Cyclic Proofs for First-Order Logic with Inductive Definitions

James Brotherston[1]

Laboratory for Foundations of Computer Science, Division of Informatics, University of Edinburgh, James Clerk Maxwell Building, King's Buildings, Mayfield Road, Edinburgh EH9 3JZ, Scotland, UK. Email: J.Brotherston@sms.ed.ac.uk

Abstract. We consider a cyclic approach to inductive reasoning in the setting of first-order logic with inductive definitions. We present a proof system for this language in which proofs are represented as finite, locally sound derivation trees with a "repeat function" identifying cyclic proof sections. Soundness is guaranteed by a well-foundedness condition formulated globally in terms of *traces* over the proof tree, following an idea due to Sprenger and Dam. However, in contrast to their work, our proof system does not require an extension of logical syntax by ordinal variables.

A fundamental question in our setting is the strength of the cyclic proof system compared to the more familiar use of a non-cyclic proof system using explicit induction rules. We show that the cyclic proof system subsumes the use of explicit induction rules. In addition, we provide machinery for manipulating and analysing the structure of cyclic proofs, based primarily on viewing them as generating regular infinite trees, and also formulate a finitary trace condition sufficient (but not necessary) for soundness, that is computationally and combinatorially simpler than the general trace condition.

1 Introduction

Induction is essential to computer science, and mathematics in general, as the fundamental principle by which to reason about many of the structures that are ubiquitous within the fields. Indeed, any inductively defined structure comes equipped with corresponding recursion and induction principles that in, e.g., a functional programming language, are invoked when defining functions by recursion on that structure and in proving mathematical properties of these functions, respectively. Similarly, structures defined by mutual induction give rise to mutual recursion and induction principles.

The default approach to inductive reasoning is to directly employ the induction principles associated with the structures under consideration; however, there has been recent interest [3–5, 7, 9, 15–20] in various forms of cyclic reasoning by which repeating sections of proof (cycles) are identified and various guardedness conditions are imposed on proofs to ensure their soundness. These conditions can be seen as encoding a termination argument or, more pertinently for our purposes, as ensuring well-foundedness of an underlying inductive argument. Forms

of cyclic, or circular, reasoning have been employed in local model checking [3], theorem proving tools and frameworks [4, 7, 9, 16], in Turchin's supercompilation [19] and in program verification based on automata [20]. It has also been studied in the context of tableau-style proof systems for the μ-calculus by Sprenger and Dam [17, 18] and Schöpp and Simpson [15], following an approach proposed by Dam and Gurov [5].

Our aim is to study cyclic reasoning in the relatively simple, yet expressive context of first-order logic extended with ordinary inductive definitions[1]. Similar formalisms typically form the basis of mechanised theorem proving tools [2, 8, 10, 13]. The contribution of this paper is twofold. Firstly, we present a sound, powerful cyclic proof system that employs only the standard syntax of first-order logic. It seems the system is most likely to be of use for proving properties of mutually inductive definitions, for which the usual induction rules are often highly complex. We examine the relationship of the cyclic proof system to a standard non-cyclic proof system; specifically, we show that the cyclic proof system is at least as powerful as the standard one. We conjecture that it is in fact no more powerful, but whether or not this is true remains open at the time of writing. (Although Sprenger and Dam give an equivalence result for cyclic and non-cyclic μ-calculus proof systems [18], the approach taken there is not obviously applicable to our systems due to significant differences in the setting, which we discuss later). Secondly, we present machinery for analysing the structure of cyclic proofs that, as well as being useful for our system for first-order logic with inductive definitions, should be more generally applicable to other cyclic proof systems based on analogous trace conditions.

The remainder of this paper is structured as follows. In section 2 we recall the syntax and semantics of first-order logic with inductive definitions ($\mathrm{FOL}_{\mathrm{ind}}$). In section 3 we introduce Gentzen-style sequent calculus rules for this language; in particular, we give rules for the case-split and induction principles associated with any inductively defined predicate. In section 4 we define two types of proof: the usual *structural proofs* familiar from sequent calculus, which use induction rules explicitly; and *cyclic proofs* represented as finite proof trees with a "repeat function" identifying cyclic proof sections, together with a global condition stipulating the existence of a *trace* on every infinite path through the proof, inspired by a similar condition proposed by Sprenger and Dam [17]. This condition guarantees the soundness of cyclic proofs. We prove that cyclic proofs subsume structural proofs by giving a translation from structural to cyclic proofs. In section 5 we present machinery, based on viewing cyclic proofs as generating infinite trees, for transforming cyclic proofs into ones having convenient structural properties. In section 6, we examine the global trace condition (which is the most general possible) imposed on cyclic proofs and formulate a localised *trace manifold* proof condition which, although less general, contains more structural information. Finally, in section 7 we summarise our progress and outline our aims for our ongoing work in this area.

[1] However, there is no difficulty in extending our approach to more complex formalisms such as iterated inductive definitions [11]

Due to space constraints, the proofs of many results in this paper have been omitted and the proofs that appear are only sketched. Full proofs will appear in the author's forthcoming PhD thesis.

2 Syntax and Semantics of $\mathbf{FOL_{ind}}$

We assume a fixed first-order signature Σ. Further, we assume that the predicate symbols of Σ are separated into two distinct types: *ordinary* and *inductive* predicate symbols. For convenience, we use vector notation to denote finite sequences of variables and terms. For example, if P is a predicate symbol of arity k, we shall write $P\boldsymbol{t}(\boldsymbol{x})$ for $P(t_1(x_1,\ldots,x_n),\ldots,t_k(x_1,\ldots,x_n))$.

We extend the standard syntax and semantics of first-order logic by introducing a simple definitional mechanism for (mutually) inductive predicates, based on Martin-Löf's schema for *ordinary productions* [11]:

Definition 2.1 (Inductive definition set). An *inductive definition set* Φ is a finite set of *productions* of the form:

$$\frac{P_1\boldsymbol{t_1}(\boldsymbol{x})\ \ldots\ P_m\boldsymbol{t_m}(\boldsymbol{x})}{P\boldsymbol{t}(\boldsymbol{x})}$$

where P is an inductive predicate symbol and $P_1,\ldots,P_m$ may be either ordinary or inductive predicate symbols.

Example 2.2. We define the predicates N,E and O via the productions:

$$\frac{}{N0}\quad\frac{Nx}{Nsx}\qquad\frac{}{E0}\quad\frac{Ex}{Osx}\qquad\frac{Ox}{Esx}$$

In structures in which all "numerals" s^k0 for $k \geq 0$ are interpreted as distinct elements, the predicates N,E and O correspond to the properties of being a natural, even and odd number respectively.

We next define the interpretation of inductively defined predicates in a Σ-structure M with domain D. Given a definition set Φ for predicates $P_1,\ldots,P_n$ of arities $k_1,\ldots,k_n$ respectively, writing $Pow(A)$ for the powerset of A, one can construct a monotone operator φ_Φ with domain and codomain $(Pow(D^{k_1}),\ldots,Pow(D^{k_n}))$ whose least fixed point is the n-tuple of least subsets of (tuples of) D closed under the rules in Φ. This least fixed point can be approached in stages, by defining an ordinal-indexed sequence (φ^α_Φ) by $\varphi^\alpha_\Phi = \bigcup_{\beta<\alpha}\varphi_\Phi(\varphi^\beta_\Phi)$. Note that we thus have $\varphi^0_\Phi = \emptyset$. The i^{th} component of φ^α_Φ is then called the α^{th} *approximant*[2] *of* P_i, written as P_i^α. For a full exposition of the construction of the operator φ_Φ (which is standard), see e.g. Aczel [1].

[2] One can show that for any predicate P defined by our schema, $P^\alpha = P^\omega$ for any $\alpha > \omega$, so it is sufficient to index approximants by natural numbers. However, the sequence of approximants does not necessarily close at ω if more complex schemas are used. We retain the ordinal notation for modularity.

We consider standard first-order formulas over Σ. The semantics of formulas are defined as usual except for the semantics of atomic formulas, which are now defined as follows, where ρ is an environment interpreting free variables in M and P^M is the interpretation of the ordinary predicate symbol P in M:

$$M \models_\rho P\boldsymbol{t} \iff \begin{cases} \rho(\boldsymbol{t}) \in \bigcup_\alpha P^\alpha & \text{if } P \text{ is an inductively defined predicate symbol} \\ \rho(\boldsymbol{t}) \in P^M & \text{otherwise} \end{cases}$$

3 Sequent Calculus Proof Rules for $\mathbf{FOL_{ind}}$

We shall consider proof systems for $\mathrm{FOL_{ind}}$ presented in the sequent calculus style originally due to Gentzen [6], which is well-established as a convenient formalism for proof-theoretic reasoning. Our rules for inductively defined predicates are essentially sequent-calculus adaptations of Martin-Löf's natural deduction rules [11]; McDowell and Miller have considered a similar system [12]. We write sequents of the form $\Gamma \vdash \Delta$, where Γ, Δ are finite multisets of formulas. We use the standard sequent calculus rules for the propositional connectives and quantifiers, as well as the following structural rules:

$$\frac{\Gamma' \vdash \Delta'}{\Gamma \vdash \Delta}\ \Gamma' \subseteq \Gamma, \Delta' \subseteq \Delta\,(\mathrm{Wk}) \qquad \frac{}{\Gamma \vdash \Delta}\ \Gamma \cap \Delta \neq \emptyset\ (\mathrm{Axiom})$$

$$\frac{\Gamma, F, F \vdash \Delta}{\Gamma, F \vdash \Delta}\,(\mathrm{ContrL}) \qquad \frac{\Gamma \vdash F, F, \Delta}{\Gamma \vdash F, \Delta}\,(\mathrm{ContrR})$$

$$\frac{\Gamma \vdash F, \Delta \qquad \Gamma, F \vdash \Delta}{\Gamma \vdash \Delta}\,(\mathrm{Cut}) \qquad \frac{\Gamma \vdash \Delta}{\Gamma[\theta] \vdash \Delta[\theta]}\,(\mathrm{Subst})$$

We also give proof rules governing the inductively defined predicates. First, for each ordinary production in Φ we obtain a right-introduction rule for an inductively defined predicate as follows:

$$\frac{P_1\boldsymbol{t_1}(\boldsymbol{x}) \ldots P_m\boldsymbol{t_m}(\boldsymbol{x})}{P\boldsymbol{t}(\boldsymbol{x})} \Rightarrow \frac{\Gamma \vdash P_1\boldsymbol{t_1}(\boldsymbol{t'_1}), \Delta \ldots \Gamma \vdash P_m\boldsymbol{t_m}(\boldsymbol{t'_m}), \Delta}{\Gamma \vdash P\boldsymbol{t}(\boldsymbol{t'}), \Delta}\,(PR_r)$$

Example 3.1. The right introduction rules for the predicate N defined in Example 2.2 are:

$$\frac{}{\Gamma \vdash N0, \Delta}\,(NR_1) \qquad \frac{\Gamma \vdash Nt, \Delta}{\Gamma \vdash Nst, \Delta}\,(NR_2)$$

Definition 3.2 (Mutual dependency). Let Φ be a fixed inductive definition set. Define the binary relation $Prem$ on *inductive* predicate symbols as the least relation satisfying: whenever there is a production in Φ containing P in the conclusion and Q among the premises, then $Prem(P, Q)$ holds. Also define $Prem^*$ to be the reflexive-transitive closure of $Prem$. Then we say two predicate symbols P and Q are *mutually dependent* if both $Prem^*(P, Q)$ and $Prem^*(Q, P)$ hold.

We now describe the construction of the induction rule for an inductively defined predicate P. First, for all predicates P_i such that P_i and P are mutually dependent,we first associate a formula $F_i\boldsymbol{z}$ with P_i, where P_i is a k-ary predicate and $\boldsymbol{z}$ is a vector of k variables. Then the induction rule schema is:

$$\frac{\text{minor premises} \qquad \Gamma, F\boldsymbol{t} \vdash \Delta}{\Gamma, P\boldsymbol{t} \vdash \Delta}\ (\text{Ind } P)$$

where $F\boldsymbol{z}$ is the formula associated with P. Then for all productions having in their conclusion a predicate P_i such that P_i and P are mutually dependent, we obtain a *minor premise* as follows[3]:

$$\frac{P_1\boldsymbol{t_1}(\boldsymbol{x}) \ldots\ P_m\boldsymbol{t_m}(\boldsymbol{x})}{P_i\boldsymbol{t}(\boldsymbol{x})} \quad \Rightarrow \quad \Gamma, G_1\boldsymbol{t_1}(\boldsymbol{x}), \ldots, G_m\boldsymbol{t_m}(\boldsymbol{x}) \vdash F_i\boldsymbol{t}(\boldsymbol{x}), \Delta$$

where $x \notin FV(\Gamma \cup \Delta)$ for all $x \in \boldsymbol{x}$ and G_k is defined for each $k \in \{1, \ldots, m\}$ by:

$$G_k = \begin{cases} F_k & \text{if } P_k \text{ and } P \text{ are mutually dependent} \\ P_k & \text{otherwise} \end{cases}$$

Example 3.3. The induction rule for the even number predicate E defined in Example 2.2 is the following, where F and G are associated with E and O respectively:

$$\frac{\Gamma \vdash F0, \Delta \qquad \Gamma, Fx \vdash Gsx, \Delta \qquad \Gamma, Gx \vdash Fsx, \Delta \qquad \Gamma, Ft \vdash \Delta}{\Gamma, Et \vdash \Delta}\ (\text{Ind } E)$$

We also consider rules implementing a *case-splitting* principle on inductively defined predicates. (Although case-splitting is a much weaker principle than induction, and indeed is subsumed by it, we shall see in Section 4 that cyclic reasoning together with the use of case-split rules subsumes explicit use of the induction rules above.) The case-split rule schema for an inductively defined predicate P is:

$$\frac{\text{case distinctions}}{\Gamma, P\boldsymbol{y} \vdash \Delta}\ (\text{Case } P)$$

where $\boldsymbol{y}$ is a vector of variables and for each production having predicate P in its conclusion, we obtain a *case distinction* as follows:

$$\frac{P_1\boldsymbol{t_1}(\boldsymbol{x}) \ \ldots\ P_m\boldsymbol{t_m}(\boldsymbol{x})}{P\boldsymbol{t}(\boldsymbol{x})} \Rightarrow \Gamma[\boldsymbol{t}(\boldsymbol{x})/\boldsymbol{y}], P\boldsymbol{t}(\boldsymbol{x}), P_1\boldsymbol{t_1}(\boldsymbol{x}), \ldots, P_m\boldsymbol{t_m}(\boldsymbol{x}) \vdash \Delta[\boldsymbol{t}(\boldsymbol{x})/\boldsymbol{y}]$$

subject to the restriction that $x \notin FV(\Gamma \cup \Delta)$ for all $x \in \boldsymbol{x}$. In such rules, the formulas $P_1\boldsymbol{t_1}(\boldsymbol{x}), \ldots, P_m\boldsymbol{t_m}(\boldsymbol{x})$ are said to be *case-descendents* of the *principal formula* $P\boldsymbol{y}$ in the rule conclusion.

[3] Martin-Löf [11] uses a definition of "linkage" between predicate symbols to generate the minor deductions (corresponding to our minor premises) of his induction rules. However, his definition can produce redundant minor deductions in some cases. Our use of Definition 3.2 is intended to avoid this.

Example 3.4. The case-split rule for the natural number predicate N is:

$$\frac{\Gamma[0/y] \vdash \Delta[0/y] \qquad \Gamma[sx/y], Nsx, Nx \vdash \Delta[sx/y]}{\Gamma, Ny \vdash \Delta} \text{ (Case } N\text{)}$$

4 Structural and Cyclic Proofs

We now proceed to give definitions of *structural proofs* familiar from traditional sequent calculus proof systems, which are finite trees labelled with sequents and proof rules. We do this with more formality than is usual, based on the notion of a *rule graph*, in order to later facilitate a formal comparison between structural proofs and cyclic proofs. Note that we write $f : X \rightharpoonup Y$ to denote a partial function and $f : X \to Y$ to denote a total function from X to Y.

Definition 4.1 (Rule graph). Let *Seqs* denote the set of all well-formed sequents in some language and *Rules* denote some set of rules. Also let $n \in \mathbb{N}$ be the maximum number of premises of any $R \in \mathit{Rules}$. Then a *rule graph* is given by (V, s, r, p), where:

- V is a set of vertices, $s : V \to \mathit{Seqs}$, $r : V \rightharpoonup \mathit{Rules}$, and $p : V \rightharpoonup V^n$ (we write $p_j(v)$ for the j^{th} component of $p(v)$);
- for all $v \in V$, $p_j(v)$ is defined just in case $r(v)$ is a rule with m premises, $1 \leq j \leq m$ and:

$$\frac{s(p_1(v)) \quad \ldots \quad s(p_m(v))}{s(v)}$$

  is an instance of rule $r(v)$.

A rule graph G can be seen as a conventional graph whose vertex set is V and whose edge set is $E = \{(v, p_j(v)) \mid v \in V \text{ and } p_j(v) \text{ is defined}\}$.

A *path* in a rule graph is a (possibly infinite) sequence $v_0 j_0 v_1 j_1 v_2 j_2 \ldots$ such that for each $i \geq 0$, $v_{i+1} = p_{j_i}(v_i)$. (We often write paths simply as $v_0 v_1 v_2 \ldots$)

Definition 4.2 (Derivation tree). A rule graph $\mathcal{D} = (V, s, r, p)$ is a *derivation tree* if there is a distinguished node $v_0 \in V$ such that for all $v \in V$, there is a unique path in $\mathcal{D}$ from v_0 to v. v_0 is called the *root* of the tree, written $\mathit{root}(\mathcal{D})$.

Definition 4.3 (Structural proof). A *structural proof* of $\Gamma \vdash \Delta$ is a finite derivation tree $\mathcal{D} = (V, s, r, p)$ such that:

- the codomain of s is the set of all well-formed sequents of FOL_{ind};
- $s(\mathit{root}(\mathcal{D})) = \Gamma \vdash \Delta$;
- the codomain of r comprises the rules described in Section 3 above;
- r is a total function on V, i.e. every node in $\mathcal{D}$ is the conclusion of some rule instance. (Note that we consider axioms to be proof rules with 0 premises.)

Definition 4.4 (Satisfaction). Let M be a fixed Σ-structure and let ρ be an environment interpeting free variables in M. We write $\Gamma \models_\rho \Delta$ to mean that if $M \models_\rho J$ for all $J \in \Gamma$ then there is a $K \in \Delta$ such that $M \models_\rho K$. We write $\Gamma \models \Delta$ if $\Gamma \models_\rho \Delta$ for all ρ.

Theorem 4.5 (Soundness of structural proof). *If there is a structural proof of $\Gamma \vdash \Delta$ then $\Gamma \models \Delta$.*

Proof. Routine. □

Proposition 4.6. *For any inductively defined predicate P, the rule (Case P) is subsumed by the rule (Ind P) in structural proofs. Specifically, any instance of the rule (Case P) in a structural proof can be replaced by an equivalent derivation containing an instance of (Ind P) and instances of the standard sequent calculus rules for first-order logic, only.*

The well-known and important *cut elimination* theorem for sequent calculus for ordinary first-order logic due to Gentzen [6] says that any use of the (Cut) rule (which corresponds to the use of auxiliary lemmas) in a derivation can be avoided. This result should generalise to our structural proof system for $\mathrm{FOL}_{\mathrm{ind}}$. However, to our knowledge no proof for such a classical sequent calculus has so far appeared in the literature , although a proof for a related intuitionistic proof system appears in McDowell and Miller [12].

4.1 A Cyclic Proof System for $\mathbf{FOL_{ind}}$

We now proceed to define a cyclic proof system for $\mathrm{FOL}_{\mathrm{ind}}$, in which the proof structures are finite derivation trees together with a function assigning to every unexpanded node in the proof tree (called a *bud*) an interior node with an identical sequent labelling (the *companion* of the bud). These structures (called *pre-proofs*) can then be viewed as cyclic rule graphs:

Definition 4.7 (Bud /companion nodes). Let $\mathcal{D} = (V, s, r, p)$ be a derivation tree. A *bud node* of $\mathcal{D}$ is a vertex $B \in V$ such that $r(B)$ is undefined, i.e. B is not the conclusion of any proof rule instance in $\mathcal{D}$.

A node $C \in V$ is said to be a *companion* for a bud node B if $r(C)$ is defined and $s(C) = s(B)$.

We remark that we do not require buds to have ancestor nodes as companions, i.e. C need not appear on the unique path in $\mathcal{D}$ from $root(\mathcal{D})$ to B.

Definition 4.8 (Cyclic pre-proof). A *cyclic pre-proof* (or simply *pre-proof*) of $\Gamma \vdash \Delta$ is a pair $(\mathcal{D} = (V, s, r, p), \mathcal{R})$, where $\mathcal{D}$ is a finite derivation tree and:

- the codomain of s is the set of all well-formed sequents of $\mathrm{FOL}_{\mathrm{ind}}$;
- $s(root(\mathcal{D})) = \Gamma \vdash \Delta$;
- the codomain of r comprises the rules described in Section 3 above *except* for the induction rules;
- $\mathcal{R} : V \rightharpoonup V$ is a function assigning a companion to every bud node in $\mathcal{D}$.

We shall see shortly that our embargo on the explicit use of induction rules in the cyclic proof system is of no consequence; all induction rules turn out to be derivable within the system.

Definition 4.9 (Pre-proof graph). Let $\mathcal{P} = (\mathcal{D}, \mathcal{R})$ be a pre-proof, where $\mathcal{D} = (V, s, r, p)$. Then the *graph* of $\mathcal{P}$ is $\mathcal{G}_\mathcal{P} = (V', s, r, p)$, where V' is obtained from V by identifying each bud node B in $\mathcal{D}$ with its companion $\mathcal{R}(B)$.

We observe that the local soundness of our proof rules is not sufficient to guarantee that pre-proofs are sound, due to the (possible) cyclicity evident in their graph representations. In order to give a criterion for soundness, we follow Sprenger and Dam [17] in formulating the notion of a *trace* following a path:

Definition 4.10 (Trace). Let $\mathcal{P}$ be a pre-proof and let (v_i) be a path in $\mathcal{G}_\mathcal{P} = (V', s, r, p)$. A *trace following* (v_i) is a sequence (τ_i) such that, for all i:

- $\tau_i = P_i\boldsymbol{t_i} \in \Gamma_i$, where $s(v_i) = \Gamma_i \vdash \Delta_i$ and P_i is inductively defined;
- if $r(v_i)$ is (Subst) then $\tau_i = \tau_{i+1}[\Theta]$, where Θ is the substitution associated with the rule instance;
- if $r(v_i)$ is (Case P) then either $\tau_{i+1} = \tau_i[\boldsymbol{t}(\boldsymbol{x})/\boldsymbol{y}]$, where $[\boldsymbol{t}(\boldsymbol{x})/\boldsymbol{y}]$ is the substitution associated with the case distinction at $s(v_{i+1})$, or τ_i is the principal formula $P\boldsymbol{y}$ of the rule instance and τ_{i+1} is a case-descendent of $P\boldsymbol{y}$. In the latter case, i is said to be a *progress point* of the trace;
- if $r(v_i)$ is not (Subst) or (Case P), then $\tau_{i+1} = \tau_i$.

An *infinitely progressing trace* is a (necessarily infinite) trace having infinitely many progress points.

Informally, a trace follows (a part of) the construction of an inductively defined predicate occurring in the left hand side of the sequents occurring on a path in a pre-proof. These predicate constructions never become larger as we follow the trace along the path, and at progress points, they actually decrease. This property is encapsulated in the following lemma and motivates the subsequent definition of a *cyclic proof*:

Lemma 4.11. *Suppose we have a pre-proof of $\Gamma_0 \vdash \Delta_0$, but there exists ρ_0 such that $\Gamma_0 \not\models_{\rho_0} \Delta_0$. Then there is an infinite path $(v_i)_{i\geq 0}$ in $\mathcal{G}_\mathcal{P}$ and an infinite sequence of environments $(\rho_i)_{i\geq 0}$ such that:*

1. *For all i, $\Gamma_i \not\models_{\rho_i} \Delta_i$, where $s(v_i) = \Gamma_i \vdash \Delta_i$;*
2. *If there is a trace $(\tau_i)_{i\geq n}$ following some tail $(v_i)_{i\geq n}$ of $(v_i)_{i\geq 0}$, then the sequence $(\alpha_i)_{i\geq n}$ of ordinals defined by $\alpha_i = (\text{least } \alpha \text{ s.t. } \rho_i(\boldsymbol{t_i}) \in P_i^\alpha \text{ where } \tau_i = P_i\boldsymbol{t_i})$ is non-increasing. Furthermore, if j is a progress point of the trace then $\alpha_{j+1} < \alpha_j$.*

Proof. (Sketch) One constructs (v_i) and (ρ_i) inductively by assuming sequences $(v_i)_{0\leq i\leq k}$ and $(\rho_i)_{0\leq i\leq k}$ satisfying the two properties of the lemma and constructing v_{k+1} and ρ_{k+1}. The proof proceeds by a case analysis on the rule $r(v_k)$; the main interesting case is when $r(v)$ is a case-split rule. □

Definition 4.12 (Cyclic proof). A pre-proof $\mathcal{P}$ is said to be a *cyclic proof* if, for every infinite path in $\mathcal{G}_\mathcal{P}$, there is an infinitely progressing trace following some tail of the path.

Theorem 4.13 (Soundness of cyclic proof). *If the sequent $\Gamma \vdash \Delta$ has a cyclic proof then $\Gamma \models \Delta$.*

Proof. (Sketch) If $\Gamma \not\models \Delta$ then we can apply Lemma 4.11 to construct infinite sequences (v_i) and (ρ_i) satisfying the two properties of the lemma. By Definition 4.12 there is an infinitely progressing trace following some tail of (v_i), so by the second property of the lemma we can construct an infinite descending chain of ordinals, which is a contradiction. □

Importantly, the property of being a cyclic proof is decidable; the problem can be reduced to the problem of checking the language of a certain Büchi automaton for emptiness. A thorough analysis of automata-theoretic decision methods is given by Sprenger and Dam [17].

Example 4.14. The following is a cyclic proof of the statement $Ex \vee Ox \vdash Nx$, where N,E and O are as defined in Example 2.2 (we omit applications of weakening):

$$\dfrac{\dfrac{}{\vdash N0}\,(NR_1) \qquad \dfrac{\dfrac{\dfrac{Ox \vdash Nx\ (\dagger)}{Oy \vdash Ny}\,(\text{Subst})}{Oy \vdash Nsy}\,(NR_2)}{Ex \vdash Nx\ (*)}\,(\text{Case } E) \qquad \dfrac{\dfrac{\dfrac{(*)\ Ex \vdash Nx}{Ey \vdash Ny}\,(\text{Subst})}{Ey \vdash Nsy}\,(NR_2)}{(\dagger)\ Ox \vdash Nx}\,(\text{Case } O)}{Ex \vee Ox \vdash Nx}\,(\vee\text{L})$$

We use pairs of identical symbols $(*)$ and $(\dagger)$ above to indicate the pairing of buds with companions; we remark that this is an example of a proof in which bud nodes have non-ancestor nodes as companions. To see that this satisfies the cyclic proof condition, observe that any infinite path through the pre-proof necessarily has a tail consisting of repetitions of the "figure-of-8" loop in this proof, and there is a trace following this path: $(Ex, Oy, Oy, Ox \equiv Ox, Ey, Ey, Ex)$ (starting from $(*)$) that progresses at Ex and at Ox. We can thus concatenate copies of this trace to obtain an infinitely progressing trace as required.

Next, we address the fundamental question of the relative strengths of the standard structural proof system and the cyclic proof system. Our first result establishes that cyclic proofs are at least as powerful as structural proofs:

Theorem 4.15. *If there is a structural proof of $\Gamma \vdash \Delta$ then there is a cyclic proof of $\Gamma \vdash \Delta$.*

Proof. (Sketch) One shows that any use of an induction rule within a structural proof can be replaced with a derivation in our cyclic proof system. Each derivation contains a cyclic proof, constructed using the minor premises of the induction rule, which essentially is an explicit justification of local soundness. Moreover, as each of these cyclic proofs is self-contained, it follows easily that the result of uniformly substituting these derivations for induction rules is also a cyclic proof.

We illustrate the procedure by demonstrating the derivation of the rule (Ind E) given in Example 3.3. First of all, define the set of formulas $M_E = \{F0, \forall x.Fx \to Gsx, \forall x.Gx \to Fsx\}$. We start as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{M_E, Ey \vdash Fy}{M_E, Et \vdash Ft}\,(\text{Subst})}{\vdots}\,(\wedge\text{L})}{\bigwedge M_E, Et \vdash Ft}\,(\wedge\text{L})}{\bigwedge M_E, \Gamma, Et \vdash Ft, \Delta}\,(\text{Wk}) \qquad \dfrac{\begin{array}{c}\text{minor premises}\\ \vdots\\ \Gamma \vdash \bigwedge M_E, \Delta\end{array}}{\Gamma, Et \vdash \bigwedge M_E, Ft, \Delta}\,(\text{Wk})}{\Gamma, Et \vdash Ft, \Delta}\,(\text{Cut}) \qquad \dfrac{\Gamma, Ft \vdash \Delta}{\Gamma, Et, Ft \vdash \Delta}\,(\text{Wk})}{\Gamma, Et \vdash \Delta}\,(\text{Cut})$$

Obtaining the minor premises of (Ind E) from $\Gamma \vdash \bigwedge M_E, \Delta$ is straightforward. We continue by providing a cyclic proof of the sequent $M_E, Ey \vdash Fy$ occurring on the leftmost branch as follows (omitting applications of weakening):

$$\dfrac{\dfrac{}{M_E \vdash F0}\,(\text{Ax}) \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{Fsx \vdash Fsx}\,(\text{Ax}) \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{Gsy \vdash Gsy}\,(\text{Ax}) \qquad M_E, Ey \vdash Fy\ (*)}{M_E, Fy \to Gsy, Ey \vdash Gsy}\,(\to\text{L})}{M_E, \forall x.Fx \to Gsx, Ey \vdash Gsy}\,(\forall\text{L})}{M_E, Ey \vdash Gsy}\,(\text{ContrL})}{M_E, Ox \vdash Gx}\,(\text{Case } O)}{M_E, Gx \to Fsx, Ox \vdash Fsx}\,(\to\text{L})}{M_E, \forall x.Gx \to Fsx, Ox \vdash Fsx}\,(\forall\text{L})}{M_E, Ox \vdash Fsx}\,(\text{ContrL})}{M_E, Ey \vdash Fy\ (*)}\,(\text{Case } E)$$

To the single bud node of this derivation we assign the root sequent as companion (indicated by $(*)$). As there is a progressing trace from the companion to the bud, this is easily seen to be a cyclic proof.

We remark at this juncture that, although structural rules such as weakening or contraction are admissible in ordinary FOL sequent calculus, they clearly are essential to the cyclic proof system as we have formulated it here, as their removal can break the required syntactic identity between bud nodes and their companions.

Theorem 4.15 gives rise to the obvious question of whether its converse also holds:

Conjecture 4.16. *If there is a cyclic proof of $\Gamma \vdash \Delta$ then there is a structural proof of $\Gamma \vdash \Delta$.*

This is the main open question of our current research. An equivalence result for cyclic and non-cyclic proof systems for μ-calculus with explicit approximations, given by Sprenger and Dam [18], gives some hope that it can be answered positively. However, this result is based on reducing cyclic reasoning with a syntax extended by ordinal variables to a principle of transfinite induction also expressed using ordinal variables. The problem of reducing cyclic reasoning on ordinary syntax to induction over predicate definitions seems significantly harder. The most promising line of inquiry appears to be to establish a translation from cyclic to structural proofs. (A semantic proof would also be of interest, but we have no idea how to obtain one.)

5 Unfolding and Folding of Cyclic Proofs

In this section and the subsequent one we study the structure of cyclic proofs, which we view here as finite representations of infinite regular proof trees. As many different cyclic proofs can represent the same infinite tree, these trees give us a natural notion of equivalence on cyclic proofs. We give machinery based on infinite trees for transforming cyclic proofs into equivalent ones having simpler combinatorial structure. We hope that such concerns may be useful in eventually establishing a translation from cyclic to structural proofs.

Definition 5.1 (Associated tree). Let $\mathcal{P} = (\mathcal{D}, \mathcal{R})$ be a pre-proof with graph $\mathcal{G}_\mathcal{P} = (V', s, r, p)$. Define $Path(\mathcal{G}_\mathcal{P})$ to be the set of finite paths through $\mathcal{G}_\mathcal{P}$ starting from $v_0 = root(\mathcal{D})$. Then the *tree of* $\mathcal{P}$ is $\mathcal{T}_\mathcal{P} = (Path(\mathcal{G}_\mathcal{P}), s^*, r^*, p^*)$, where $s^*((v_i)_{0\leq i\leq n}) = s(v_n)$, $r^*((v_i)_{0\leq i\leq n}) = r(v_n)$, and:

$$p_j^*((v_i)_{0\leq i\leq n}) = \begin{cases} ((v_i)_{0\leq i\leq n}.p_j(v_n)) & \text{if } p_j(v_n) \text{ defined} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Proposition 5.2. *For any pre-proof $\mathcal{P}$, $\mathcal{T}_\mathcal{P}$ is a derivation tree.*

We write $f(x) \simeq g(x)$, where f and g are partial functions, to mean that $f(x)$ is defined iff $g(x)$ is defined, and if $f(x)$ is defined then $f(x) = g(x)$.

Definition 5.3 (Rule graph homomorphism). Let $G = (V, s, r, p)$ and $H = (V', s', r', p')$ be rule graphs. A *rule graph homomorphism* from G to H is a map $f : V \to V'$ satisfying, for all $v \in V$, $s'(f(v)) = s(v)$, $r'(f(v)) \simeq r(v)$ and $p'_j(f(v))) \simeq f(p_j(v))$.

We say two rule graphs G and H are *isomorphic*, written $G \cong H$, if there exist rule graph homomorphisms $f : G \to H$ and $g : H \to G$ such that $f \circ g = g \circ f = id$, where id is the identity function.

Lemma 5.4. *For any pre-proof $\mathcal{P} = (\mathcal{D}, \mathcal{R})$, there is a surjective rule graph homomorphism $f_\mathcal{P} : \mathcal{T}_\mathcal{P} \to \mathcal{G}_\mathcal{P}$ such that $f_\mathcal{P}(root(\mathcal{T}_\mathcal{P})) = root(\mathcal{D})$.*

Lemma 5.5. *If $\mathcal{T}_\mathcal{P} \cong \mathcal{T}_{\mathcal{P}'}$ then $\mathcal{P}$ is a cyclic proof if and only if $\mathcal{P}'$ is.*

Theorem 5.6. *Let G be a rule graph, let T_1, T_2 be derivation trees, and let $f_1 : T_1 \to G$ and $f_2 : T_2 \to G$ be rule graph homomorphisms such that $f_1(root(T_1)) = f_2(root(T_2))$. Then $T_1 \cong T_2$.*

The following is a useful general theorem for extracting a pre-proof from a (possibly infinite) proof tree:

Theorem 5.7. *Let $T = (V, s, r, p)$ be a derivation tree with no bud nodes and with root node v_0, let G be a finite rule graph, and let $f : T \to G$ be a surjective rule graph homomorphism. Also, for each infinite branch $\pi = v_0 v_1 v_2 \dots$ in T, let $m_\pi < n_\pi$ be numbers such that $f(v_{m_\pi}) = f(v_{n_\pi})$. Then we define $\mathcal{D} = (V', s', r', p')$ and $\mathcal{R}$ by "folding down" T as follows:*

- *$V' = \{v \in V \mid$ for all infinite $\pi = v_0 v_1 v_2 \dots .$ if $\exists k. v = v_k$ then $k \leq n_\pi\}$*
- *$s'(v) = s(v)$ for all $v \in V' (\subset V)$*
- *if $v \in V'$ and $\exists \pi = v_0 v_1 v_2 \dots$ s.t. $v = v_{n_\pi}$, then $r'(v)$ and $p'(v)$ are undefined, i.e. v is a bud node of $\mathcal{D}$ and we define $\mathcal{R}(v) = v_{m_\pi}$. (If there is more than one branch π meeting this criterion, we may choose any suitable v_{m_π}.) Otherwise we define $r'(v) = r(v)$ and $p'(v) = p(v)$.*

Then $\mathcal{P} = (\mathcal{D}, \mathcal{R})$ is a pre-proof and there are surjective rule graph homomorphisms $T \to \mathcal{G}_\mathcal{P} \to G$. Furthermore, the homomorphism from T to $\mathcal{G}_\mathcal{P}$ maps $v_0 = root(T)$ to $v_0 = root(\mathcal{D})$.

Our intended application of Theorem 5.7 is to obtain a cyclic proof $\mathcal{P}'$ with convenient structural properties from a given cyclic proof $\mathcal{P}$ by "folding down" $\mathcal{T}_\mathcal{P}$. This application is illustrated by our next result.

Definition 5.8 (Cycle normal form). Let $\mathcal{P} = (\mathcal{D}, \mathcal{R})$ be a cyclic pre-proof. $\mathcal{P}$ is said to be in *cycle normal form* if, for every bud node B in $\mathcal{D}$, its companion $\mathcal{R}(B)$ is a (strict) ancestor of B.

Theorem 5.9. *Let $\mathcal{P}$ be a cyclic proof and for each infinite branch $\pi = v_0 v_1 v_2 \dots$ in $\mathcal{T}_\mathcal{P}$, let $m_\pi < n_\pi$ be numbers such that $f_\mathcal{P}(m_\pi) = f_\mathcal{P}(n_\pi)$. Then* any *pre-proof obtained from Theorem 5.7 by "folding down" $\mathcal{T}_\mathcal{P}$ is a cyclic proof, and furthermore is in cycle normal form.*

Proof. As there is a surjective rule graph homomorphism from $\mathcal{T}_\mathcal{P}$ to $\mathcal{G}_\mathcal{P}$ by Lemma 5.4, we can apply Theorem 5.7 to obtain a pre-proof $\mathcal{P}' = (\mathcal{D}', \mathcal{R}')$, and by the theorem, there is a surjective rule graph homomorphism $g : \mathcal{T}_\mathcal{P} \to \mathcal{G}_{\mathcal{P}'}$ such that $g(root(\mathcal{T}_\mathcal{P})) = root(\mathcal{D}')$. As there is also a surjective rule graph homomorphism $f_{\mathcal{P}'} : \mathcal{T}_{\mathcal{P}'} \to \mathcal{G}_{\mathcal{P}'}$ such that $f_{\mathcal{P}'}(root(\mathcal{T}_\mathcal{P})) = root(\mathcal{D}')$, again by Lemma 5.4, we can apply Theorem 5.6 to conclude $\mathcal{T}_\mathcal{P} \cong \mathcal{T}_{\mathcal{P}'}$, and since $\mathcal{P}$ is a cyclic proof, so is $\mathcal{P}'$ by Lemma 5.5.

To see that $\mathcal{P}'$ is in cycle normal form, we simply observe that the construction of Theorem 5.7 ensures $\mathcal{R}'(B)$ is always an ancestor of B for any bud node B. □

We remark that we also have a direct proof of cycle-normalisation (the transformation of an arbitrary cyclic proof to one in cycle normal form), which will appear in the author's PhD thesis.

6 Trace-Based Proof Conditions

The general trace condition (cf. Definition 4.12) qualifying pre-proofs as cyclic proofs is both computationally and combinatorially complex. In order to simplify our analysis of cyclic proof structures, we consider the formulation of alternative trace conditions that are sufficient for pre-proofs to be cyclic proofs, and that also provide a greater degree of explicit structural information on pre-proofs.

We present one definition of such a condition — the existence of a so-called *trace manifold* for a pre-proof — which is apparently less general than Definition 4.12 and formulated with respect to a so-called *induction order* (a notion introduced by Schöpp [14] and crucially employed by Sprenger and Dam [18]). A trace manifold consists of finite trace segments together with conditions ensuring that for any infinite path, the segments can be "glued together" to yield an infinitely progressing trace on that path.

Definition 6.1 (Strong / weak connectivity). A directed graph $G = (V, E)$ is said to be *strongly connected* if for any $v, v' \in V$, there is a path in G from v to v'. G is said to be *weakly connected* if $(V, E \cup E^{-1})$ is strongly connected.

Definition 6.2 (Structural connectivity). Let $\mathcal{P} = (\mathcal{D}, \mathcal{R})$ be a pre-proof in cycle normal form and denote the set of bud nodes occuring in $\mathcal{D}$ by $\mathcal{B}$. Define the relation $\leq_{\mathcal{P}}$ on $\mathcal{B}$ by: $B_2 \leq_{\mathcal{P}} B_1$ if $\mathcal{R}(B_2)$ appears on the unique $\mathcal{D}$-path $\mathcal{R}(B_1) \dots B_1$.

Definition 6.3 (Induction order). A partial order $\lhd$ on $\mathcal{B}$ is said to be an *induction order* for $\mathcal{P}$ if $\lhd$ is *forest-like*, i.e. $(B \lhd B_1 \wedge B \lhd B_2)$ implies $(B_1 = B_2 \vee B_1 \lhd B_2 \vee B_2 \lhd B_1)$, and every weakly $\leq_{\mathcal{P}}$-connected set $S \subseteq \mathcal{B}$ has a $\lhd$-greatest element.

Definition 6.4 (Trace manifold). Let $\mathcal{P} = (\mathcal{D}, \mathcal{R})$ be a pre-proof in cycle normal form, let $\mathcal{B} = \{B_1, \dots, B_n\}$ be the set of bud nodes occurring in $\mathcal{P}$ and let $\lhd$ be an induction order for $\mathcal{P}$. A *trace manifold* with respect to $\lhd$ is a set of traces: $\{\tau_{ij} \mid B_i, B_j \in \mathcal{B}, B_i \lhd B_j\}$ satisfying:

- τ_{ij} follows the $\mathcal{D}$-path $\mathcal{R}(B_i) \dots B_i$ in S;
- $\tau_{ij}(B_i) = \tau_{ij}(\mathcal{R}(B_i))$;
- $B_j \leq_{\mathcal{P}} B_i \lhd B_k$ implies $\tau_{jk}(\mathcal{R}(B_j)) = \tau_{ik}(\mathcal{R}(B_j))$;
- for each i, τ_{ii} has at least one progress point.

Lemma 6.5. *Let $\mathcal{P}$ be a pre-proof in cycle normal form and let $\lhd$ be an induction order for $\mathcal{P}$. If $\mathcal{P}$ has a trace manifold with respect to $\lhd$, then $\mathcal{P}$ is a cyclic proof.*

Proof. (Sketch) Definition 4.12 can be reformulated as quantified over strongly connected subgraphs of $\mathcal{G}_{\mathcal{P}}$. It can then be shown that, since $\mathcal{P}$ is in cycle normal form, such subgraphs can be characterised as $\mathcal{D}$-paths of the form $\mathcal{R}(B) \dots B$ whose bud endpoints are weakly $\leq_{\mathcal{P}}$-connected. From this one can analyse the composition of any infinite path through $\mathcal{G}_{\mathcal{P}}$ and construct an infinitely progressing trace using the appropriate components given by the trace manifold. □

We remark that, in fact, our translation from structural to cyclic proofs (cf. Theorem 4.15) transforms structural proofs into cyclic proofs with trace manifolds.

7 Conclusions and Future Work

We have formulated a cyclic proof system for first-order logic with ordinary inductive definitions that, importantly, uses only the standard syntax of sequent calculus for first-order logic and the standard sequent calculus proof rules, together with simple unfolding rules for inductively defined predicates. A global condition formulated in terms of *traces* over infinite paths in the proof is used to guarantee soundness. This approach essentially amounts to a postponement in the choice of induction principle; induction principles are not chosen within the proof itself, but rather implicitly via eventual satisfaction of the trace condition.

Cyclic proofs have been demonstrated to subsume the usual structural proofs that use explicit induction rules. Establishing the status of our Conjecture 4.16 — that cyclic proofs are no more powerful than structural proofs in terms of what can be proved — appears very difficult due to the complexity inherent in the trace condition on cyclic proofs and in our definition schema. We have developed tools for analysing the structure of cyclic proofs: in particular, a general theorem allowing cyclic proofs to be transformed via a folding operation on infinite trees. The useful result of cycle-normalisation is a simple corollary of this theorem.

We can also define cyclic proof more locally in terms of *trace manifolds* at the (possible) expense of some generality. We have observed that any structural proof can be transformed into a cyclic proof with a trace manifold. So structural provability $\Rightarrow$ trace manifold provability $\Rightarrow$ cyclic provability. Establishing whether either of these implications hold in reverse is clearly of interest but appears very difficult. The main problem with the latter — transforming an arbitrary cyclic proof into one having a trace manifold — is that as traces on two infinite paths can behave entirely differently, it is not obvious that a manifold need exist. It may be possible to use our "tree-folding" machinery of Section 5 in conjunction with a combinatorial argument about traces to establish this property, and we are looking into this presently. As for the former problem — translating cyclic proofs with trace manifolds into structural proofs — the main difficulty appears to lie in resolving the case-splits that represent trace progress points with cases of explicit induction rules.

Acknowledgements

The author wishes to thank, primarily, his supervisor, Alex Simpson. Thanks are also due to Alan Bundy, Lucas Dixon, Geoff Hamilton, Alberto Momigliano, Alan Smaill, and Rene Vestergaard for fruitful discussions, and to the anonymous referees.

References

1. Peter Aczel. An introduction to inductive definitions. In Jon Barwise, editor, *Handbook of Mathematical Logic*, pages 739–782. North-Holland, 1977.
2. Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development*. EATCS: Texts in Theoretical Computer Science. Springer-Verlag, 2004.
3. Julian Bradfield and Colin Stirling. Local model checking for infinite state spaces. *Theoretical Computer Science*, 96:157–174, 1992.
4. Thierry Coquand. Infinite objects in type theory. In H. Barendregt and T. Nipkow, editors, *Types for Proofs and Programs*, pages 62–78. Springer, 1993.
5. Mads Dam and Dilian Gurov. μ-calculus with explicit points and approximations. *Journal of Logic and Computation*, 12(2):255–269, April 2002.
6. Gerhard Gentzen. Investigations into logical deduction. In M.E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131. North-Holland, 1969.
7. Eduardo Giménez. *A Calculus of Infinite Constructions and its application to the verification of communicating systems*. PhD thesis, Ecole Normale Supérieure de Lyon, 1996.
8. M.J.C. Gordon and T.F. Melham, editors. *Introduction to HOL: a theorem proving environment for higher order logic*. Cambridge University Press, 1993.
9. Geoff Hamilton. Poítin: Distilling theorems from conjectures. To appear.
10. Matt Kaufmann, Panagiotis Manolios, and J Strother Moore. *Computer-Aided Reasoning: An Approach*. Kluwer Academic Publishers, June 2000.
11. Per Martin-Löf. Haupstatz for the intuitionistic theory of iterated inductive definitions. In J.E. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*. North-Holland, 1971.
12. Raymond McDowell and Dale Miller. Cut-elimination for a logic with definitions and induction. *Theoretical Computer Science*, 232:91–119, 2000.
13. Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.
14. Ulrich Schöpp. Formal verification of processes. Master's thesis, University of Edinburgh, 2001.
15. Ulrich Schöpp and Alex Simpson. Verifying temporal properties using explicit approximants: Completeness for context-free processes. In *Foundations of Software Science and Computation Structure: Proceedings of FoSSaCS 2002*, volume 2303 of *Lecture Notes in Computer Science*, pages 372–386. Springer-Verlag, 2002.
16. Carsten Schürmann. *Automating the Meta-Theory of Deductive Systems*. PhD thesis, Carnegie-Mellon University, 2000.
17. Christoph Sprenger and Mads Dam. A note on global induction mechanisms in a μ-calculus with explicit approximations. *Theoretical Informatics and Applications*, July 2003. Full version of FICS '02 paper.
18. Christoph Sprenger and Mads Dam. On the structure of inductive reasoning: circular and tree-shaped proofs in the μ-calculus. In *Proceedings of FOSSACS 2003*, volume 2620 of *Lecture Notes in Computer Science*, pages 425–440, 2003.
19. Valentin Turchin. The concept of a supercompiler. *ACM Transactions on Programming Languages and Systems*, 8:90–121, 1986.
20. M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. *Logic in Computer Science, LICS '86*, pages 322–331, 1986.