding extensions of the native proof system for safety/termination, as described in research theme B above.

Yet, the challenges in the case of predicate inference are slightly different to those for simply establishing program properties under a given precondition. While we naturally want e.g. to establish memory safety of large programs, it is not in fact desirable to invent the inductive definition of a single predicate guaranteeing memory safety of the same program, because such a definition would likely be far too large and complicated to make any sense to a human. Rather, our aim should be to discover the (hopefully relatively simple) definitions of the *building blocks* of a program. Thus we believe the correct approach is to integrate cyclic abduction with existing shape analyses, allowing us to ignore portions of code that conform to shapes described by definitions already known to the analysis, and attempt to abduce definitions only for the unknown parts.

In the case of memory safety, there is a further interesting aspect because there is much more leeway for interpreting the definition obtained by abduction. In the case of a program intended to traverse a nil-terminated linked list, the most general predicate guaranteeing termination is an acyclic nil-terminated list, but the most general predicate guaranteeing memory safety might be considered to be a cyclic linked list, or even perhaps an infinite stream of pointers, which is most naturally defined *coinductively*, rather than inductively.

***Theme D: Verification of temporal properties of programs.*** Temporal logics, featuring "always" or "eventually" modalities, are frequently used in program analysis to express the evolution of programs during their execution. As these modalities can be expressed using standard fixed point constructions, it is reasonable to expect that we can formulate cyclic proof systems for entailment in such logics, and for verifying temporal logic properties of programs. However, an additional complication in such a setting is that many interesting properties require the temporal modalities to be nested, which corresponds to the nesting of least and greatest fixed points in $\mu$-calculus, or of inductive and coinductively defined predicates. This entails a significant extension to the global soundness condition on cyclic proofs to take account of the nesting of fixed points.

***Theme E: Cyclic proof in interprocedural program analysis.*** Having developed in the other research themes above cyclic proof methods for entailment, frame inference, abduction, and establishing properties of individual procedures, the aim of this theme is to integrate these methods, first, into an *intraprocedural* program analysis based on separation logic that aims to verify individual procedures of tens or perhaps hundreds of lines, and subsequently into an *interprocedural* program analysis that scales to larger code bases. The basis for such program analyses might be provided, e.g., by jStar [12] or Space Invader [24]. However, we suspect that the integration process is unlikely to be a simple matter of swapping in our cyclic theorem provers for general inductive predicates where these analysers presently call their own proof procedures for a few hard-coded inductive predicates. For example, these existing proof procedures are often decidable and therefore guaranteed terminating, whereas this seems highly unlikely to be true for the case of provers manipulating general inductive predicates.

*Linkage between research themes and outputs.* The themes outlined above feed into another, and we expect that an advance e.g. in the generalisation of conjectures will impact performance in all themes positively. However, apart from Theme E, progress in one theme does not strictly depend on progress in any of the others. Each of the themes poses both theoretical and implementation challenges; we fully expect that experimental evidence will impact on the theoretical integration of cyclic proof with other proof search mechanisms, as well as vice versa. The success of our research in each of the main themes will thus be measured by experimental performance of the implemented tools, as well as by academic publication of our ideas and results. A subsidiary output of the research will be a large corpus of test examples, harvested from the literature on program analysis and inductive theorem proving and from the proof queries raised by existing analysers.

**2.5 Academic impact.** This proposal aims to develop techniques and tools, based on the central technique of cyclic proof, for undertaking a range of different problems involving general inductive definitions that arise in interprocedural program analyses based on separation logic. We propose to test the efficacy of these techniques and tools by integrating them within such a large-scale analysis framework. If successful, this project has the potential to greatly boost the automation and program coverage of such analysis tools, which would have a huge impact. We note that the potential for substantial impact is by no means dependent on success in all areas of the proposal. For example, successful progress on the entailment problem for separation logic formulas employing arbitrary inductive predicates (see Theme A) has the potential for significant impact on the performance of existing verification tools, which routinely raise such queries during the course of an analysis. At the same time, we hope that our smaller-scale cyclic proof techniques for inventing the definitions of inductive predicates (Theme C), and for checking advanced safety and correctness properties (Theme B), will help to plug the gaps in the program coverage of existing shape analyses. Thus we see the main group of academic beneficiaries as researchers working on large-scale shape analyses using separation logic. We mention as examples the groups working in Microsoft Research Cambridge, Queen Mary, Imperial, Singapore, CMU and ITU Copenhagen.

We also expect that our research, if successful, would bring cyclic proof much closer to becoming a mainstream technique for inductive proof search. We believe that, until recently, the seemingly high technical design overhead associated with building a cyclic theorem prover has made it a less than attractive prospect for most researchers working on automatic inductive theorem proving. Our development of the Cyclist prover solved many of these design challenges and provides a potential basis for future provers; we believe the research programme above will let us take the next step and demonstrate the benefits of cyclic proof for automatic search. In the best case, one could hope we might show that cyclic proof can rival or outperform the capabilities of explicit-induction theorem proving. Thus we hope that researchers working on inductive proof search in a wide variety of settings will also benefit academically from this proposal. We mention in particular the very substantial number of researchers employing mainstream tools like Coq and Isabelle, as well as the groups working on inductive theorem proving e.g. at Edinburgh (IsaPlanner), Texas (ACL2), Imperial (Zeno), and INRIA / LIX (Tac).

## References

[1] Autexier, S., Hutter, D., Mantel, H., Schairer, A.: System description: Inka 5.0 - a logic voyager. In: Proceedings of CADE-16. Springer (1999)

[2] Avenhaus, J., Kühler, U., Schmidt-Samoa, T., Wirth, C.P.: How to prove inductive theorems? QuodLibet! In: Proceedings of CADE-19, pp. 328–333. Springer (2003)

[3] Berdine, J., Calcagno, C., O'Hearn, P.W.: Smallfoot: Modular automatic assertion checking with separation logic. In: Proc. FMCO, pp. 115–137. Springer (2005)

[4] Boyer, R.S., Moore, J.: A Computational Logic. Academic Press (1979)

[5] Brotherston, J.: Formalised inductive reasoning in the logic of bunched implications. In: Proceedings of SAS-14, pp. 87–103. Springer (2007)

[6] Brotherston, J., Distefano, D., Petersen, R.L.: Automated cyclic entailment proofs in separation logic. In: Proceedings of CADE-23, pp. 131–146. Springer (2011)

[7] Bundy, A.: The automation of proof by mathematical induction. In: Handbook of Automated Reasoning, vol. I, chap. 13, pp. 845–911. Elsevier Science (2001)

[8] Bundy, A., van Harmelen, F., Horn, C., Smaill, A.: The Oyster-Clam system. In: Proceedings of CADE-10, pp. 647–648. Springer (1990)

[9] Calcagno, C., Distefano, D., O'Hearn, P., Yang, H.: Compositional shape analysis by means of bi-abduction. Journal of the ACM 58(6) (Dec 2011)

[10] Dam, M., Gurov, D.: $\mu$-calculus with explicit points and approximations. J. Log. Comp. 12(2), 255–269 (2002)

[11] Distefano, D., O'Hearn, P.W., Yang, H.: A local shape analysis based on separation logic. In: Proceedings of TACAS-12, pp. 287–302. Springer (2006)

[12] Distefano, D., Parkinson, M.: jStar: Towards practical verification for Java. In: Proceedings of OOPSLA. pp. 213–226. ACM (2008)

[13] Dixon, L.: A Proof Planning Framework for Isabelle. Ph.D. thesis, University of Edinburgh (2005)

[14] Fogarty, S., Vardi, M.Y.: Büchi complementation and size-change termination. In: Proceedings of TACAS-15, pp. 16–30 (2009)

[15] Kaufmann, M., Manolios, P., Moore, J.S.: Computer-Aided Reasoning: An Approach. Kluwer (June 2000)

[16] Leveson, N.: Medical devices: The Therac-25. In: Safeware: System Safety & Computers. Addison-Wesley (1995)

[17] Mahoney, M.S.: The Mathematical Career of Pierre de Fermat. Princeton University Press, 2nd edn. (1994)

[18] Nguyen, H.H., Chin, W.N.: Enhancing program verification with lemmas. In: Proceedings of CAV, pp. 355–369. Springer (2008)

[19] Reynolds, J.C.: Separation logic: A logic for shared mutable data structures. In: Proceedings of LICS-17. pp. 55–74. IEEE Computer Society (2002)

[20] Schöpp, U., Simpson, A.: Verifying temporal properties using explicit approximants: Completeness for context-free processes. In: Proc. FoSSaCS, pp. 372–386. Springer (2002)

[21] Sonnex, W., Drossopoulou, S., Eisenbach, S.: Zeno: an automated prover for properties of recursive data structures. In: Proc. TACAS-18, pp. 407–421. Springer (2012)

[22] Stirling, C., Walker, D.: Local model checking in the modal $\mu$-calculus. Theor. Comp. Sci. 89, 161–177 (1991)

[23] Wirth, C.P.: Progress in computer-assisted inductive theorem proving by human-orientedness and descente infinie? Tech. Rep. SR-2006-01, Univ. Saarlandes (2010)

[24] Yang, H., Lee, O., Berdine, J., Calcagno, C., Cook, B., Distefano, D., O'Hearn, P.: Scalable shape analysis for systems code. In: Proceedings of CAV, pp. 385–398. Springer (2008)