Mechanical certification of FOLID cyclic proofs

Sorin Stratulat

Mechanical Certification of FOL_{ID} Cyclic Proofs

Sorin Stratulat[1*]

[1*]Université de Lorraine, CNRS, LORIA, Metz, F-57000, France.

Corresponding author(s). E-mail(s):
sorin.stratulat@univ-lorraine.fr;

Abstract

Cyclic induction is a powerful reasoning technique that can soundly stop the proof development and make the proof experience successful. In the setting of first-order logic with inductive definitions (FOL_{ID}), cyclic proofs can be built automatically by the Cyclist prover but their implementations are error-prone and the human validation may be tedious. On the other hand, cyclic induction is not yet integrated in certifying proof environments that support first-order logic and inductive definitions, as Isabelle and Coq.

We propose a solution to check in Coq the cyclic proofs produced by E-Cyclist, an extension of Cyclist that implements a more efficient soundness validation method, by using the very general Noetherian induction principle integrated in Coq. Our work is based on a methodology already used successfully for certifying formula-based Noetherian induction proofs. The advantages of our approach are threefold:

- I) The certification of cyclic FOL_{ID} proofs is *mechanical*. The user can validate with Coq every single step from the the E-Cyclist proofs, as well as the induction arguments, and identify errors in a very precise way.
- II) The Coq users can also perform cyclic induction *directly*. Coq functions are provided to help them to deal with the induction part.
- III) There is a great potential for *automation*. The methodology has already been used to automatically convert implicit induction proofs, as those built by the SPIKE prover, to Coq scripts.

Keywords: automated reasoning, cyclic induction, first-order logic with inductive definitions, proof certification, Coq, E-Cyclist

1 Introduction

In the setting of first-order logic with inductive definitions (FOL_{ID}), *cyclic induction* reasoning can be used as a sound argument for stopping the expansion of some proof nodes during the proof process. CLKID$^\omega$ [3, 4] is the standard inference system able to perform it. It extends the sequent-based LK proof system [8] with rules that process equalities and inductive predicates. It can build finite tree derivations called *cyclic pre-proofs* that may have special leaves called *buds*, i.e., nodes that are labelled by sequents already occurring in the derivation. Hence, it it possible to associate to each bud a unique node labelled by the same sequent, called *companion*. The bud-companion relations may introduce *cycles*, leading to *infinite* paths in the pre-proofs.

Cyclist [5] is the *de facto* theorem prover that implements CLKID$^\omega$ to automatize the FOL_{ID} cyclic reasoning. When displayed to the user, a pre-proof is represented in an indented text form. Fig. 1 partially illustrates a proof state built for some sequent `N(x),N(y) ⊢ Q(x,y)`. The nodes are prefixed by naturals that increment in chronological order, starting by 0 for the root node. On the right-hand side of each sequent is displayed additional information related to the inference rule applied on the sequent, as well as the list of the naturals of the nodes labelled by the sequents built by the application of the rule, if any. An exception is done for the `Backl` rule that applies on a bud sequent. In this case, the list has only the companion node. To highlight the bud-companion relation from the figure, we employed a red arrow that starts from the bud node 13 and points to its companion node 0. It introduces a cycle that can be visited by infinite paths, e.g., $[0, 2, 4, 5, 7, 8, 0, 2, ...]$.

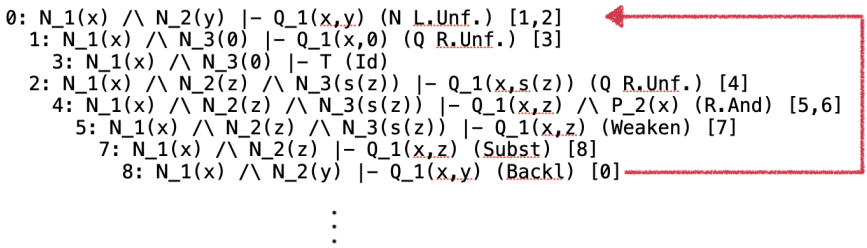

Fig. 1: Indented representation of a derivation tree in Cyclist.

The inductive atoms from each sequent are indexed in order to track their evolution in a derivation tree. The inductive antecedent atoms (IAAs), found on the left-hand side of the '|-' sign of a sequent, are crucial for checking the soundness of a pre-proof. It is a non-trivial process that consists in verifying some *global trace condition*. The idea is to associate a *trace* to every infinite path p from a cyclic pre-proof, built from IAAs occurring in the sequents labelling the nodes from p, then show that *progress* steps happen infinitely

often. The standard way to check the global trace condition is by using a complete procedure, based on an exponential complement operation for Büchi automata [9], that was implemented in Cyclist. Later [11, 13], a polynomial-time but incomplete ordering-based procedure has been proposed for checking cyclic pre-proofs built for a variant of CLKID$^\omega$ that was also implemented in an extension of Cyclist, referred to as E-Cyclist [15]. Even if the theoretical results underlying the two checking procedures seem to be well accepted by the theorem proving community, their implementations are error-prone and the human validation may be tedious. Hence, the need for mechanical solutions for certifying FOL_{ID} cyclic reasoning.

In this paper, we provide a convenient way to mechanically certify E-Cyclist sound pre-proofs, referred to as *proofs*, using the Coq proof assistant [16]. The main idea comes from the observation that Coq can represent sequents as first-order formulas and reproduce the first-order reasoning, define and manipulate inductive predicates, and perform induction reasoning by the means of the Noetherian induction principle. In other words, the logic underlying Coq is enough expressive to reproduce FOL_{ID} specifications and every single pre-proof step. The most technical part consists in converting the soundness arguments used for cyclic induction into Coq script based on Noetherian induction. For this, we consider the ordering-based procedure implemented in E-Cyclist to associate to each critical node n, as the companions involved in cycles, a *measure value* (weight) consisting of a multiset of IAAs from the sequent labelling n. These weights will help Coq to define a *Noetherian (well-founded) induction ordering* to be used in the Coq scripts for showing by Noetherian induction that the measures decrease along the cycles. The way the Noetherian induction arguments are built is given by a methodology [12] developed to certify with Coq formula-based Noetherian induction reasoning, as that presented in the implicit induction proofs produced with the SPIKE theorem prover [14]. In the paper, it helped to successfully certify the E-Cyclist proofs of all the FOL_{ID} examples included in the official distribution of Cyclist, as well as the 2-Hydra problem [2].

Our approach presents two other advantages. Firstly, we give evidence that the Coq users can *directly* perform cyclic induction reasoning. Beyond the existing examples, we provide a set of Coq functions that help them to define the Noetherian induction ordering and to deal with the cyclic induction reasoning. Secondly, there is room for automation; in the past, the methodology was used to completely automatically convert SPIKE proofs to Coq scripts [12].

The rest of the paper has five sections and one appendix with two sections. Section 2 presents as a running example a pre-proof built for one of the non-trivial FOL_{ID} examples taken from the official distribution of Cyclist. Section 3 shows how to convert the pre-proof to Coq script. The machinery for checking whether the pre-proof is a proof and the translation of the induction reasoning part are described in the next two sections. The soundness checking procedure implemented in E-Cyclist and the measure values for the running example are presented in Section 4. The general procedure for

building the Noetherian induction arguments from a cyclic proof, as well as its application on the Cyclist pre-proof, are given in Section 5. The last section concludes and gives directions for future work. The appendix completes the Coq script from Section 3 with the induction reasoning part, as produced with our methodology and provides the missing proofs from Section 4.

2 A case study: the 'P and Q' example

The following example, taken from the official distribution of Cyclist, is known as 'P and Q' [17]. The notations are those used by E-Cyclist to define the inductive predicate symbols N, P and Q over the function symbols 0 and s (the 'successor' function):

$$N\ \{\quad true \Rightarrow N(0)\ |\quad N_1(x) \Rightarrow N(s(x))\ \}\ ;$$

$$P\ \{\quad true \Rightarrow P(0)\ |\quad P_1(x)\ \&\ Q_2(x, s(x)) \Rightarrow P(s(x))\ \}\ ;$$

$$Q\ \{\quad true \Rightarrow Q(x, 0)\ |\quad Q_1(x, y)\ \&\ P_2(x) \Rightarrow Q(x, s(y))\ \}\ ;$$

The predicate $N(x)$ checks whether x is a natural number, while the predicate $P(x)$ and $Q(x, y)$ [18] are mutually defined. E-Cyclist can prove the sequent $N_1(x), N_2(y) \vdash Q_1(x, y)$, saying that $Q(x, y)$ holds whenever x and y are naturals. Here, we have used the interpretation of a sequent in first-order logic, according to which the conjunction of its antecedents implies the disjunction of its *succedents* (the atoms found on the rhs of $\vdash$). Several induction systems succeeded to prove it in different logical settings [17, 3, 12]. Its E-Cyclist pre-proof is given in Fig. 2.

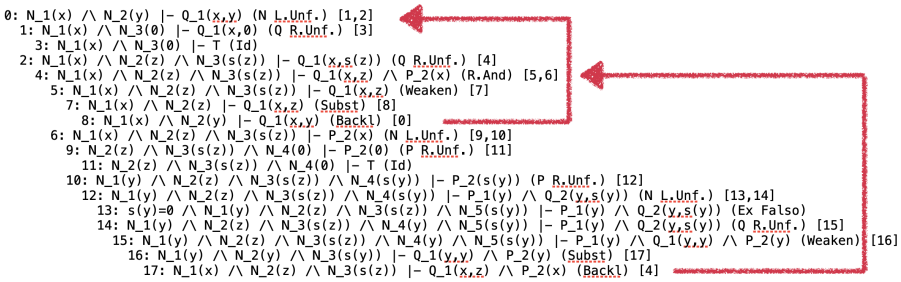

Fig. 2: The E-Cyclist pre-proof of $N_1(x), N_2(y) \vdash Q_1(x, y)$.

The proof of $N_1(x), N_2(y) \vdash Q_1(x, y)$ starts by instantiating y by 0 and $s(z)$ with the rule `(L.Unf.)`, where z is a fresh variable. In FOL_{ID}, the instantiation schema is built from the definition of inductive predicate symbols, in our case N, by instantiating the arguments of one of the IAAs headed by N, in our case $N_2(y)$. For Node 2, $N_2(z)$ was added as IAA from the unfolding of $N_3(s(z))$ using the second axiom defining N. The `(R.Unf.)` rule unfolds the conclusion of the sequent labelling Node 1, i.e., $Q_1(x, 0)$, with the first axiom defining Q to get a sequent that is validated by the `(Id)` rule; E-Cyclist will not try to expand further valid sequents. For Node 2, $Q_1(x, s(z))$ is unfolded by the second axiom defining Q to get a conjunction in the succedent part. After the application of the LK-rule $(\wedge R)$, denoted as `(R.And)`, the weakening LK-rule (Wk), denoted by `(Weaken)`, is applied on the sequent labelling Node 5, then rename Node 7 by `(Subst)` to the root sequent. E-Cyclist considers that Node 8 is a bud and stops its development. For Node 6, `(L.Unf.)` is applied to instantiate the argument x of $N_1(x)$, then the conclusions of the two resulting sequents are unfolded using the definition of P. The sequent labelling Node 11 is true, while that labelling Node 12 has a conjunction as conclusion. E-Cyclist prefers to apply again `(L.Unf.)` to instantiate $s(y)$ given as argument to $N_4(s(y))$. `(Ex Falso)` rule concludes that the sequent labelling Node 13 is true because it has $s(y) = 0$ (a false atom) in the antecedent part. The atom $Q_2(y, s(y))$ from the conclusion of the sequent labelling Node 14 unfolds, then `(Weaken)` is applied before renaming with (Subst) in order to get a bud whose companion is Node 4. The derivation tree is a pre-proof because no other node can be expanded.

Compared with other proof scenarios from [17, 3, 12], the E-Cyclist pre-proof is not the most effective. For example, one could have omitted the last `(L.Unf.)` step which only adds the already present $N(y)$ to the antecedent part of the sequent labelling Node 12.

In the next section, we present the Coq translation of the resulting pre-proof.

3 The Coq script of the pre-proof

The Coq specifications are typed. To cope with the untyped Cyclist specification, we assume that there is a *unique* type variable *T*. It serves as a domain definition for the function symbols denoted by *zero* (for 0) and *succ* (for s), also defined as variables, as well as for the inductive predicates **P** and **Q**:

```
Variable T: Set.
Variable zero: T.
Variable succ: T → T.

Inductive N: T → Prop :=
  n1: N zero
| n2: ∀ x, N x → N (succ x)
```

.

```
Inductive P: T → Prop :=
 p1: P zero
| p2: ∀ x, (P x ∧ Q x (succ x)) → P (succ x)
  with
Q: T → T → Prop :=
 q1: ∀ y, Q y zero
| q2: ∀ x y, (Q x y ∧ P x) → Q x (succ y).
```

The sequent $N_1(x), N_2(y) \vdash Q_1(x, y)$ is translated into Coq as the first-order formula: $\forall\ x\ y,\ \mathbf{N}\ x \to \mathbf{N}\ y \to \mathbf{Q}\ x\ y$ before trying to prove it as the theorem `true_0`. We follow the proof scenario from the previous section by translating every proof step from Fig. 2. The resulting Coq proof of `true_0`, is incomplete since the bud formulas are not proved.

```
Coq < Theorem true_0: forall x y, N x -> N y -> Q x y.
Coq < Proof.
1 goal


============================
forall x y : T, N x -> N y -> Q x y
```

The representation of a Coq proof *goal* recalls that of a sequent with only one succedent. Each goal has a *conclusion* (formula to be proved), and a *context* consisting of named *hypotheses* (formulas), variables and local definitions, both separated by the horizontal bar '==========='. From a logical point of view, the conjunction of the hypotheses implies the conclusion.

By the `intro` tactic, one can discover the main ingredients of the initial sequent from the cyclic proof, where the IAA $N_1(x)$ (resp., $N_2(y)$) is labelled by `H` (resp., `H0`).

```
Coq < intros x y H H0.
1 goal
x, y : T
H : N x
H0 : N y
============================
Q x y
```

The `inversion` tactic is used to simulate the application of `(L.Unf.)`. Its argument is the label of an inductive atom from the context. The name of the fresh variable is given in the square brackets. Here is how the variable `y`

is instantiated:

```
Coq < inversion H0 as [| z].
2 goals

x, y : T
H : N x
H0 : N y
H1 : zero = y
============================
Q x zero
goal 2 is:
Q x (succ z)
```

Two new goals are generated. Notice that Coq shows the full information only for the first goal; for the second, it displays only the conclusion. The first axiom of `Q`, labelled by `q1`, is applied for proving the first goal.

```
Coq < -  apply q1.
1 goal

x, y : T
H : N x
H0 : N y
H1 : zero = y
============================
Q x zero
This subproof is complete, but there are some unfocused goals.
Focus next goal with bullet -.
1 goal
goal 1 is:
Q x (succ z)
```

We use rewriting and the unfolding with `q2` on the conclusion to get a conjunction whose conjuncts are separated in two new goals using the tactic `split`.

```
Coq < - rewrite <- H2 in H0. clear H2 y. apply q2. split.
2 goals

x : T
H : N x
z : T
H0 : N (succ z)
H1 : N z
============================
```

```
Q x z
goal 2 is:
P x
```

The first goal is a bud if the variable z is renamed to y. Its validation is cancelled by the tactic `give_up`.

```
Coq < -- give_up.
1 goal

x : T
H : N x
z : T
H0 : N (succ z)
H1 : N z
============================
Q x z
This subproof is complete, but there are some unfocused goals.
Focus next goal with bullet --.
1 goal
goal 1 is:
P x
```

For the second goal, the variable `x` is instantiated, resulting two new goals.

```
Coq < -- inversion H as [| y].
2 goals

x : T
H : N x
z : T
H0 : N (succ z)
H1 : N z
H2 : zero = x
============================
P zero
goal 2 is:
P (succ y)
```

The first goal is proved using `p1`.

```
Coq < --- apply p1.
1 goal

x : T
H : N x
```

```
z : T
H0 : N (succ z)
H1 : N z
H2 : zero = x
============================
P zero
This subproof is complete, but there are some unfocused goals.
Focus next goal with bullet ---.
1 goal
goal 1 is:
P (succ y)
```

The second goal applies `p2` then `Q y y /\ P y` is firstly assumed as a bud, then used in the proof as a hypothesis.

```
Coq < --- rewrite <- H3 in H. apply p2. assert (Q y y /\ P y).
2 goals

x, y : T
H : N (succ y)
z : T
H0 : N (succ z)
H1 : N z
H2 : N y
H3 : succ y = x
============================
Q y y /\ P y
goal 2 is:
P y /\ Q y (succ y)
```

The development of the first goal is stopped.

```
Coq < + give_up.
1 goal

x, y : T
H : N (succ y)
z : T
H0 : N (succ z)
H1 : N z
H2 : N y
H3 : succ y = x
============================
Q y y /\ P y
This subproof is complete, but there are some unfocused goals.
```

```
Focus next goal with bullet +.
1 goal
goal 1 is:
P y /\ Q y (succ y)
```

The Coq proof of the fact that the hypothesis `Q y y /\ P y` implies `P y /\ Q y (succ y)` is:

```
Coq < + destruct H4. split; trivial. apply q2. split; trivial.
No more goals, but there are some goals you gave up:
2 goals
goal 1 is:
Q x z
goal 2 is:
Q y y /\ P y
You need to go back and solve them.
```

Coq informs that the proof is incomplete and lists the two remaining goals to be proved. They correspond to the sequents labelling Nodes 7 and 16 in the pre-proof listed in Fig. 2.

In the next section, we explain why that the pre-proof from Fig. 2 is a proof using the soundness checking procedure implemented in E-Cyclist.

4 The soundness checking procedure

A cyclic *proof* is a cyclic pre-proof that satisfies the global trace condition discussed in the introductory part. To build a trace for an infinite path p, we have to assign a trace value for each of its nodes. We start by choosing an IAA from the sequent labelling the first node in p. For the next nodes in p, we proceed recursively. We associate to a node n one of the IAAs from its labelling sequent by following a *cause effect* rule that is: either i) the same trace value t for the previous node in p, or ii) the IAA resulting from the unfolding of t during the application of an (L.Unf.) that uses t. The last case corresponds to a *progress point*. The global trace condition holds if, for each infinite path, the case ii) occurs infinitely often.

Example 1 For the pre-proof from Fig. 2, there are three cases to build infinite paths:

1. by following the nodes $0, 2, 4, 5, 6, 7, 8, 0, ...$ The corresponding trace is N_2y, $\underline{N_2z}$, N_2z, N_2z, N_2z, N_2z, N_2y, N_2y,....[1] The progress point is underlined.
2. by following the nodes $4, 6, 10, 12, 14, 15, 16, 17, 4, ...$ The corresponding trace is N_1x, N_1x, $\underline{N_1y}$, N_1y, N_1y, N_1y, N_1y, N_1x, N_1x,....

[1] For sake of simplicity, the parentheses are omitted.

3. by infinitely intertwining finite parts from the paths presented in the first and the second cases. The trace along this path is built from the IAAs indexed by 1. The underlined IAA from the second case occurs infinitely often.

The reader may also consult Example 5.2.3 from [3] presenting a different CLKID$^\omega$ proof of the 'P and Q' conjecture.

The soundness checking procedure implemented in E-Cyclist was described in [11, 13]. It provides *sufficient* conditions to satisfy the global trace condition. Its two main steps are detailed in the following.

1. The normalisation step.

A pre-proof represented by the derivation tree $\mathcal{T}$ is transformed into a forest of connected derivation trees for which i) all companions are root nodes, ii) the root of $\mathcal{T}$ is among the root nodes, and iii) every path p leading a root r to a bud b, called *rb-path* and denoted by $r \rightarrow b$, ends with a $(Subst)$-step which is also the unique $(Subst)$-step in p.

Example 2 The result of the normalisation operation generates two trees with the roots 0 and 4, where the node 4 is duplicated twice in the tree with root 0. Once, to perform a stuttering step by applying a $(Subst)$-step with the identity substitution. The other time, to separate from the main tree the subtree rooted by it, followed by the creation of a new bud-companion relation. The normalised pre-proof is given in Fig. 3:

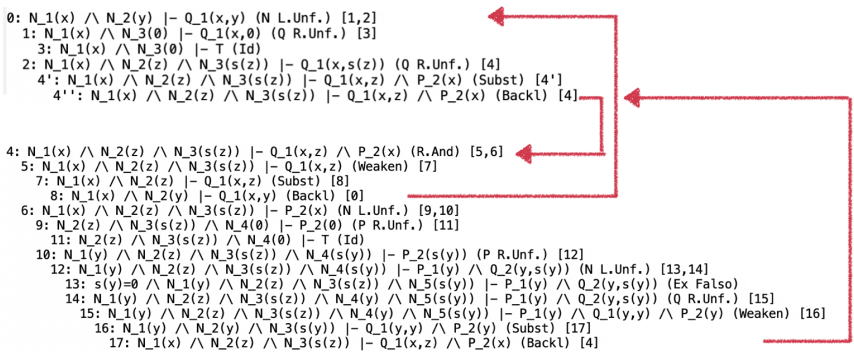

Fig. 3: The normalisation of the pre-proof from Fig. 2.

It can be noticed that every infinite path in the E-Cyclist pre-proof are also reproduced in its normal form, by duplicating the node 4, and vice-versa, by deleting the occurrences introduced during the normalisation step.

2. The ordering and derivability constraints step.

The next step is to associate to each root r, from a cycle of the normalized pre-proof, a measure value in terms of a multiset of IAAs from the sequent labelling r such that some derivability and ordering constraints are satisfied. The *derivability constraints* consist in checking that, for any rb-path p found in a cycle, every IAA from the measure value of the root companion of its bud can be derived from some IAA of the measure value of its root, according to some trace following p.

Example 3 For our example, we list all traces following the rb-paths found in cycles:

- $0 \to 4''$: i) N_1x, N_1x, N_1x, N_1x $(=)$, ii) $N_2y, \underline{N_2z}, N_2z, N_2z(>)$, and iii) $N_2y, N_3sz, N_3sz, N_3sz$ $(=)$,
- $4 \to 8$: i) N_1x, N_1x, N_1x, N_1x $(=)$, ii) N_2z, N_2z, N_2z, N_2y $(=)$, and
- $4 \to 17$: i) $N_1x, N_1x, \underline{N_1y}, N_1y, N_1y, N_1y, N_1y, N_1x$ $(>)$, ii) $N_1x, N_1x, N_4sy, \overline{N_4sy}, \underline{N_4y}, N_4y, N_4y, N_2z$ $(>)$, and iii) $N_1x, N_1x, N_4sy, N_4sy, \overline{N_5sy}, N_5sy, N_3sy, N_3sz$ $(=)$.

Each trace $A_r, ..., A_b$ following any of the rb-paths from Example 3 was annotated by $(>)$ if it has at least one progress point; the progress points are underlined. We say that A_r is 'greater than' A_b, denoted as $A_r > A_b$. The traces with no progress points are annotated by $(=)$. In this case, A_r and A_b are said to be *equal*. An *infinitely progressing trace* has infinite progress points.

Recall that the global trace condition requires that, for any infinite path p of the pre-proof, there is an infinitely progressing trace starting from some point of p. On the other hand, p can be represented in the normal form of the pre-proof as an infinite concatenation of rb-paths starting from some point. We will show that this condition is ensured if, for each rb-path $r \to b$ of p, we have $M(r) >_{mul} M(c(b))$, where $c(b)$ is the companion of b and $M(r)$ (resp., $M(c(b))$) is the measure value for r (resp., $c(b)$). The ordering $>_{mul}$ is the *trace-based multiset extension* of $>$, i.e., $M(r) >_{mul} M(c(b))$ if the equal IAAs from $M(r)$ and $M(c(b))$ are deleted, to get $M(r)'$ and $M(c(b))'$, and either i) $M(c(b))'$ is empty and $M(r)'$ is not empty, or ii) for each IAA $a \in M(c(b))'$ there is an IAA $b \in M(r)'$ such that $b > a$. This definition of $>_{mul}$ was firstly introduced in [15] and is a substitution-free version of the $<_\pi$-derivability relations used in [11, 13].

Example 4 In Fig. 4, only the indexes of the IAAs are listed but one can notice that the heuristics implemented in E-Cyclist [15] proposed (a permutation of) $\{N_1x, N_1x, N_2y, N_2y\}$ and $\{N_1x, N_1x, N_2z, N_2z, N_3sz\}$ as measure values for the nodes 0 and 4, respectively. In the same figure, the links of IAAs from buds to roots are checked, for each rb-path. For the rb-path $0 \to 4''$, $M(4'')$ is $M(4)$ since 4 is the companion of $4''$ and includes the IAAs indexed by 1, 2 and 3. The link of each of them to the IAAs from $M(0)$ are as follows:

```
Measures proposed for the roots in cycles:
    4: [2, 3, 1, 1, 2]
    0: [2, 1, 1, 2]
Checking the link of IAAs from buds to roots:
    4 to 0: | 1 -> 1 [false ]| 2 -> 2 [true ]| 3 -> 2 [false ] ==> true
    17 to 4: | 1 -> 1 [true ]| 2 -> 1 [true ]| 3 -> 1 [false ] ==> true
    8 to 4: | 1 -> 1 [false ]| 2 -> 2 [false ] ==> true
The proof has succeeded
```

Fig. 4: The measure values computed by E-Cyclist.

- the 1-indexed IAA of $M(4)$ (i.e., $N_1(x)$) is linked to the 1-indexed IAA of $M(0)$ (i.e., $N_1(x)$) by an non-progressing trace. This is denoted by `1 -> 1 [false]` in Fig. 4;
- the 2-indexed IAA of $M(4)$ (i.e., $N_2(z)$) is linked to the 2-indexed IAA of $M(0)$ (i.e., $N_2(y)$) by a progressing trace. This is denoted by `2 -> 2 [true]`;
- the 3-indexed IAA of $M(4)$ (i.e., $N_3(s(z))$) is linked to the 2-indexed IAA of $M(0)$ (i.e., $N_2(y)$) by an non-progressing trace. This is denoted by `3 -> 2 [false]`.

The progressing (resp., non-progressing) traces are denoted by `true` (resp., `false`).

Similarly, one can check which traces are progressing or not along the rb-paths $4 \to 17$ and $4 \to 8$.

Finally, the derivability and ordering constraints are checked for each rb-path, as follows:

- $0 \to 4''$: $\{N_1x, N_1x, N_2y, N_2y\} >_{mul} \{N_1x, N_1x, N_2z, N_2z, N_3sz\}$. Since N_2y equals N_3sz, the pairwise deletion of equal IAAs gives $\{N_2y\} >_{mul} \{N_2z, N_2z\}$ which holds since $N_2y > N_2z$. This is denoted by `==> true` in Fig. 4;
- $4 \to 17$: $\{N_1x, N_1x, N_2z, N_2z, N_3sz\} >_{mul} \{N_1x, N_1x, N_2z, N_2z, N_3sz\}$. The equal IAAs are deleted to issue $\{N_1x, N_2z, N_2z, N_3sz\} >_{mul} \{N_1x, N_1x, N_2z, N_2z\}$, for which N_1x from the lhs of $>_{mul}$ is greater than any IAA from $\{N_1x, N_1x, N_2z, N_2z\}$;
- $4 \to 8$: $\{N_1x, N_1x, N_2z, N_2z, N_3sz\} >_{mul} \{N_1x, N_1x, N_2y, N_2y\}$. It boils down to $\{N_3sz\} >_{mul} \{\}$.

We conclude that the global trace condition is satisfied.

We give below some important results of our approach.[2]

Lemma 1 (transitivity of $>_{mul}$) *$>_{mul}$ is transitive in cycles.*

[2]The full proofs are given in Appendix B.

Theorem 1 (soundness) *The global trace condition holds for a pre-proof if each rb-path $r \to b$ belonging to a cycle from its normalized pre-proof satisfies $M(r) >_{mul} M(c(b))$.*

5 Building Noetherian induction proofs from cyclic proofs

In the following, we will take advantage of the fact that the measure values assigned to the roots help also to represent the pre-proofs as Noetherian induction proofs for which the induction ordering is defined using the measure values for the roots, as computed in the previous section. We will use the (first-order) formula-based instance of the Noetherian induction principle [10]:

$$(\forall m \in \mathcal{E}, (\forall k \in \mathcal{E}, k < m \Rightarrow\ k) \Rightarrow\ m) \Rightarrow (\forall p \in \mathcal{E},\ p),$$

where $\mathcal{E}$ is a set of first-order formulas and $<$ is a Noetherian induction ordering defined over first-order formulas which forbids infinite strictly decreasing sequences of formulas. The principle states that any formula k from $\mathcal{E}$ can be used as induction hypothesis in the proof of another formula m from $\mathcal{E}$, as long as k is smaller than m.

In our setting, $\mathcal{E}$ is the set of (the first-order logic representation of) root sequents and $<$ the multiset extension $<^{wf}_{mul}$ of a Noetherian ordering $<^{wf}$ over the IAAs from the root sequents. The link between Noetherian induction reasoning and cyclic reasoning is based on the notion of $<_{pi}$*-derivability* introduced in [11, 13]. The idea is to use the measure values for the root sequents as computed by E-Cyclist. Given an rb-path $r \to b$, let δ be the substitution used in the `(Subst)`-step of $r \to b$ and θ the *cumulative substitution* for $r \to b$, i.e., the composition of all substitutions instantiating IAAs along $r \to b$ in `(L.Unf.)`-steps. It can be noticed that if $l_1 \in M(r)$ and $l_2 \in M(c(b))$ such that $l_1 = l_2$, then $l_1\delta \equiv l_2\theta$, where $\equiv$ is the syntactic equality. For the case when $l_1 > l_2$, we have to show that $l_1\delta <^{wf} l_2\theta$. We further denote by σ_{id} the identity substitution.

Example 5 For the running example, the measure value computed by E-Cyclist for the root 0 (resp., 4) are $\{Nx, Nx, Ny, Ny\}$ (resp., $\{Nx, Nx, Nz, Nz, Nsz\}$). In practice, one can consider that all the inductive predicate symbols are equivalent, hence the measure values for the roots are multisets built only from their arguments. The measure value for the root 0 (resp., 4) becomes $\{x, x, y, y\}$ (resp., $\{x, x, z, z, sz\}$).

The ordering constraints checking the validity of the rb-paths are:

- $0 \to 4''$: $\{x, x, z, z, sz\}[\sigma_{id}] <^{wf}_{mul} \{x, x, y, y\}[\{y \mapsto sz\}]$,
- $4 \to 8$: $\{x, x, y, y\}[\{y \mapsto z\}] <^{wf}_{mul} \{x, x, z, z, sz\}[\sigma_{id}]$, and
- $4 \to 17$: $\{x, x, z, z, sz\}[\{x \mapsto y;\ z \mapsto y\}] <^{wf}_{mul} \{x, x, z, z, sz\}[\{x \mapsto sy\}]$,

where the indexes for the inductive predicates are omitted.

They are satisfied if $<^{wf}$ is a multiset path ordering (mpo) [1] built from any precedence over the symbols $\{s, 0\}$.

We say that the rb-path $r \to b$ is *valid* if $M(c(b))[\delta] <^{wf}_{mul} M(r)[\theta]$. In the Noetherian induction setting, $M(c(b))[\delta]$ plays the role of *induction hypothesis* and $M(r)[\theta]$ that of the *induction conclusion*.

If every rb-path from a cycle of a normalized pre-proof resulting from a pre-proof $\mathcal{P}$ is valid is sufficient to conclude that $\mathcal{P}$ is a cyclic proof, even if $<^{wf}_{mul}$ is not Noetherian [11, 13].

In order to certify with Coq the Noetherian induction reasoning from FOL$_{\text{ID}}$ pre-proofs, we apply the certification methodology for formula-based Noetherian induction proofs [12]. Firstly, the mpo and its multiset extension, denoted by `less`, are defined using the Coccinelle library [6, 7]. To each function symbol from the Coq specification corresponds a Coccinelle function symbol with an arity, status and index:

```
Inductive symb : Set :=
| id_0
| id_S
.

Definition arity (f:symb) :=
  match f with
  | id_0 ⇒ term_spec.Free 0
  | id_S ⇒ term_spec.Free 1
  end.

Definition status (f:symb) :=
  match f with
  | id_0 ⇒ rpo.Mul
  | id_S ⇒ rpo.Mul
  end.

Definition index (f:symb) :=
  match f with
  | id_0 ⇒ 2
  | id_S ⇒ 3
  end.
```

The function *model_nat* translates Coq terms to Coccinelle terms:

```
Variable model_nat : T → term.
Axiom model_nat_0: model_nat zero = (Term id_0 nil).
Axiom model_nat_succ: ∀ (u1: T), model_nat (succ u1) = (Term id_S ((model_nat u1)::nil)).
```

The above equalities can be used for rewriting during the proofs:

```
Hint Rewrite model_nat_0 model_nat_succ : model_nat.
Ltac rewrite_model := autorewrite with model_nat.
```

The axiom stating the freeness of 0 and *succ* is required to reproduce the reasoning behind the application of the `(Ex Falso)` rule:

```
Axiom zero_different_from_succ: ∀ x, (zero = succ x) → False.
```

If the pre-proof has several non-singleton strongly connected components (SCCs), one should define a precedence w.r.t. their processing order, such that the SCC S_1 is treated after the SCC S_2 if S_1 depends on S_2. For each SCC, several steps should be performed. They are illustrated for the unique SCC S from the running example:

1. attach a measure value to each root from S inside a *functional* to ensure that whenever the root formula is instantiated the mesure value is also instantiated with the same substitution:

   `Definition type_LF_0_4 :=` $T \to T \to$ (`Prop` $\times$ (`List.list` **term**)).

   `Definition F_0 : type_LF_0_4:=` (`fun` *u1 u2* $\Rightarrow$ (**N** *u1* $\to$ **N** *u2* $\to$ **Q** *u1 u2*, ((*model_nat u1*)::(*model_nat u1*)::(*model_nat u2*)::(*model_nat u2*):: nil))).

   `Definition F_4 : type_LF_0_4:=` (`fun` *u1 u2* $\Rightarrow$ (**N** *u1* $\to$ **N** *u2* $\to$ **N** (*succ u2*) $\to$ **Q** *u1 u2* $\wedge$ **P** *u1*, ((*model_nat u1*)::(*model_nat u1*)::(*model_nat u2*)::(*model_nat u2*):: (*model_nat* (*succ u2*)):: nil))).

2. define the list of functionals:

   `Definition LF_0_4 := [F_0,F_4].`

3. define and prove the main lemma by reproducing the parts rooted by the root nodes in the pre-proof.[3] The main lemma validates the application conditions for the formula-based instance of the Noetherian induction principle:

   `Lemma main_0_4 :` $\forall$ *F*, In *F* LF_0_4 $\to \forall$ *u1*, $\forall$ *u2*, ($\forall$ *F'*, In *F'* LF_0_4 $\to \forall$ *e1*, $\forall$ *e2*, less (snd (*F' e1 e2*)) (snd (*F u1 u2*)) $\to$ fst (*F' e1 e2*)) $\to$ fst (*F u1 u2*).

4. prove all formulas from the list of functionals as the all_true conjecture, using the main lemma and the Noetherian induction principle integrated in Coq:

   `Theorem all_true:` $\forall$ *F*, In *F* LF_0_4 $\to \forall$ *u1*: *T*, $\forall$ *u2*: *T*, fst (*F u1 u2*).

5. finally, prove the root conjectures as direct consequences of all_true:

   `Theorem true_0:` $\forall$ *x y*, **N** *x* $\to$ **N** *y* $\to$ **Q** *x y*.

   `Theorem true_4:` $\forall$ *x y*, **N** *x* $\to$ **N** *y* $\to$ **N** (*succ y*) $\to$ **Q** *x y* $\wedge$ **P** *x*.

This completes the goal set in Section 3 to finish the proof of true_0.

5.1 Other examples

The certification procedure has also been successfully used for checking the other E-Cyclist proofs from [15], based on the FOL_{ID} examples from the standard distribution of Cyclist and the 2-Hydra problem [2]. For convenience, the axioms for the inductive predicates used in the examples are:

$$E\ \{\quad true \Rightarrow E(0)\ \mid\ O_1(x) \Rightarrow E(s(x))\quad\};$$

$$O\ \{\quad E_1(x) \Rightarrow O(s(x))\quad\};$$

$$R\ \{\quad true \Rightarrow R(0,x)\ \mid\ R_1(x,0) \Rightarrow R(s(x),0)\ \mid\ R_1(s(s(x)),y) \Rightarrow R(s(x),s(y))\quad\};$$

[3] The full Coq proof script of the main lemma is given in Appendix A.

$$
\begin{array}{ll}
ADD\ \{ & p\ \{ \\
\quad N_1(x) \Rightarrow ADD(0,x,x)\ \mid & \quad true \Rightarrow p(0,0)\ \mid \\
\quad ADD_1(x,y,z) \Rightarrow ADD(s(x),y,s(z)) & \quad true \Rightarrow p(s(0),0)\ \mid \\
\} ; & \quad true \Rightarrow p(x,s(0))\ \mid \\
 & \quad p_1(x,y) \Rightarrow p(s(x),s(s(y)))\ \mid \\
 & \quad p_1(s(y),y) \Rightarrow p(0,s(s(y)))\ \mid \\
 & \quad p_1(s(x),x) \Rightarrow p(s(s(x)),0) \\
 & \} ;
\end{array}
$$

The inductive predicates E, O, and ADD refer to the even and odd numbers, as well as to the addition on naturals, respectively. R is an inductive predicate given as example in Cyclist and p formalizes the 2-Hydra problem.

Table 1 contains some statistics about the certification of a dozen of E-Cyclist proofs. The columns show, from left to right, the number of the problem as it figures in the standard distribution of Cyclist (excepting for the 2-Hydra problem listed at the bottom line), and related to the normalized E-Cyclist pre-proof: its number of SCCs, its root sequents and their measure values.[4]

An important part of the Coq script built for the running example was shared with the Coq script for the other examples, as the definition of the Coccinelle orderings.[5]. The main intellectual energy was spent to prove the main lemmas.

In some cases, the translation of the application of `(L.Unf.)` using an IAA of the form $N(s(x))$ requires an axiom stating the injectivity of `succ`:

```
Axiom succ_is_injective: ∀ x y, (succ x = succ y) → x = y.
```

6 Conclusions and future work

We have adapted a general procedure for certifying formula-based Noetherian induction reasoning to check with Coq FOL$_{ID}$ cyclic proofs, as those produced by the E-Cyclist prover. Given a cyclic proof of a sequent S, the output is the Coq script proving a bunch of theorems and lemmas among which there is the formula representing S in first-order logic. The deductive part of the main lemma translates the steps from the cyclic proof. Some of them are converted to Coq script in a one-to-one translation. Others, like the unfolding of a conjunct in a `(R.Unf.)` step, may require more than one Coq proof steps. In fact, Coq is enough powerful to translate *any* E-Cyclist inference step. For the translation of classical proof steps, the axiom of excluded middle is required

[4]The full Coq specifications and proof scripts are archived and can be downloaded from https://drive.google.com/file/d/1E3_kO72mgAzZexEjHbbL8_nrf4LkAMIG.

[5]see the files `utils.v`, `coccinelle_utils.v` and `ordering.v` in the archive.

#	Theorem	# SCC	Root Sequents	Measure values	
01	$O_1(x) \vdash N(x)$	1	$O_1(x) \vdash N(x)$	$\{x\}$	
02	$E_1(x) \vee O_2(x) \vdash N(x)$	2	$E_1(x) \vdash N(x)$	$\{x\}$	
			$O_2(x) \vdash N(x)$	$\{x\}$	
03	$E_1(x) \vee O_1(x) \vdash N(x)$	1	$E_1(x) \vdash N(x)$	$\{x\}$	
			$O_1(x) \vdash N(x)$	$\{x\}$	
04	$N_1(x) \vdash O(x) \vee E(x)$	1	$N_1(x) \vdash O(x) \vee E(x)$	$\{x\}$	
05	$N_1(x) \wedge N_2(y) \vdash Q(x,y)$	1	$N_1(x) \wedge N_2(y) \vdash Q(x,y)$	$\{x,x,y,y\}$	
			$N_1(x) \wedge N_2(z) \wedge N_3(s(z)) \vdash Q(x,z) \wedge P(x)$	$\{x,x,z,z,s(z)\}$	
07	$N_1(x) \vdash ADD(x,0,x)$	1	$N_1(x) \vdash ADD(x,0,x)$	$\{x\}$	
08	$N_1(x) \wedge N_2(y) \wedge ADD_3(x,y,z) \vdash N(z)$	1	$N_1(x) \wedge N_2(y) \wedge ADD_3(x,y,z) \vdash N(z)$	$\{x,z\}$	Yes
09	$N_1(x) \wedge N_2(y) \wedge ADD_3(x,y,z) \vdash ADD(x,s(y),s(z))$	1	$N_1(x) \wedge N_2(y) \wedge ADD_3(x,y,z) \vdash ADD(x,s(y),s(z))$	$\{x,z\}$	
12	$N_1(x) \wedge N_2(y) \vdash R(x,y)$	2	$N_1(x) \wedge N_2(y) \vdash R(x,y)$	$\{y\}$	
			$N_1(w) \wedge N_3(s(s(w))) \wedge N_4(s(w)) \wedge N_5(0) \vdash R(s(w),0)$	$\{w\}$	
	$N_1 x \wedge N_2 y \vdash p_1(x,y)$	1	$N_1 x \wedge N_2 y \vdash p_1(x,y)$	$\{x,y\}$	

Table 1: Statistics about the mechanical checking of the E-Cyclist proofs.

because the logic behind Coq is constructive. Therefore, our approach can find errors in the cyclic proofs in a very precise way, at the level of inference steps. The induction reasoning used in the main lemma is based on the Noetherian induction principle integrated in Coq. Hence, our work establishes a bridge between formula-based Noetherian induction and FOL_{ID} cyclic induction.

The procedure has been successfully applied to certify every cyclic proof built for the FOL_{ID} examples from the official distribution of Cyclist, as well as for the 2-Hydra problem. More examples can be processed if the cyclic reasoning is integrated directly in Coq. In the current state, the induction reasoning is done manually. In the future, we intend to *automatize* it, by using an approach similar to [12] allowing for the automatic conversion of SPIKE proofs to Coq scripts.

References

[1] Baader F, Nipkow T (1998) Term Rewriting and All That. Cambridge University Press

[2] Berardi S, Tatsuta M (2019) Classical system of Martin-Lof's inductive definitions is not equivalent to cyclic proofs. Logical Methods in Computer Science 15(3). https://doi.org/10.23638/LMCS-15(3:10)2019

[3] Brotherston J (2006) Sequent calculus proof systems for inductive definitions. PhD thesis, University of Edinburgh

[4] Brotherston J, Simpson A (2011) Sequent calculi for induction and infinite descent. Journal of Logic and Computation 21(6):1177–1216. https://doi.org/10.1093/logcom/exq052

[5] Brotherston J, Gorogiannis N, Petersen RL (2012) A generic cyclic theorem prover. In: APLAS-10 (10th Asian Symposium on Programming Languages and Systems), LNCS, vol 7705. Springer, pp 350–367, https://doi.org/10.1007/978-3-642-35182-2_25

[6] Contejean E, Courtieu P, Forest J, et al (2007) Certification of automated termination proofs. Frontiers of Combining Systems pp 148–162. https://doi.org/10.1007/978-3-540-74621-8_10

[7] Contejean E, Paskevich A, Urbain X, et al (2010) A3PAT, an approach for certified automated termination proofs. In: Gallagher JP, Voigtländer J (eds) PEPM - Proceedings of the 2010 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation, PEPM 2010, Madrid, Spain. ACM, pp 63–72

[8] Gentzen G (1935) Untersuchungen über das logische Schließen. I. Mathematische Zeitschrift 39:176–210. https://doi.org/10.1007/BF01201353

[9] Michel M (1988) Complementation is more difficult with automata on infinite words. Tech. rep., CNET

[10] Stratulat S (2012) A unified view of induction reasoning for first-order logic. In: Voronkov A (ed) Turing-100 (The Alan Turing Centenary Conference), EPiC Series, vol 10. EasyChair, pp 326–352

[11] Stratulat S (2017) Cyclic proofs with ordering constraints. In: Schmidt RA, Nalon C (eds) TABLEAUX 2017 (26th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods), LNAI, vol 10501. Springer, pp 311–327, https://doi.org/10.1007/978-3-319-66902-1_19

[12] Stratulat S (2017) Mechanically certifying formula-based Noetherian induction reasoning. Journal of Symbolic Computation 80, Part 1:209–249. https://doi.org/10.1016/j.jsc.2016.07.014

[13] Stratulat S (2018) Validating back-links of $\mathrm{FOL}_{\mathrm{ID}}$ cyclic pre-proofs. In: Berardi S, van Bakel S (eds) CL&C'18 (Seventh International Workshop on Classical Logic and Computation), no. 281 in EPTCS, pp 39–53, https://doi.org/10.4204/EPTCS.281.4

[14] Stratulat S (2020) SPIKE, an automatic theorem prover – revisited. In: SYNASC 2020: Proceedings of the 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing. IEEE Computer Society, pp 93–96

[15] Stratulat S (2021) E-Cyclist: Implementation of an efficient validation of $\mathrm{FOL}_{\mathrm{ID}}$ cyclic induction reasoning. In: Kutsia T (ed) 9th International Symposium on Symbolic Computation in Software Science, pp 129–135

[16] The Coq development team (2020) The Coq Reference Manual. INRIA, http://coq.inria.fr/doc

[17] Wirth CP (2004) Descente infinie + Deduction. Logic Journal of the IGPL 12(1):1–96. https://doi.org/10.1093/jigpal/12.1.1

[18] Wirth CP (2012) Computer-assisted human-oriented inductive theorem proving by *descente infinie* - a manifesto. Logic Journal of the IGPL 20(6):1046–1063. https://doi.org/10.1093/jigpal/jzr048

A The Coq proof of F_0 in the main lemma

```
intros. simpl. intros. rename H into Hind.
    (* instantiate y from Q(x,y) *)
 inversion H1 as [|z].
- (* Q(x,0) *) rewrite ← H. apply q1.
- (* Q(x,s(z)) *) apply q2. split.
    (* induction step for Q(x,z) *)
  -- apply (Hind F_0).
     --- simpl. left. trivial.
     --- rewrite ← H2. unfold snd. unfold F_4. unfold F_0.
rewrite_model. abstract solve_rpo_mul.
     --- trivial.
     --- trivial.
    (* instantiate x from P(x) *)
  -- rewrite ← H2 in Hind. rewrite ← H2 in H1. clear H2 y. inversion
H0 as [|y].
     --- rewrite ← H2. apply p1.
     --- apply p2. rewrite ← H3 in H0.
    (* instantiate P(s(y)) *)
        inversion H0.
        + apply zero_different_from_succ in H4. contradiction.
        + rewrite H4.
        ++ assert (Q y y ∧ P y).
    (* induction step for Q y y / P y *)
           apply (Hind F_4).
          +++ simpl. right. left. trivial.
          +++ unfold snd. unfold F_4. unfold F_0. rewrite ← H3.
rewrite_model. abstract solve_rpo_mul.
          +++ trivial.
          +++ trivial.
          +++ trivial.
          +++ destruct H6. split; trivial. apply q2; trivial. split;
trivial.
```

B The missing proofs from Section 4

B.1 Proof of Lemma 1

Proof Let us assume two adjacent rb-paths $r_1 \to b_1$ and $r_2 \to b_2$ in a cycle such that r_2 is the companion of b_1, $M(r_1) >_{mul} M(c(b_1))$, and $M(r_2) >_{mul} M(c(b_2))$. We will try to prove that $M(r_1) >_{mul} M(c(b_2))$.

Since the roots/buds are repeated infinitely along the cycle, none of their measure values should be empty. If, by contradiction, we assume that a measure value is empty, there exists an rb-path $r \to b$ in the cycle such that $M(r)$ is empty. But there is no $<_{mul}$ such that $M(r) > M(c(b))$.

We show that there should be a trace along the path $[r_1, \ldots, b_1, r_2, \ldots, b_2]$. Since $M(c(b_2))$ is not empty, let $l_3 \in M(c(b_2))$. By the definition of $<_{mul}$, there exists an IAA $l_2 \in M(r_2)$ such that $l_2 > l_3$ or $l_2 = l_3$. A similar reasoning can be applied on l_2 as for l_3 to conclude that there is $l_1 \in M(r_1)$ such that $l_1 > l_2$ or $l_1 = l_2$. Hence, the trace is $l_1, \ldots, l_2, l_2, \ldots, l_3$.

Finally, we check whether $M(r_1) >_{mul} M(c(b_2))$. As previously, for each IAA $l_3 \in M(c(b_2))$ there is an IAA $l_2 \in M(r_2)$ such that $l_2 > l_3$ or $l_2 = l_3$, and an IAA $l_1 \in M(r_1)$ such that $l_1 > l_2$ or $l_1 = l_2$. We perform a case analysis on the comparison results between l_2 and l_3, as well as l_1 and l_2:

- if $l_2 > l_3$ and $l_1 > l_2$, then $l_1 > l_3$;
- if $l_2 > l_3$ and $l_1 = l_2$, then $l_1 > l_3$;
- if $l_2 = l_3$ and $l_1 > l_2$, then $l_1 > l_3$;
- if $l_2 = l_3$ and $l_1 = l_2$, then $l_1 = l_3$.

We show that $M(r_1) >_{mul} M(c(b_2))$ when there is at least one $l_3 \in M(c(b_2))$ for which it exists $l_1 \in M(r_1)$ such that $l_1 > l_3$. After the pairwise deletion of equal IAAs from $M(r_1)$ and $M(c(b_2))$, to get $M(r_1)'$ and $M(c(b_2))'$, we have that $l_3 \in M(c(b_2))'$, $l_1 \in M(r_1)'$ and $l_1 > l_3$. By the definition of $>_{mul}$, we have that $M(r_1) >_{mul} M(c(b_2))$, because the same reasoning can be done for any other IAA from $M(c(b_2))'$ as for l_3.

If for all $l_3 \in M(c(b_2))$, there exists $l_1 \in M(r_1)$ such that $l_3 = l_1$, then it exists $l_2 \in M(r_2)$ such that $l_1 = l_2$ and $l_2 = l_3$. Since $M(r_2) >_{mul} M(c(b_2))$, after the pairwise deletion of IAAs from $M(r_2)$ and $M(c(b_2))$, to get $M(r_2)'$ and $M(c(b_2))'$, we have that $M(c(b_2))'$ is empty and $M(r_2)'$ is not empty. Similarly, since $M(r_1) >_{mul} M(c(b_1))$, we have that $M(c(b_1))'$ is empty and $M(r_1)'$ is not empty. We conclude that after the pairwise deletion of IAAs from $M(r_1)$ and $M(c(b_2))$, to get $M(r_1)'$ and $M(c(b_2))'$, we have that $M(c(b_2))'$ is empty and $M(r_1)'$ is not empty. Hence $M(r_1) >_{mul} M(c(b_2))$, as required.

□

B.2 Proof of Theorem 1

Proof Let $\mathcal{P}$ be a pre-proof and p_0 an infinite path in $\mathcal{P}$. Let p be the infinite path from the normalized pre-proof of $\mathcal{P}$ built from p_0 by duplicating the nodes as shown during the normalization process. By construction, the path p is built starting from some point only from the concatenations of rb-paths from cycles from the normalized pre-proof. Since the number of roots is finite in $\mathcal{P}$, there is a root r in p that occurs infinitely often in p. Hence, there is an infinite sub-path p' of p of the form $[r, \ldots, r, \ldots]$ which can be represented as the infinite concatenation of finite sub-paths $[r, \ldots, b]$, where r is the companion of b and r occurs only once in each sub-path. Each such sub-path is built from a finite number of concatenations of rb-paths of the form $[r_1, \ldots, b_1]$ for which $M(r_1) >_{mul} M(c(b_1))$. Since $<_{mul}$ is transitive, by Lemma 1, we have that $M(r) >_{mul} M(c(b))$ for each sub-path $[r, \ldots, r, \ldots, b]$. Since $M(r)$ is not empty, for the same reasons as presented in the proof of Lemma 1, there is a trace following p'.

By contradiction, we assume that the number of progress points in all the traces along p' is finite. Since the cardinality of the trace measures for each root in p' is finite, there is a sub-path $[r, \ldots, b]$ defined as above whose traces have no progress points and satisfies $M(r) >_{mul} M(c(b))$. After the pairwise deletion of equal IAAs from $M(r)$ and $M(c(b))$, to get $M(r)'$ and $M(c(b))'$, we perform a case analysis by considering whether $M(c(b))'$ is empty or not.

- $M(c(b))'$ is not empty. By the definition of $>_{mul}$, there is an IAA $l \in M(c(b))'$ for which there is an IAA $l' \in M(r)'$ such that $l' > l$. Hence, the trace leading l' to l has at least one progress point. Contradiction.
- $M(c(b))'$ is empty. By a similar reasoning as given in the proof of Lemma 1. The cardinality of $M(r)$ is greater than that of $M(c(b))$. But $M(r) = M(c(b))$ because r is the companion of b. Contradiction, again.

Since there is a trace along p' that has an infinite number of progress points, p has also an infinitely progressing trace starting from some point. Since p_0 can be built from p by deleting the extra nodes added during the normalisation process, it has an infinitely progressing trace starting from some point, too.

We conclude that $\mathcal{P}$ satisfies the global trace condition.

□