Induction and cycles

Quentin Heath
Parsifal, LIX, École polytechnique

April 11, 2014

Contents

Introduction

The first part lays down the matter, *i.e.* the logic and predicates. The second part defines cyclic proofs and gives a theorem about them, while the third and fourth parts give the details of the proof of this theorem. The fifth part compares the presently defined cyclic proofs with similar concepts seen in other work.

1 Landscape

1.1 Logic

We study here an unspecified logic where formulae are built with atomic formulae and a number of connectives. Connectives may be presented at will in n-ary form, or by a binary version and a neutral element. Negation is not considered here a logical

connective; instead, we use $\cdot \supset \bot$ and don't make non-constructive assumptions on the duality of connectives.

We mainly use a standard two-sided presentation of sequent calculus, where hypotheses and conclusions of a sequent are multisets; but when considering a sequent with a side containing a single formula, we use a non-standard notation with square brackets and a sign: $[\Gamma; A]^\epsilon$ where Γ is a multiset, A a formula and $\epsilon \in \{+, -\}$. There, $[\Gamma\,;\, A]^+$ means $\Gamma \vdash A$ and $[\Gamma\,;\, A]^-$ means $A \vdash \Gamma$. We extend this notation to the case where the single formula and the formulae from the multiset have the type `t -> prop` instead of **`prop`**, and we write $[\Gamma\,;\, A]^\epsilon$ for $[\Gamma\, \boldsymbol{x}\,;\, A\, \boldsymbol{x}]^\epsilon$ where $\boldsymbol{x}$ is a fresh eigenvariable with type `t`. The motivation for this notation is that $[\Gamma\,;\, A]^\epsilon$ expresses a simple fact about the truth ($\epsilon = +$) or falsity ($\epsilon = -$) of a formula given some context, notion which is central to the simple setting of this paper.

In addition to the usual atoms (variables, neutral elements, applications in normal form), we first consider the equality connective $\doteq$, and its dual $\neq$. Most of the time, they are notations which shorten formulae and reasoning for free; they are used all over but few mention of them are made.

We also consider as an atom-like formula the application of a defined predicate to an argument (predicates are presented in uncurried form, and defined hereafter). This is not an atom per-se, as we add inference rules for predicates, but these rules are not purely logical nor structural. This, and the need to apply an axiom rule to that kind of formula, justify the use of the name *defined atom*.

1.2 Defined predicates

The approach to a predicate is rather that of fixpoints than that of definitions. Although we use the definition-style notation

$$p\, \boldsymbol{x} \triangleq p\, a \wedge q\, \boldsymbol{x} \qquad\qquad q\, \boldsymbol{x} \triangleq p\, \boldsymbol{x} \wedge q\, b$$

to write examples, it is not well-suited for a specification, as some care is needed in the choice of what we call *flavour* of the definition, *i.e.* whether it is μ (inductive definition) or ν (co-inductive definition). Having mutually recursive predicates with mixed flavours has non-trivial semantics and is beyond the scope of this paper. Assuming some mutually recursive predicates p and q have the same flavour $\delta \in \{\mu, \nu\}$, we can represent them by a single recursive predicate r:

$$r\,(p, \boldsymbol{x}) \overset{\delta}{=} r\,(p, a) \wedge r\,(q, \boldsymbol{x}) \qquad\qquad r\,(q, \boldsymbol{x}) \overset{\delta}{=} r\,(p, \boldsymbol{x}) \wedge r\,(q, b)$$

or even by what we call now a definition, *i.e.* the pair (δ, B) of the flavour and the higher order body of the definition:

$$r\,(\boldsymbol{y}, \boldsymbol{x}) \overset{\delta}{=} B\, r\,(\boldsymbol{y}, \boldsymbol{x})$$

In all of the following, δ represents an unary connective and δB denotes the corresponding defined predicate (defined as either $\mu r.B\, r$ or $\nu r.B\, r$ wrt. the usual meaning of μ and ν). The only equality we use between defined predicates is syntactic equality of both the flavour and the definition body. $\delta B\, \boldsymbol{t}$ is called a defined atom, and $B\,(\delta B)\, \boldsymbol{t}$ is the formula obtained by unfolding δB once in this defined atom. It should be noted that we do not assume $\boldsymbol{t}$ to be ground at that point.

The only stratification constraint is now for the body of a definition to be monotonous. We expect to be able to derive this fact, which means that occurrences of

the higher-order argument must appear positively in the body. Instead of requiring an explicit total order on the definitions, as we would have with a list of predicate declarations, we use a partial well-founded *stratification order* $\sqsubseteq$ such that $\delta_i B_i \sqsubseteq \delta_j B_j$ if the formula B_j involves the predicate $\delta_i B_i$.

1.3 Literals

Although the $\cdot^\epsilon$ notation doesn't stand for a negation connective and defined atoms are not actual atoms, we call $a = [\Gamma\,;\, \delta B\, \boldsymbol{t}]^\epsilon$ a *literal judgement*, or simply *literal*, and $\tilde{a} = [\Gamma\,;\, B\,(\delta B)\,\boldsymbol{t}]^\epsilon$ denotes the corresponding unfolded judgement.

1.4 Notations

The following notations are used throughout this work:

notation	meaning	assumptions
$\doteq, \neq$	object-level (in)equality	unification rules
$[\Gamma\,;\, A]^\epsilon$	$\Gamma\,\boldsymbol{x} \vdash A\,\boldsymbol{x}$ or $A\,\boldsymbol{x} \vdash \Gamma\,\boldsymbol{x}$	x free
$a = [\Gamma\,;\, \delta B\,\boldsymbol{t}]^\epsilon$	literal	$(\delta, \epsilon) \in \{\mu, \nu\} \times \{-, +\}$
$\tilde{a} = [\Gamma\,;\, B\,(\delta B)\,\boldsymbol{t}]^\epsilon$	unfolded literal	
$\delta_i B_i \sqsubseteq \delta_j B_j$	stratification ordering	
$\textsc{ct} = (\mathcal{V}_u, \mathcal{V}_l, u, l)$	cyclic tree	$(\mathcal{V}_u \cup \mathcal{V}_l, u)$ rooted tree
$\textsc{clt} = (\textsc{ct}, (\mathcal{L}, H), L)$	cyclic labelled tree	
$(\textsc{clt}, \Pi)$	cyclic proof	
$((C, \rightrightarrows), (\mathcal{L}, H), L)$	cyclic components tree	$(C, \rightrightarrows)$ built from $\textsc{ct}$
$[\Gamma, S_c\,;\, \delta_c B_c]^{\epsilon_c}$	proof control point	S_c invariant for $\delta_c B_c$

1.5 A simple example

Listing 1: Simple predicate (mutually recursive version)

```
Define
  coinductive a : prop, coinductive b : prop,
  coinductive c : prop, coinductive d : prop,
  coinductive e : prop, coinductive f : prop,
  coinductive g : prop
by a := b /\ e
;  b := c /\ d
;  c := b      % b loops (together with c)
;  d
;  e := f /\ g
;  f := g
;  g := f      % f loops (together with g)
.
```

Listing 1 shows a Bedwyr program which solving will involve co-induction. It will be used throughout this paper to illustrate definitions and theorems.

2 Cyclic proofs

The underlying structure of a cyclic proof is given by a *cyclic labelled tree*, *i.e.* a tree enhanced with back-arcs and labelled with literals. This tree is defined in two steps,

the first of which regard only the nodes, which have no semantics and only show the structure of the proof.

Definition 1 (cyclic tree)**.** *Consider the following objects:*

- *a finite set of nodes $\mathcal{V}$ partitioned into $\mathcal{V}_u$ (unfolding nodes) and $\mathcal{V}_l$ (looping nodes)*
- *an unfolding function $u : \mathcal{V}_u \to 2^{\mathcal{V}}$, such that $(\mathcal{V}, u)$ has a rooted tree structure where all nodes of $\mathcal{V}_l$ are leaves*
- *a looping function $l : \mathcal{V}_l \to \mathcal{V}_u$, giving for every looping node a non-degenerate back-arc: $\forall v \in \mathcal{V}_l$, $l(v) \prec v$, where the strict partial order $\prec$ is the transitive closure of u*

Then $(\mathcal{V}_u, \mathcal{V}_l, u, l)$ is called a cyclic tree structure (Figure 1).

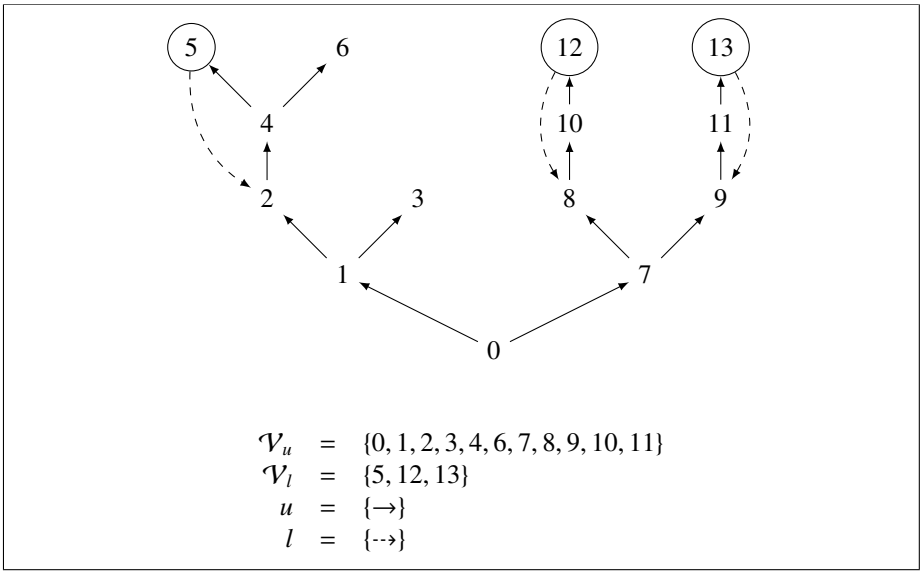


Figure 1: Cyclic tree

This definition imposes that looping nodes are leaves. The next step links this structure to the proof itself.

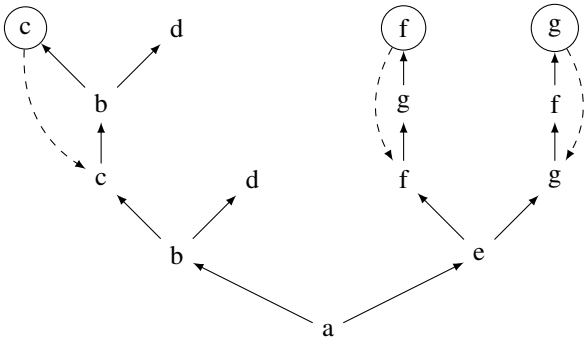
Definition 2 (cyclic labelled tree)**.** *Consider the following objects:*

- *a cyclic tree* $\text{CT} = (\mathcal{V}_u, \mathcal{V}_l, u, l)$
- *a finite set of literals $\mathcal{L}$, and a hypotheses function $H : \mathcal{L} \to 2^{\mathcal{L}}$ which is stratified,* i.e. *such that, if $H([\Gamma\,;\, \delta B\, \mathbf{t}]^{\epsilon}) = \{[\Gamma\,;\, \delta_i B_i\, \mathbf{t}_i]^{\epsilon_i} \mid i\}$:*
  - $\delta_i B_i \sqsubseteq \delta B$
  - $\delta B i = \delta B \Rightarrow \epsilon_i = \epsilon$[1]

[1]This constraint means that the proof of a fact about a predicate can't depend on the refutation of another fact about the same predicate, *et vice versa*. This is expected if the predicate is defined by a monotonous function. As a consequence, all literals of a dependency cycle have the same sign ϵ (and, obviously, the same flavour δ and definition body B).

- *a surjective labelling function* $L : \mathcal{V} \to \mathcal{L}$ *such that:*
  - *the sons of an unfolding node are labelled with the hypotheses of its label:* $\forall v \in \mathcal{V}_u,\ \{L(w) \mid w \in u(v)\} = H(L(v))$
  - *a back-arc points back to an ancestor with the same label:* $\forall v \in \mathcal{V}_l,\ L(l(v)) = L(v)$
  - *the label* $[\Gamma\,;\, \delta B\, \mathbf{t}]^{\epsilon}$ *of a looping node is copacetic:* $(\delta, \epsilon) \in \{(\mu, -), (\nu, +)\}$

Then (CT, $(\mathcal{L}, H), L$) *is called a cyclic labelled tree structure (Figure 2).*



$$\begin{array}{rcl} \mathcal{L} & = & \{a, b, c, d, e, f, g\} \\ H & = & \{(a \mapsto \{b, e\}), (b \mapsto \{c, d\}), (c \mapsto \{b\}), \\ & & (d \mapsto \varnothing), (e \mapsto \{f, g\}), (f \mapsto \{g\}), (g \mapsto \{f\})\} \\ L & = & \{0 \mapsto a, 1 \mapsto b, 7 \mapsto e, \ldots\} \end{array}$$

We write a instead of the label $[\,;\, a]^{+}$ for clarity.

Figure 2: Cyclic labelled tree

This definition doesn't ensure existence, and thus implies constraints on the cyclic tree. It also allows for a suboptimal labelling (or rather a suboptimal cyclic tree) in the sense that parts of the proof may be duplicated. This can be avoided to some point without loss of expressivity by some minimality criteria, the first of which is called *cut-free minimality*.

Definition 3 (cut-free-minimal cyclic labelled tree)**.** *Consider the following objects:*

- *a cyclic tree* CT = $(\mathcal{V}_u, \mathcal{V}_l, u, l)$
- *a cyclic labelled tree* CLT = (CT, $(\mathcal{L}, H), L$) *such that:*
  - *an unfolding node has exactly one son for each of its label's hypotheses: for any* $v \in \mathcal{V}_u$*, the restricted of the labelling function* $L_{|u(v)} : u(v) \to H(L(v))$ *is a bijection*
  - *a looping node has exactly one ancestor with the same label, and an unfolding node has none:* $\forall u, v \in \mathcal{V}_u,\ u \prec v \Rightarrow L(u) \neq L(v)$

Then CLT *is cut-free-minimal (Figure 3).*

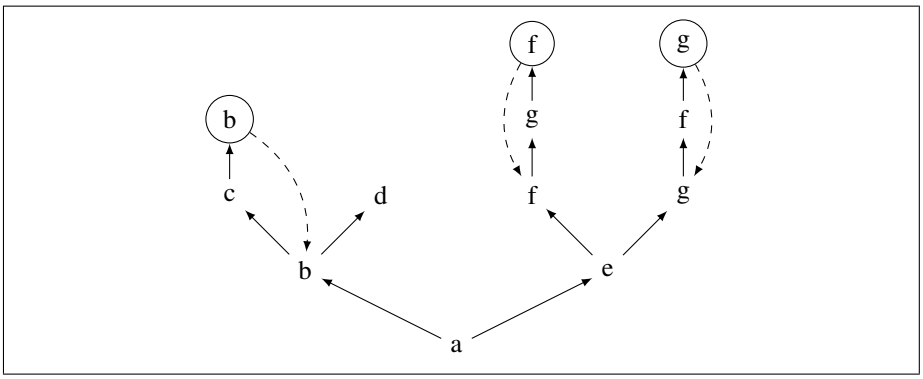


Figure 3: Cut-free minimal cyclic labelled tree

The purpose of this kind of minimality is only to reduce computation length; in all of the following, it will not be assumed[2].

Definition 4 (cyclic proofs)**.** *Consider the following objects:*

- *a cyclic tree* $\textsc{ct} = (\mathcal{V}_u, \mathcal{V}_l, u, l)$
- *a cyclic labelled tree* $\textsc{clt} = (\textsc{ct}, (\mathcal{L}, H), L)$
- *a set of open derivations* $\Pi = \{\Pi_a,\ a \in \mathcal{L}\}$ *such that:*
  - Π_a *is a proof of the unfolded literal* $\tilde{a}$ *with the* $b \in H(a)$ *as assumptions*
  - Π_a *contains no fixpoint rules (neither unfolding nor (co-)induction)*

Then $(\textsc{clt}, \Pi)$ *is a cyclic proof of the literal which labels the root of* clt *(Figure 4).*

Theorem 1. *If there is a cyclic proof of a, there is a traditional proof of a in a system with induction rules.*

Proof. The proof relies on the aggregation of the present nodes into components, each of which is associated with a set (*i.e.* conjunction) of labels. By building a derivation (*synthetic rule*, section 4) for each component and organizing them into a tree (*cyclic components tree*, section 3), we get a proof of the root set of labels, which subsumes the label of the original root. □

[2]Another kind of minimality (which, combined with this one, gives something we call *full minimality*) comes from the use of arcs that point to nodes that occur sooner in the depth-first traversal, but are not ancestors. Although the representation and, in our case, the implementation, are similar, this has more to do with regular cuts than with (co-)induction, and is not treated here.

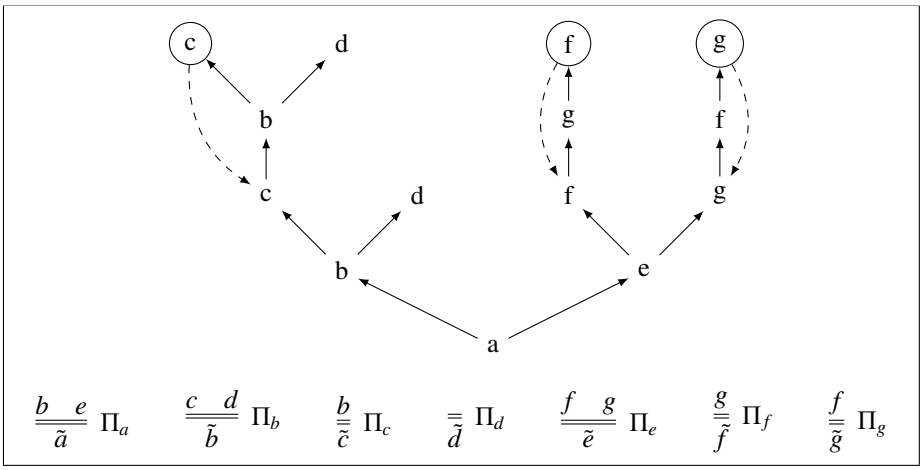


Figure 4: Cyclic proof (of a)

3 Cyclic components tree

We introduce a new intermediate tree designed to discriminate between the unfolding and the looping looping parts of $(\mathcal{V}_u \cup \mathcal{V}_l, u)$. The new nodes are *cyclic components* of the old previous tree, *i.e.* sets of nodes which contain cyclic behaviours.

Definition 5 (cyclic components tree)**.** *Consider the following objects:*

- *a cyclic tree* $\text{CT} = (\mathcal{V}_u, \mathcal{V}_l, u, l)$
- *a cyclic labelled tree* $\text{CLT} = (\text{CT}, (\mathcal{L}, H), L)$
- *a binary relation* $\twoheadrightarrow$ *on* $\mathcal{V} = \mathcal{V}_u \cup \mathcal{V}_l$ *defined as* $u \cup l = \{(v, w) \mid w \in u(v)\} \cup \{(v, l(v))\}$
- *a partition* $C : \mathcal{V} \to 2^{\mathcal{V}}$ *of* $(\mathcal{V}, \twoheadrightarrow)$ *in connected components, which is*
  - *finer than stratification: if* $C(v_1) = C(v_2)$ *and* $L(v_i) = [\Gamma\,;\, \delta_i B_i\, \boldsymbol{t}_i]^{\epsilon_i}$*, then* $(\delta_1 B_1, \epsilon_1) = (\delta_2 B_2, \epsilon_2)$[3]
  - *coarser than the partition into strongly connected components of* $(\mathcal{V}, \twoheadrightarrow)$ [4]
- *a binary relation* $\rightrightarrows$ *defined on* $C = \{C(v) \mid v \in \mathcal{V}\}$*, adapted from* $\twoheadrightarrow$ *such that* $(C, \rightrightarrows)$ *is a tree of subsets of* $\mathcal{V}$

Then $((C, \rightrightarrows), (\mathcal{L}, H), L)$ *is a cyclic components tree associated with* CLT *(Figure 5).*

The existence of such a tree follows that of the partition C, which itself comes from the fact that strongly connected components are themselves *finer than stratification*, which in turn follows from the constraints on the hypotheses function H.

[3] As a consequence, if $v \in c$, we can write $L(v) = [\Gamma\,;\, \delta_c B_c\, \boldsymbol{t}_v]^{\epsilon_c}$ instead of $L(v) = [\Gamma\,;\, \delta_v B_v\, \boldsymbol{t}_v]^{\epsilon_v}$.

[4] At most, the partition is that of the strongly connected components, and the invariants are minimal in size and maximal in number; at least, it is the strata partition, and there is a single maximal invariant for each predicate. The purpose of this flexibility is to allow for an external source to provide a partition by whatever means it chooses (heuristic, actual computation of the strongly connected components, or no computation at all).

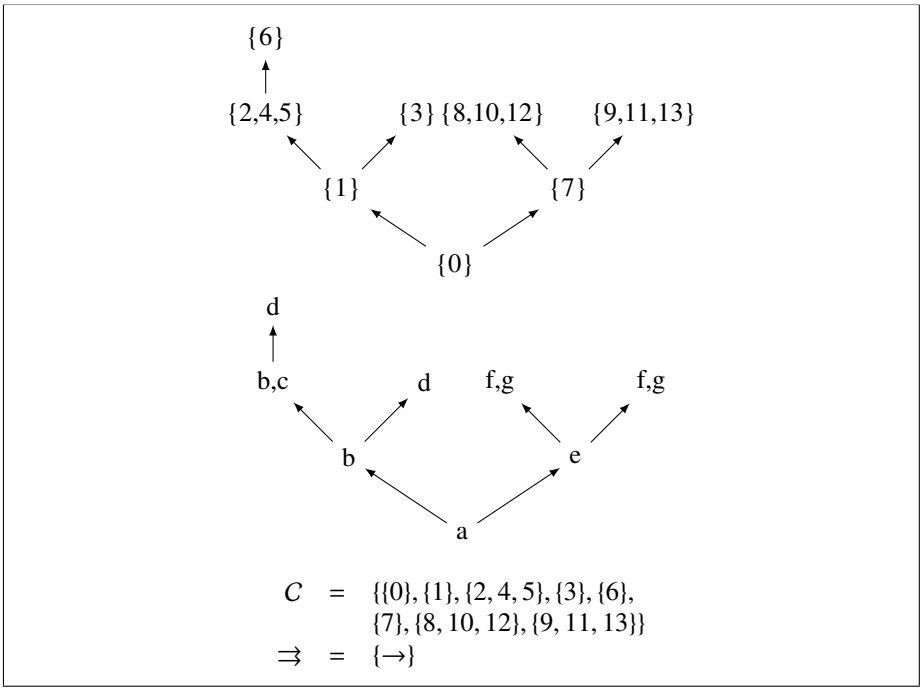


Figure 5: Cyclic components tree, with the corresponding tree of sets of labels

Each component c is associated with the set of labels $\{L(v) \mid v \in c\}$, with $L(v) = [\Gamma\,;\, \delta_c B_c\, \boldsymbol{t}_v]^{\epsilon_c}$; the goal is to prove all these labels at once. As only the argument $\boldsymbol{t}_v$ discriminates them, we can represent c by $[\Gamma, S_c\, \boldsymbol{x}\,;\, \delta_c B_c\, \boldsymbol{x}]^{\epsilon_c}$, *i.e.* $[\Gamma, S_c\,;\, \delta_c B_c]^{\epsilon_c}$, where S_c is a characteristic predicate of the set of arguments $\{\boldsymbol{t}_v \mid v \in c\}$:

$$S_c = \begin{cases} \lambda \boldsymbol{x}.\, \bigwedge_{v \in c} (\boldsymbol{x} \neq \boldsymbol{t}_v) & \epsilon_c = - \\ \lambda \boldsymbol{x}.\, \bigvee_{v \in c} (\boldsymbol{x} \doteq \boldsymbol{t}_v) & \epsilon_c = + \end{cases}$$

In section 4, S_c is used as invariant for a (co-)induction rule applied on $[\Gamma, S_c\,;\, \delta_c B_c]^{\epsilon_c}$.

It is then easy to derive the following rules (the ellipsis indicate that v takes all the values from c):

$$\frac{}{[\Gamma, S_c\,;\, S_c]^{\epsilon_c}}\ C_1 \qquad \frac{[\Gamma\,;\, P\, \boldsymbol{t}_v]^{\epsilon_c} \quad \cdots}{[\Gamma, S_c\,;\, P]^{\epsilon_c}}\ C_2 \qquad \frac{[\Gamma, S_c\,;\, P]^{\epsilon_c}}{[\Gamma\,;\, P\, \boldsymbol{t}_v]^{\epsilon_c}}\ C_3$$

The sequent $[\Gamma, S_c\,;\, P]^{\epsilon_c}$ translates as "the predicate P has the expected behaviour of the predicate $\delta_c B_c$ on the component c". Therefore the $[\Gamma, S_c\,;\, \delta_c B_c]^{\epsilon_c}$ sequents can be seen as major control points of the proof, and label the components. Assume that, for each component c, we have a synthetic rule $\pi(c)$ which is a proof of $[\Gamma, S_c\,;\, \delta_c B_c]^{\epsilon_c}$ using the assumptions $[\Gamma, S_d\,;\, \delta_d B_d]^{\epsilon_d}$ for each d such as $c \rightrightarrows d$:

$$\frac{[\Gamma, S_d\,;\, \delta_d B_d]^{\epsilon_d} \quad \cdots}{[\Gamma, S_c\,;\, \delta_c B_c]^{\epsilon_c}}\ \pi(c)$$

Then by combining these synthetic rules we obtain a closed derivation, which synthetic structure is that of the tree $(C, \rightrightarrows)$ (Figure 6), and which is a proof of $[\Gamma, S_r\,;\, \delta_r B_r]^{\epsilon_r}$, where r is the root of $(C, \rightrightarrows)$. From this we derive $[\Gamma\,;\, \delta_r B_r\, \boldsymbol{t}_v]^{\epsilon_r}$, *i.e.* $L(v)$, for any

element v of r, including the root of $\mathcal{V}$ itself. Therefore the existence of the $\pi(c)$ proves Theorem 1.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\{d\}}\,\pi(\{6\})}{\{b,c\}}\,\pi(\{2,4,5\}) \quad \dfrac{}{\{d\}}\,\pi(\{3\})}{\{b\}}\,\pi(\{1\}) \quad \dfrac{\dfrac{}{\{f,g\}}\,\pi(\{8,10,12\}) \quad \dfrac{}{\{f,g\}}\,\pi(\{9,11,13\})}{\{e\}}\,\pi(\{7\})}{\{a\}}\,\pi(\{0\})$$

We write $\{L(v) \mid v \in c\}$ instead of $[\Gamma, S_c\,;\, \delta_c B_c]^{\epsilon_c}$ for clarity.

Figure 6: Synthetic proof (of $[\Gamma, S_{\{0\}}\,;\, \delta_{\{0\}} B_{\{0\}}]^{\epsilon_{\{0\}}}$, *i.e.* $[\,;\, a]^+$)

4 Synthetic rules

In order to complete the proof, we now have to use the provided derivations

$$\dfrac{\dfrac{b \quad \cdots}{}}{\tilde{a}}\,\Pi_a,\ b \in H(a)$$

to build the synthetic rules $\pi(c)$:

$$\dfrac{[\Gamma, S_d\,;\, \delta_d B_d]^{\epsilon_d} \quad \cdots}{[\Gamma, S_c\,;\, \delta_c B_c]^{\epsilon_c}}\,\pi(c),\ c \rightrightarrows d$$

If $a = [\Gamma\,;\, \delta_c B_c\, \boldsymbol{t}_v]^{\epsilon_c}$ ($c = C(v)$ being the component containing v), then Π_a is a proof of $\tilde{a} = [\Gamma\,;\, B_c\,(\delta_c B_c)\, \boldsymbol{t}_v]^{\epsilon_c}$ with the set of hypotheses $\{[\Gamma\,;\, \delta_{C(w)} B_{C(w)}\, \boldsymbol{t}_w]^{\epsilon_{C(w)}}\}$, where w takes some values in the set of sons of the node v in the cyclic labelled tree, *i.e.* $w \in u(v)$. This means that either $C(v) = C(w)$, or $C(v) \rightrightarrows C(w)$. We have three kind of hypotheses:

$C(w) = c$**:** the hypothesis is part of the same component c (*e.g.* both labels are part of the same loop)

$C(w) = d \neq c$ **but** $\delta_c B_c = \delta_d B_d$**:** the hypothesis is part of another component d (such as $c \rightrightarrows d$) which relates to the same predicate, and $\epsilon_c = \epsilon_d$

$C(w) = e \neq c$ **and** $\delta_c B_c \neq \delta_e B_e$**:** the hypothesis is part of another component e (such as $c \rightrightarrows e$) which relates to another predicate

In either case, the hypothesis $[\Gamma\,;\, \delta_{C(w)} B_{C(w)}\, \boldsymbol{t}_w]^{\epsilon_{C(w)}}$ follows from $[\Gamma, S_{C(w)}\,;\, \delta_{C(w)} B_{C(w)}]^{\epsilon_{C(w)}}$. From this we can build the derivation

$$\dfrac{[\Gamma, S_c\,;\, \delta_c B_c]^{\epsilon_c} \quad [\Gamma, S_d\,;\, \delta_d B_d]^{\epsilon_d} \quad \cdots \quad [\Gamma, S_e\,;\, \delta_e B_e]^{\epsilon_e} \quad \cdots}{[\Gamma\,;\, B_c\,(\delta_c B_c)\, \boldsymbol{t}_v]^{\epsilon_c}}\,\Pi^v(\delta_c B_c)$$

where d takes all the values such that $c \rightrightarrows d$ and $\delta_c B_c = \delta_d B_d$, and e takes all the values such that $c \rightrightarrows e$ and $\delta_c B_c \sqsupset \delta_e B_e$.

Moreover, $\Pi^v(\delta_c B_c)$, as Π_a, doesn't contain unfolding or (co-)induction rules, so none of its rules actually depend on $\delta_c B_c$, and we really have the derivation

$$\dfrac{[\Gamma, S_c\,;\, P]^{\epsilon_c} \quad [\Gamma, S_d\,;\, P]^{\epsilon_d} \quad \cdots \quad [\Gamma, S_e\,;\, \delta_e B_e]^{\epsilon_e} \quad \cdots}{[\Gamma\,;\, B_c\, P\, \boldsymbol{t}_v]^{\epsilon_c}}\,\Pi^v(P)$$

for any predicate P.

We will now proceed with two sets of simplified rules that can be substituted for the general $\pi(c)$ in some cases, before merging them to show the general case itself.

4.1 Unfolding

The simplest case is pure unfolding with no (co-)induction rule; this happens when there is no dependency within the component, *i.e.* there are no two elements $v, w \in c$ such that $w \in u(v)$. It implies that the assumptions of $\Pi^v(P)$ are really only the $[\Gamma, S_d \,;\, P]^{\epsilon_d}$ and the $[\Gamma, S_e \,;\, \delta_e B_e]^{\epsilon_e}$ (and in the case of the strongly connected components partition, it also implies that the component c is a singleton).

This case can involve a least fixpoint on the right ($(\delta, \epsilon) = (\mu, +)$) or a greatest fixpoint on the left ($(\delta, \epsilon) = (\nu, -)$).

An invariant would be of no use here; Π^v already contains most of the necessary rules, and we only need to add the initial unfolding. We call $\epsilon\delta'(B)$ the unfolding on the ϵ side of the fixpoint δB, and $\pi(c)$ is:

$$\dfrac{\dfrac{\dfrac{[\Gamma, S_d \,;\, \delta_d B_d]^{\epsilon_d} \quad \cdots \quad [\Gamma, S_e \,;\, \delta_e B_e]^{\epsilon_e} \quad \cdots}{[\Gamma \,;\, B_c\,(\delta_c B_c)\,\boldsymbol{t}_v]^{\epsilon_c}}\ \Pi^v(\delta_c B_c)}{[\Gamma, S_c \,;\, B_c\,(\delta_c B_c)]^{\epsilon_c}}\ \cdots}{[\Gamma, S_c \,;\, \delta_c B_c]^{\epsilon_c}}\ \epsilon_c\delta'_c(B_c)\ C_2$$

where the ellipses indicate that:

- d takes all the values such that $c \rightrightarrows d$ and $\delta_c B_c = \delta_d B_d$
- e takes all the values such that $c \rightrightarrows e$ and $\delta_c B_c \sqsupset \delta_e B_e$
- v takes all the values from c

Figure 7 show an example of use of this rule on the cyclic components tree from Figure 5.

$$\dfrac{\dfrac{\dfrac{[\Gamma, S_{\{1\}} \,;\, \delta_{\{1\}} B_{\{1\}}]^+ \quad [\Gamma, S_{\{7\}} \,;\, \delta_{\{7\}} B_{\{7\}}]^+}{[\Gamma \,;\, B_c\,(\delta_c B_c)\,\boldsymbol{t}_0]^+}\ \Pi^0(\delta_c B_c)}{[\Gamma, S_c \,;\, B_c\,(\delta_c B_c)]^+}\ C_2}{[\Gamma, S_c \,;\, \delta_c B_c]^+}\ +\nu'(B_c) \qquad \dfrac{\dfrac{\dfrac{b \quad e}{b \wedge e}\ \Pi^0(\delta_c B_c)}{(b \wedge e)}\ C_2}{(a)}\ \epsilon_c\delta'_c(B_c)$$

Figure 7: Unfolding for $c = \{0\}$ (synthetic rule and informal interpretation)

4.2 Looping

The second simplest case is pure looping; this happens when there is no dependency on other components, *i.e.* there is no element $v \in c$ such that $u(v) \notin c$. It implies that the only assumption of $\Pi^v(P)$ is $[\Gamma, S_c \,;\, P]^{\epsilon_c}$ (and that the component c is a leaf of $(C, \rightrightarrows)$, *i.e.* there is no d such that $c \rightrightarrows d$).

We assume that there are some actual dependencies within the component, otherwise this is covered by the unfolding-only synthetic rule; therefore $(\delta, \epsilon) \in \{(\mu, -), (\nu, +)\}$.

Π^v now needs ot be paired with a proper induction (resp. co-induction) rule to handle the loops, and since no arc goes out of the component, S_c itself can be used

as invariant. We call $\delta(B, X)$ the induction or co-induction on the fixpoint δB with the invariant X, and $\pi(c)$ is:

$$\dfrac{\dfrac{}{[\Gamma, S_c\,;\, S_c]^{\epsilon_c}}\,C_1 \qquad \dfrac{\dfrac{\dfrac{}{[\Gamma, S_c\,;\, S_c]^{\epsilon_c}}\,C_1}{[\Gamma\,;\, B_c\,S_c\,\boldsymbol{t}_v]^{\epsilon_c}}\,\Pi^v(S_c) \quad \cdots}{[\Gamma, S_c\,;\, B_c\,S_c]^{\epsilon_c}}\,C_2}{[\Gamma, S_c\,;\, \delta_c B_c]^{\epsilon_c}}\,\delta_c(B_c, S_c)$$

where the ellipsis indicates that v takes all the values from c. Figure 8 show an example of use of this rule on the cyclic components tree from Figure 5.

$$\dfrac{\dfrac{}{[\Gamma, S_c\,;\, S_c]^{+}}\,C_1 \qquad \dfrac{\dfrac{\dfrac{}{[\Gamma, S_c\,;\, S_c]^{+}}\,C_1}{[\Gamma\,;\, B_c\,S_c\,\boldsymbol{t}_8]^{+}}\,\Pi^8(S_c) \quad \dfrac{\dfrac{}{[\Gamma, S_c\,;\, S_c]^{+}}\,C_1}{[\Gamma\,;\, B_c\,S_c\,\boldsymbol{t}_{10}]^{+}}\,\Pi^{10}(S_c) \quad \dfrac{\dfrac{}{[\Gamma, S_c\,;\, S_c]^{+}}\,C_1}{[\Gamma\,;\, B_c\,S_c\,\boldsymbol{t}_{12}]^{+}}\,\Pi^{12}(S_c)}{[\Gamma, S_c\,;\, B_c\,S_c]^{+}}\,C_2}{[\Gamma, S_c\,;\, \delta_c B_c]^{+}}\,\nu(B_c, S_c)$$

$$\dfrac{\dfrac{}{f, g \vdash f, g}\,C_1 \qquad \dfrac{\dfrac{\dfrac{}{f, g \vdash g}\,C_1}{f, g \vdash g}\,\Pi^8(S_c) \quad \dfrac{\dfrac{}{f, g \vdash f}\,C_1}{f, g \vdash f}\,\Pi^{10}(S_c) \quad \dfrac{\dfrac{}{f, g \vdash g}\,C_1}{f, g \vdash g}\,\Pi^{12}(S_c)}{(g) \wedge (f) \wedge (g)}\,C_2}{(f) \wedge (g) \wedge (f)}\,\nu(B_c, S_c)$$

Figure 8: Looping for $c = \{8, 10, 12\}$ (synthetic rule and informal interpretation)

4.3 General case

In the general case, looping and unfolding happen simultaneously. We still need a (co-)induction rule, and its invariant still needs to be S_c in order for Π^v to be used for all $v \in c$ and no other v. Unfortunately, this invariant doesn't satisfy the $[\Gamma, S_d\,;\, \cdot]^{\epsilon_d}$ assumptions (the $[\Gamma, S_e\,;\, \cdot]^{\epsilon_e}$ are not problematic as the invariant doesn't occur in them), therefore we have to use the (co-)induction on another predicate than $\delta_c B_c$.

We still have the restriction that $(\delta, \epsilon) \in \{(\mu, -), (\nu, +)\}$; we only write about the $(\nu, +)$ case as the other one is dual.

To build the auxiliary predicate $\delta_c A_c$, we "inflate" the definition body B_c so that the derivation Π^v ends with a predicate that satisfies both $[\Gamma, S_c\,;\, \cdot]^{\epsilon_c}$ and the $[\Gamma, S_d\,;\, \cdot]^{\epsilon_d}$. This suggests an aggregation of previous components:

$$A_c\,P = B_c\,(P \vee U_c) \qquad\qquad U_c = \bigvee_{c \rightrightarrows d,\ \delta_c B_c = \delta_d B_d} S_d$$

so that, as $\epsilon_d = \epsilon_c = +$, the following hold:

$$A_c\,S_c = B_c\,(S_c \vee U_c) \qquad\qquad \dfrac{[\Gamma, S_d\,;\, S_d]^{\epsilon_d}}{[\Gamma, S_d\,;\, U_c]^{\epsilon_d}}\,\vee_R$$

The way it is built, the new predicate $\delta_c A_c$ is expected to be equal to $\delta_c B_c$ under the

assumptions $[\Gamma, S_d\,;\,\delta_d B_d]^{\epsilon_d}$. We adapt the previous $\pi(c)$ derivations to it:

$$\dfrac{\dfrac{}{[\Gamma, S_c\,;\,S_c]^{\epsilon_c}}\,C_1}{[\Gamma, S_c\,;\,\delta_c A_c]^{\epsilon_c}}\,\dfrac{\dfrac{\dfrac{}{[\Gamma, S_c\,;\,S_c]^{\epsilon_c}}\,C_1}{[\Gamma, S_c\,;\,S_c \vee U_c]^{\epsilon_c}}\,\vee_R \quad \dfrac{\dfrac{}{[\Gamma, S_d\,;\,S_d]^{\epsilon_d}}\,C_1}{[\Gamma, S_d\,;\,S_c \vee U_c]^{\epsilon_d}}\,\vee_R \quad \cdots \quad [\Gamma, S_e\,;\,\delta_e B_e]^{\epsilon_e} \quad \cdots}{[\Gamma\,;\,A_c\,S_c\,\boldsymbol{t}_v]^{\epsilon_c}}\,\Pi^v(S_c \vee U_c) \quad \cdots}{[\Gamma, S_c\,;\,A_c\,S_c]^{\epsilon_c}}\,\delta_c(A_c, S_c)\,C_2$$

where only the $[\Gamma, S_e\,;\,\delta_e B_e]^{\epsilon_e}$ assumptions occur, and where the ellipses indicate that:

- d takes all the values such that $c \rightrightarrows d$ and $\delta_c B_c = \delta_d B_d$
- e takes all the values such that $c \rightrightarrows e$ and $\delta_c B_c \sqsupset \delta_e B_e$
- v takes all the values from c

To complete $\pi(c)$, we also need the derivation

$$\dfrac{[\Gamma, S_c\,;\,\delta_c A_c]^{\epsilon_c} \qquad \begin{array}{c}[\Gamma, S_d\,;\,\delta_d B_d]^{\epsilon_d} \quad \cdots \\ \vdots \end{array}}{[\Gamma, S_c\,;\,\delta_c B_c]^{\epsilon_c}}$$

which we expect to contain the remaining assumptions $[\Gamma, S_d\,;\,\delta_d B_d]^{\epsilon_d}$, but not to depend on anything but the definition of $\delta_c A_c$ with respect to $\delta_c B_c$. This is just bookkeeping (but not necessarily short, as it involves the proof of monotonicity of B_c):

$$\dfrac{[\Gamma, S_c\,;\,\delta_c A_c]^{\epsilon_c} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{[\Gamma, \delta_c A_c\,;\,\delta_c A_c]^{\epsilon_c}}\,I \qquad \dfrac{[\Gamma, S_d\,;\,\delta_d B_d]^{\epsilon_d} \quad \cdots}{[\Gamma, U_c\,;\,\delta_c B_c]^{\epsilon_c}}\,\vee_L}{[\Gamma, U_c\,;\,\delta_c A_c]^{\epsilon_c}}\,\vee_L}{[\Gamma, \delta_c A_c \vee U_c\,;\,\delta_c A_c]^{\epsilon_c}}\,B_{c\nearrow}}{[\Gamma, A_c\,\delta_c A_c\,;\,B_c\,\delta_c A_c]^{\epsilon_c}}\,-\epsilon_c\delta'_c(A_c)}{[\Gamma, \delta_c A_c\,;\,B_c\,\delta_c A_c]^{\epsilon_c}}\,\delta_c(B_c, \delta_c A_c)$$

$$\dfrac{\dfrac{\dfrac{}{[\Gamma, \delta_c B_c\,;\,\delta_c B_c]^{\epsilon_c}}\,I}{[\Gamma, \delta_c B_c\,;\,\delta_c B_c \vee U_c]^{\epsilon_c}}\,\vee_R}{[\Gamma, B_c\,\delta_c B_c\,;\,A_c\,\delta_c B_c]^{\epsilon_c}}\,B_{c\nearrow}}{[\Gamma, \delta_c B_c\,;\,A_c\,\delta_c B_c]^{\epsilon_c}}\,-\epsilon_c\delta'_c(B_c)\,\delta_c(A_c, \delta_c B_c)$$

where the ellipses indicate that d takes all the values such that $c \rightrightarrows d$ and $\delta_c B_c = \delta_d B_d$.

For the dual $(\mu, -)$ case, we only have to replace disjunctions by conjunctions in this derivation and in the definition of U_c and A_c.

This completes the construction of the synthetic rules in the general case, and thus the translation from a cyclic proof with no (co-)induction to a traditional proof with (co-)induction, therefore proving Theorem 1.

5 Comparisons

5.1 Tabling calculus

Work on cyclic proofs as they appear in Bedwyr was already presented in [Hea14], which introduces both *tabling calculus* and a tentative invariant justifying it.

In tabling calculus, the structure of the proof isn't explicitly laid down as a tree $(\mathcal{V}, u)$. Instead, three kinds of literals occurrences are distinguished, corresponding to labels of different kinds of nodes in the new approach: *root occurrences* (unfolding nodes), *looping occurrences* (looping nodes), and *secondary occurrences* (unused in the present work). The proofs contain local information about these occurrences, split in three kinds of contexts, and therefore don't need an additional run through a tree to recover it. These contexts correspond to the three parts of the derivation with respect to the position of the current sequent, giving the system a zipper-like behaviour:

input table: tabling rules occurring before in a postfix traversal

local table: (co-)induction rules occurring as ancestors

output table: tabling rules occurring as descendants

Although it was not formally proved, we believe that there is a direct correspondence between this presentation and a *full-minimal cyclic proof* (Figure 9).

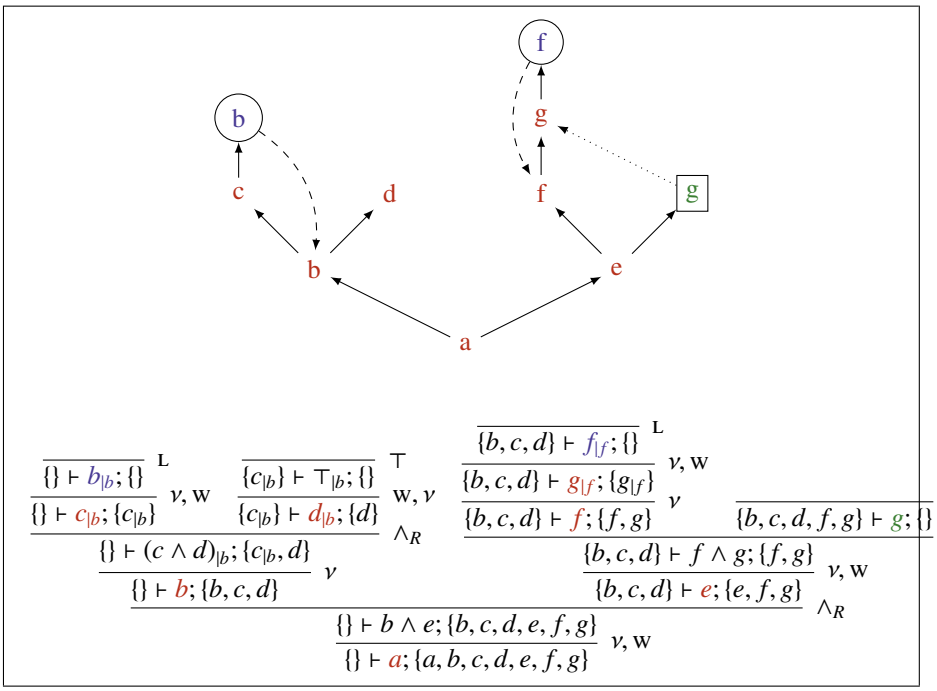


Figure 9: Full minimal cyclic labelled tree and the corresponding tabling calculus derivation (root occurrences, looping occurrences, secondary occurrences)

The invariant is built so as to mimic the computation of Bedwyr; it can't handle loops from multiple atoms, and its size depends on the size of the loops. On the other hand, the present work only uses structural information resulting from the computation to build the invariant, and thus succeeds in a broader range of situations, while its size only depends on the number of different atoms.

Although the memoization aspect of tabling is fully considered in tabling calculus, and in a simple manner, it is not included in the general definition of cyclic proofs anymore, as it can be seen as orthogonal to the invariant construction. Work to bring

it back as part of the concept of *full-minimality* is ongoing as it can allow a significant decrease in proof size.

5.2 Brotherston's cyclic proofs

Another formalization of induction via cyclic arcs added to regular proofs was presented in [Bro05].

References

[Bro05] James Brotherston. "Cyclic Proofs for First-Order Logic with Inductive Definitions". In: *Proceedings of TABLEAUX-14*. Vol. 3702. LNAI. Springer-Verlag, 2005, pp. 78–92.

[Hea14] Quentin Heath. *Partially justifying an implementation of tabling in an extension of the propositional Horn fragment of Bedwyr v1.4*. Feb. 2014. URL: `http://slimmer.gforge.inria.fr/bedwyr/doc/bedwyr14tabling.pdf` (visited on 03/16/2014).