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 consists in blocking the proof development of certain subgoals already encountered during the proof process. In the setting of first-order logic with inductive definitions and equality (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 into certifying proof environments that support first-order logic and inductive definitions, such as Isabelle and Coq.

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

- I) The certification of cyclic FOL_{ID} proofs is *mechanical*. Coq can validate every single step from the E-Cyclist proofs, as well as the induction arguments; also, it helps to identify errors in a very precise way.
- II) There is a great potential for *automation*. The methodology has already been used to automatically convert to Coq scripts implicit induction proofs.
- III) Cyclic induction can be *directly* performed in Coq. Coq functions are provided to manage the induction part.

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

1 Introduction

Broadly speaking, *cyclic induction* is a reasoning technique used to stop the proof of certain subgoals if they have been already generated during the proof process. In the setting of first-order logic with inductive definitions and equality (FOL_{ID}), CLKID^{ω} [1, 2] is the standard inference system that can build (cyclic) proofs based on this technique. It extends the sequent-based LK proof system [3] with rules that process equalities and inductive predicates. The *cyclic pre-proof* of a sequent S is a finite tree-shaped proof derivation rooted by S that may have special terminal nodes called *buds*. Every bud has associated one *companion* node in the tree, labeled with the *same* sequent.

The bud-companion relations may introduce *cycles*, leading to *infinite* paths in the pre-proofs. Some CLKID^{ω} pre-proofs may be *unsound*, for example, the pre-proof of a sequent S that results from instantiating with some constant c a variable v from S, then generalizing c by replacing it with v resulting in S. Cyclic induction can be used for building sound pre-proofs, referred to as *proofs*, as a heuristic for deciding whether the bud sequents should be further processed or not.

Cyclist [4] is the *de facto* theorem prover that implements CLKID^{ω} to automate the FOL_{ID} cyclic (induction) reasoning. The Cyclist pre-proofs are represented as an indented text, displaying one node per line. A node N is labeled using the pattern

$$n : S \ (R) \ nl \quad ,$$

where n is a unique natural number associated to N, S is a sequent, R is the name of the inference rule applied on S, and nl is a list of naturals corresponding, except the `Backl` rule introduced in the next paragraph, to the nodes built for the sequents resulting from the application of R on S. The list nl is not displayed when no new sequents are generated. The values for n increment in chronological order, the root node being prefixed by 0.

Fig. 1 highlights a cycle from the Cyclist pre-proof, fully presented later in Fig. 2, of the sequent `N(x),N(y)` $\vdash$ `Q(x,y)`, represented by Cyclist in the indexed form `N_1(x) /\ N_2(y) |- Q_1(x,y)` and in the body text by the more compact form $N_1(x), N_2(y) \vdash Q_1(x, y)$. One reason for indexing the inductive atoms is to distinguish their duplicates, a feature required to represent traces, as explained in the next paragraph. The `Backl` (backlink) rule has been applied on Node 8 to express the fact that it is a bud for which Node 0, the only element of nl, is its companion. To ease the identification of cycles, the backlinks are represented by solid arrows leading buds to their companions. For example, the backlink from Fig. 1 introduces a cycle that can be visited by the infinite path $[0, 2, 4, 5, 7, 8, 0, 2, ...]$.

A sequent can be viewed as two multisets of formulas separated by the `|-` symbol. It can be interpreted as a logical implication stating that the conjunctions of formulas from the lhs of `|-`, called *antecedents*, imply the disjunction of formulas from the rhs of `|-`, called *succedents*. The inductive (*antecedent*) atoms

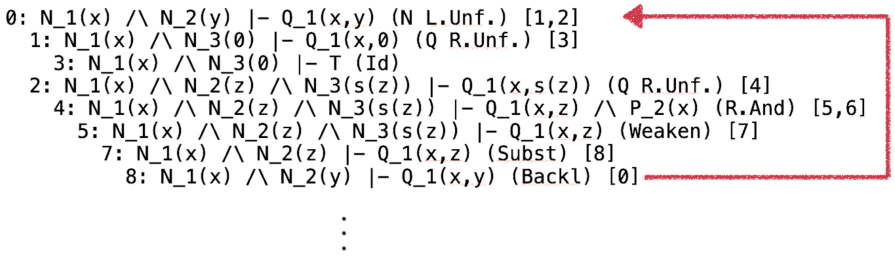

Fig. 1: Indented representation of a derivation tree in (E-)Cyclist.

(IAAs) are instrumental in checking the soundness of a pre-proof. The checking procedure is non-trivial and consists in verifying a *global trace condition*. The idea is to associate to every infinite path p in the pre-proof a *trace* built from IAAs occurring in the sequents labeling the nodes from p, then show that some *progress* steps happen infinitely often along p. The standard way to check the global trace condition is by using a complete procedure, based on an exponential complement operation for Büchi automata [5], which was implemented in Cyclist. Later [6, 7], 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 [8]. Due to the complexity of its implementation, consisting of thousands of lines of Ocaml (www.ocaml.org) code, the generated proofs are error-prone and their validation by humans can be tedious. Hence, there is a need for solutions that can certify mechanically the FOL$_{\text{ID}}$ cyclic reasoning.

In this paper, we propose a convenient way to mechanically certify E-Cyclist proofs using the certifying proof environment provided by the Coq proof assistant [9]. The main idea comes from the fact that Coq can define and manipulate inductive predicates, reason on first-order formulas, and perform induction reasoning by the means of the Noetherian induction principle. In other words, the logic underlying Coq is rather expressive to represent FOL$_{\text{ID}}$ specifications and the Coq's inference system is sufficiently powerful to reproduce every single pre-proof step. The most technical part consists in converting the soundness arguments underlying the cyclic induction reasoning into a 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, such as the companions occurring in cycles, a *weight* (measure value) consisting of a multiset of IAAs from the sequent of n. These weights will be crucial for the definition of the required *Noetherian (well-founded) induction orderings*. The Noetherian induction arguments are built following a methodology developed previously [10] to certify formula- and Noetherian induction-based first-order reasoning with Coq, in particular implicit induction proofs produced by the SPIKE theorem prover [11]. In this paper, the same methodology will be applied to certify the E-Cyclist proofs of the FOL$_{\text{ID}}$ examples included

in the official distribution of the release CSL-LICS14 of Cyclist, as well as the 2-Hydra problem [12].

Our approach presents two other advantages. Firstly, we give evidence that the Coq users can *directly* perform cyclic induction reasoning in Coq. The presented examples may serve as a tutorial and can inspire the users to produce and certify new cyclic proofs in Coq. To alleviate the certification process, we provide a set of Coq functions that can be reused to define the Noetherian induction orderings and to deal with the cyclic induction reasoning. Secondly, there is room for automation; in the past, it has been shown that SPIKE proofs can be automatically converted to Coq scripts [10] by the means of the certification methodology. Also, Coq tactics have been devised to automatically perform implicit induction reasoning [13] in Coq. Hence, this paper presents strong grounds to believe that similar results can be achieved in the case of the FOL_{ID} cyclic proofs.

Related work. As noticed in [4], there is a similitude between the global trace condition underlying the cyclic proofs and the 'size-change' termination condition [14] requiring that every infinite path in the control flow graph should have some weights that infinitely strictly decrease during the program execution, according to a well-founded ordering. Originally developed for termination analysis, the size-change termination has been checked empirically [15] and proved to be more effective w.r.t. a Büchi complementation approach. Recently [16], the size-change principle has been used to develop a cyclic theorem prover able to reason on unconditional equalities.

Structure of the paper. The rest of the paper has five sections and one appendix. Section 2 presents as a running example an E-Cyclist 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 E-Cyclist pre-proof to a Coq script. Section 4 presents the machinery for checking whether the E-Cyclist pre-proof is a cyclic proof, including the soundness procedure implemented in E-Cyclist and the definition of the weights. The methodology for translating the induction reasoning part and, in particular, the procedure for building the Noetherian induction arguments underlying the cyclic proof, are described in Section 5. This section illustrates how the Coq script for certifying the running example was built and summarizes the results for the rest of the examples taken from the official distribution of Cyclist, as well as the 2-Hydra example. The last section concludes and gives directions for future work. The appendix completes the Coq script from Section 3 with the induction reasoning part 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 the 'P and Q' problem [17] and consists in proving the atom $Q(x, y)$,

for all naturals x and y. In [1], Q is an inductive predicate that is mutually defined using another inductive predicate P. The two predicates are defined over the function symbols 0 and s (the 'successor' function) using the axioms:

$$\frac{}{P(0)} \qquad \frac{P(x) \quad Q(x, s(x))}{P(s(x))} \qquad \frac{}{Q(x, 0)} \qquad \frac{Q(x, y) \quad P(x)}{Q(x, s(y))}$$

In the FOL_{ID} setting, the 'P and Q' problem requires also the formalization of naturals, by the means of the inductive predicate N defined by the axioms:

$$\frac{}{N(0)} \qquad \frac{N(x)}{N(s(x))}$$

E-Cyclist requires a more elaborated version of these definitions by representing each axiom as a logical implication:

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

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

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

where *true* is the boolean constant interpreted as being always true.

The inductive atoms from the condition part of the axioms are indexed, and the index values play an important role in the proof strategy. For example, conjectures 02 and 03 from Table 1 refer to the same sequent for which E-Cyclist generated two different proofs only by changing the index values. More details are discussed later in Section 5.

In the past, several induction reasoning systems succeeded to prove the 'P and Q' problem in different logical settings [1, 10, 17]. In our case, it is formalized in E-Cyclist as the sequent $N_1(x), N_2(y) \vdash Q_1(x, y)$. Its pre-proof is given in Fig. 2.

Before explaining how the pre-proof was built, we give some details about the E-Cyclist inference system. Almost all of its inference rules are implementations of the CLKID$^\omega$ rules presented in the following. The classical first-order reasoning is performed by CLKID$^\omega$ using Gentzen's LK [3] rules displayed in Fig. 3. It can be noticed that, for each logical connective, there are two basic *introduction* rules that introduce it on the left and on the right of the $\vdash$ symbol, respectively.

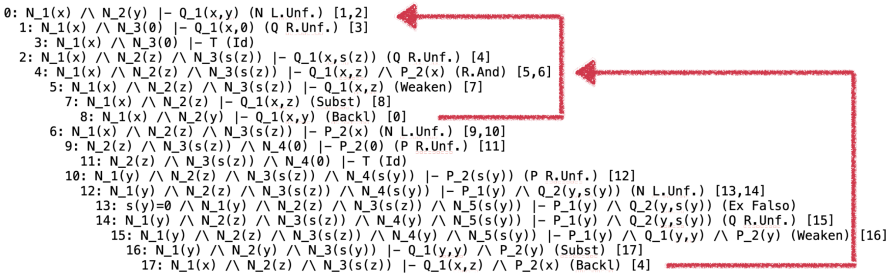

Fig. 2: The E-Cyclist pre-proof of $N_1(x), N_2(y) \vdash Q_1(x, y)$.

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

$$\frac{\Gamma \vdash F, \Delta}{\Gamma, \neg F \vdash \Delta}\ (\neg L) \qquad \frac{\Gamma, F \vdash \Delta}{\Gamma \vdash \neg F, \Delta}\ (\neg R) \qquad \frac{\Gamma, F \vdash \Delta \quad \Gamma, G \vdash \Delta}{\Gamma, F \vee G \vdash \Delta}\ (\vee L)$$

$$\frac{\Gamma \vdash F, G, \Delta}{\Gamma \vdash F \vee G, \Delta}\ (\vee R) \qquad \frac{\Gamma, F, G \vdash \Delta}{\Gamma, F \wedge G \vdash \Delta}\ (\wedge L) \qquad \frac{\Gamma \vdash \Delta}{\Gamma[\theta] \vdash \Delta[\theta]}\ (Subst)$$

$$\frac{\Gamma \vdash F, \Delta \quad \Gamma, G \vdash \Delta}{\Gamma, F \Rightarrow G \vdash \Delta}\ (\Rightarrow L) \qquad \frac{\Gamma, F \vdash \Delta}{\Gamma, \exists \overline{x} F \vdash \Delta}\ \overline{x} \cap FV(\Gamma \cup \Delta) = \emptyset\ (\exists L)$$

$$\frac{\Gamma \vdash F, \Delta}{\Gamma \vdash \forall \overline{x} F, \Delta}\ \overline{x} \cap FV(\Gamma \cup \Delta) = \emptyset\ (\forall R) \qquad \frac{\Gamma, F[\{\overline{x} \mapsto \overline{t}\}] \vdash \Delta}{\Gamma, \forall \overline{x} F \vdash \Delta}\ (\forall L)$$

$$\frac{\Gamma \vdash F, \Delta \quad \Gamma, F \vdash \Delta}{\Gamma \vdash \Delta}(Cut) \qquad \frac{\Gamma \vdash F, F, \Delta}{\Gamma \vdash F, \Delta}(contrR) \qquad \frac{\Gamma, F \vdash G, \Delta}{\Gamma \vdash F \Rightarrow G, \Delta}(\Rightarrow R)$$

$$\frac{\Gamma \vdash F, \Delta \quad \Gamma \vdash G, \Delta}{\Gamma \vdash F \wedge G, \Delta}(\wedge R) \qquad \frac{\Gamma, F, F \vdash \Delta}{\Gamma, F \vdash \Delta}(contrL) \qquad \frac{\Gamma \vdash F[\{\overline{x} \mapsto \overline{t}\}], \Delta}{\Gamma \vdash \exists \overline{x} F, \Delta}(\exists R)$$

Fig. 3: Rules for classical first-order logic.

CLKID$^{\omega}$ has also rules to introduce inductive atoms. The *unfold* rule $(R.(1))$ introduces to the right an inductive atom $P(\overline{t}')$ using one of the axioms defining P, denoted by (1) and defined as:

$$(Q_1(\overline{u}_1) \wedge \ldots \wedge Q_h(\overline{u}_h) \wedge P_1(\overline{t}_1) \wedge \ldots \wedge P_l(\overline{t}_l)) \Rightarrow P(\overline{t}), \tag{1}$$

where h, l are naturals, $\overline{u}_1 \ldots \overline{u}_h, \overline{t}_1 \ldots \overline{t}_l, \overline{t}, \overline{t}'$ are term vectors, $Q_1, \ldots, Q_h$ are ordinary predicate symbols, and $P_1, \ldots, P_l, P$ are inductive predicate symbols. $(R.(1))$ can be applied on $\Gamma \vdash P(\overline{t}'), \Delta$ as:

$$\frac{\mathit{seq\_Q\_inst} \qquad \mathit{seq\_P\_inst}}{\Gamma \vdash P(\overline{t}'), \Delta}\,(R.(1)) \quad ,$$

if $P(\overline{t}')$ is an instance of $P(\overline{t})$ using some substitution σ. *seq_Q_inst* (resp., *seq_P_inst*) is the multiset of sequents $\bigcup_{i=1}^{h}\{\Gamma \vdash Q_i(\overline{u}_i)[\sigma], \Delta\}$ (resp., $\bigcup_{j=1}^{l}\{\Gamma \vdash P_j(\overline{t}_j)[\sigma], \Delta\}$) resulting in the replacement of $P(\overline{t}')$ from $\Gamma \vdash P(\overline{t}'), \Delta$ with the instantiations of $Q_i(\overline{u}_i)$ ($i \in [1..h]$) (resp., $P_j(\overline{t}_j)$ ($j \in [1..l]$)) using σ.

($Case$) introduces inductive atoms to the left:

$$\frac{\text{case distinctions}}{\Gamma, P(\overline{t}') \vdash \Delta}\,(Case\ P(\overline{t}'))$$

A *case distinction* is built for each axiom of the form (1):

$$\Gamma, \overline{t}' = \overline{t}, Q_1(\overline{u}_1), \ldots, Q_h(\overline{u}_h), P_1(\overline{t}_1), \ldots, P_l(\overline{t}_l) \vdash \Delta \tag{2}$$

where each free variable y from (1) is fresh w.r.t. the *free variables* from the conclusion of the rule (y can be renamed to a fresh variable, otherwise). The inductive atom $P(\overline{t}')$ is the *active formula* of ($Case\ P(\overline{t}')$) and the IAAs $P_1(\overline{t}_1), \ldots, P_l(\overline{t}_l)$ occurring in each case distinction are *case descendents* of $P(\overline{t}')$.

Finally, CLKID$^{\omega}$ includes also the two rules ($= R$) and ($= L$) from Fig. 4 that introduce equalities on the right and on the left, respectively:

$$\frac{}{\Gamma \vdash t = t, \Delta}\,(= R)$$

$$\frac{\Gamma[\{x \mapsto u;\ y \mapsto t\}] \vdash \Delta[\{x \mapsto u;\ y \mapsto t\}]}{\Gamma[\{x \mapsto t;\ y \mapsto u\}], t = u \vdash \Delta[\{x \mapsto t;\ y \mapsto u\}]}\,(= L)$$

Fig. 4: Sequent-based rules for equality reasoning.

where $\{x \mapsto u;\ y \mapsto t\}$ (resp., $\{x \mapsto t;\ y \mapsto u\}$) is the substitution that replaces x by u and y by t (resp., x by t and y by u).

The E-Cyclist pre-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 `(N L.Unf.)`, where z is a fresh variable. `(N L.Unf.)` is an implementation of ($Case$) and ($= L$) that defines the instantiation schema from the two axioms of N. For Node 2, $N_2(z)$ was added as IAA from the unfolding of $N_3(s(z))$ using the second axiom of N. By only displaying `(N L.Unf.)`, it is not clear which one of the IAAs $N_1(x)$ and $N_2(y)$ was used. Hopefully, it can be noticed from what is following in the pre-proof that the variable y of $N_2(y)$ was instantiated. The `(Q R.Unf.)` rule unfolds $Q_1(x, 0)$ from the sequent of Node 1 with the first axiom defining Q to get a new sequent that is validated by the `(Id)` rule.[1] This combination of `(R.Unf.)`

[1] `T` stands for *true*.

and (Id) rules is an indirect implementation of the LK's (Ax) rule. The valid sequents are no longer processed by E-Cyclist.

The pre-proof construction is continued by unfolding $Q_1(x, s(z))$ from the sequent of Node 2 using the second axiom defining Q to get a succedent conjunction. After the application of the E-Cyclist implementation of the LK-rule $(\wedge R)$, referred to as (R.And), the implementation of the weakening LK-rule (Wk), denoted by (Weaken), is applied on the sequent of Node 5, then renamed the sequent of Node 7 by (Subst), which is the implementation of the LK-rule $(Subst)$, to the root sequent. E-Cyclist identifies Node 8 as a bud and stops its development. (N L.Unf.) is further applied on the sequent of Node 6 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 of Node 11 is true, while that of Node 12 has a conjunction in the succedent part. E-Cyclist applies (N L.Unf.) for the last time to perform a case analysis on $s(y)$ given as an argument to $N_4(s(y))$.

(Ex Falso) is the example of an E-Cyclist rule that has no equivalent in CLKID$^\omega$. Based on the assumption that the constructors 0 and s for naturals are *free*, i.e., there is no equality relation between them, it concludes that the sequent of Node 13 is true. The inductive succedent atom $Q_2(y, s(y))$ from the sequent of Node 14 unfolds. Then, (Weaken) is applied before renaming it with (Subst) in order to get a bud whose companion is Node 4. No inference rule is applied anymore and the resulting derivation tree is a pre-proof of $N_1(x), N_2(y) \vdash Q_1(x, y)$.

The script of the E-Cyclist pre-proof is different from other proof scripts solving the 'P and Q' problem [1, 10, 17]. For example, the FOL_{ID} pre-proof from [1] has three buds. Also, the E-Cyclist pre-proof is not optimized and can be simplified. For example, the last (L.Unf.) step, which adds one more time $N(y)$ to the antecedent part of the sequent of Node 12, can be omitted.

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

3 The Coq script of the pre-proof

It is well-known that 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 two declarations of the function symbols denoted by *zero* (for 0) and *succ* (for s), as well as for the definition of 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: $\forall\ x\ y,\ N\ x \to N\ y \to Q\ x\ y$ before trying to prove it as the theorem `true_0` by translating every pre-proof step from Fig. 2.

```
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 Coq proof *goals* can be thought of as sequents with a single formula in the 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 '==========='.

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 labeled 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 can reproduce 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. The variable `y` is further 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`, labeled 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 into 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 can be considered as a bud whose companion is the root goal if the variable z is renamed to y and the hypothesis `H1` is deleted. Its validation is canceled 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 in 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)
```

p2 is applied on the second goal. The new goal `Q y y /\ P y` is first identified 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 last goal, stating 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 of `true_0` is incomplete and lists the two remaining goals to be proved. They correspond to the sequents of Nodes 7 and 16 in the E-Cyclist pre-proof.

In the next section, we show that the pre-proof from Fig. 2 is a proof. Its soundness will be checked by the procedure implemented in E-Cyclist. The soundness arguments are employed further in Section 5 to prove by Noetherian induction the remaining two goals in order to complete the Coq proof of `true_0`.

4 The soundness 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 in the cyclic pre-proof, we have to assign a trace value for each of its nodes. We start by choosing an IAA from the sequent of the first node in p. For the other nodes n, we proceed recursively. Let t be the trace value and S the sequent of the node occurring just before n in p. The trace value for n is one of the IAAs from the sequent of n that is either i) t, or ii) one of the case descendents of t if the (L.Unf.) rule was applied on S using t as an active formula. The last case corresponds to a *progress point* in p. The global trace condition holds if, for each infinite path, 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,....[2] 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,....
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 [1] presenting a different CLKID$^\omega$ proof of the 'P and Q' conjecture.

The soundness procedure implemented in E-Cyclist was described in [6, 7]. It provides *sufficient* conditions to satisfy the global trace condition. Its two main steps are detailed in the following.

1. The normalization 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 $r \to b$, ends with a $(Subst)$-step which is also the *unique* $(Subst)$-step in p.

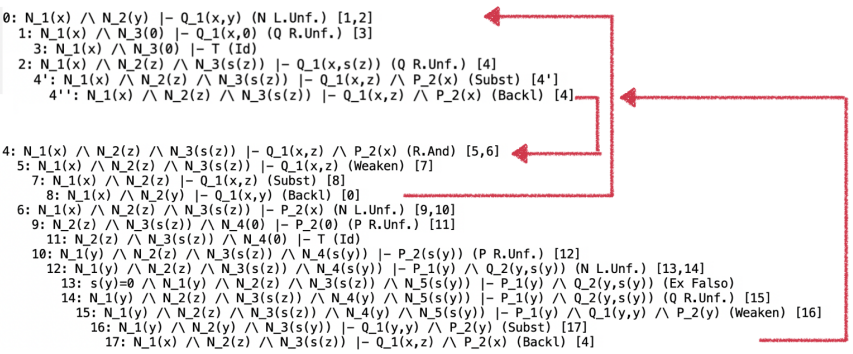

Fig. 5: The normalization of the pre-proof from Fig. 2.

Example 2 The result of the normalization operation generates two trees with the roots 0 and 4, where 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,

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

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 normalized pre-proof is depicted in Fig. 5.

Remark 1 Every infinite path in the E-Cyclist pre-proof is also reproduced in its normal form, by duplicating Node 4, and vice-versa, by deleting the occurrences introduced during the normalization step.

2. The ordering and derivability constraints step.

The next step is to associate a weight to each root r from the cycles of the normalized pre-proof, in terms of a multiset of IAAs from the sequent of r such that some derivability and ordering constraints are satisfied. The *derivability constraints* consist in checking that, for every rb-path $r \to b$ from a cycle, each IAA from the weight of the root companion of b can be derived from an IAA belonging to the weight of r, 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) $N_1x, N_1x, N_4sy, N_4sy, \overline{N_5sy}, N_5sy, N_3sy, N_3sz$ $(=)$.

Every 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. In this case, 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, we say that A_r and A_b are *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 $W(r) >_{mul} W(c(b))$, where $c(b)$ is the root companion of b and $W(r)$ (resp., $W(c(b))$) is the weight of r (resp., $c(b)$). The ordering $>_{mul}$ is the *trace-based multiset extension* of $>$, i.e., $W(r) >_{mul} W(c(b))$ if the equal IAAs from $W(r)$ and $W(c(b))$ are pairwisely deleted, to get $W(r)'$ and $W(c(b))'$, and either i) $W(c(b))'$ is empty and $W(r)'$ is not empty, or ii) for each IAA $a \in W(c(b))'$ there is an IAA $b \in W(r)'$ such that $b > a$. This definition of $>_{mul}$ was first introduced in [8] and is a substitution-free version of the $<_\pi$-derivability relations used in [6, 7].

```
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. 6: The weights computed by E-Cyclist.

Example 4 In Fig. 6, only the indexes of the IAAs are listed but one can notice that the heuristics implemented in E-Cyclist [8] proposed (a permutation of) $\{N_1x, N_1x, N_2y, N_2y\}$ and $\{N_1x, N_1x, N_2z, N_2z, N_3sz\}$ as weights of Node 0 and Node 4, respectively. In the same figure, the links of IAAs from $W(b)$ to $W(r)$ are checked, for each rb-path $r \to b$. For the rb-path $0 \to 4''$, $W(4'')$ is $W(4)$ since 4 is the companion of $4''$ and includes the IAAs indexed by 1, 2 and 3, i.e., N_1x, N_2z, and N_3sz. Their links to the IAAs from $W(0)$ are built as follows:

- N_1x of $W(4)$ is linked to N_1x of $W(0)$ by a non-progressing trace. This is denoted by `1 -> 1 [false]` in Fig. 6;
- N_2z of $W(4)$ is linked to N_2y of $W(0)$ by a progressing trace. This is denoted by `2 -> 2 [true]`;
- N_3sz of $W(4)$ is linked to N_2y of $W(0)$ by a non-progressing trace. This is denoted by `3 -> 2 [false]`.

The fact that a trace is progressing (resp., non-progressing) was denoted by `[true]` (resp., `[false]`).

Similarly, the traces along the rb-paths $4 \to 17$ and $4 \to 8$ can be checked if they are progressing or not.

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

- $0 \to 4''$: $\{N_1x, N_1x, N_2y, N_2y\} >_{mul} \{N_1x, N_1x, N_2z, N_2z, N_3sz\}$. 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. 6;
- $4 \to 17$: $\{N_1x, N_1x, N_2z, N_2z, N_3sz\} >_{mul} \{N_1x, N_1x, N_2z, N_2z, N_3sz\}$. 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, that are new w.r.t. [8].[3]

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

Theorem 1 (soundness) *The global trace condition holds for any pre-proof $\mathcal{P}$ for which the rb-paths $r \to b$ from the cycles of the normalized pre-proof of $\mathcal{P}$ satisfy $W(r) >_{mul} W(c(b))$.*

5 Building Noetherian induction proofs from cyclic proofs

In the following, we show how the E-Cyclist proofs can be represented as Noetherian induction proofs. The induction reasoning is based on the (first-order) formula-based instance of the Noetherian induction principle [18]:

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

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

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 and cyclic reasoning is based on the notion of $<_\pi$*-derivability* introduced in [6, 7] as an alternative to the E-Cyclist method to validate cyclic pre-proofs. The idea is to use the weights of the root nodes from cycles, built as explained in the previous section, to define $<^{wf}$ as a Noetherian ordering that satisfies some ordering constraints introduced in the next paragraph.

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. If $l_1 \in W(c(b))$ and $l_2 \in W(r)$, it can be noticed that $l_1[\delta] \equiv l_2[\theta]$ when l_1 equals l_2, where $\equiv$ is the syntactic equality. On the other hand, it is sufficient to show that $l_1[\delta] <^{wf} l_2[\theta]$ when $l_2 > l_1$. We further denote by σ_{id} the identity substitution that replaces variables by themselves, for which $W[\sigma_{id}] \equiv W$, for every weight W.

Example 5 For the running example, the weight computed by E-Cyclist for the root 0 (resp., 4) is $\{Nx, Nx, Ny, Ny\}$ (resp., $\{Nx, Nx, Nz, Nz, Nsz\}$). In practice, one can consider that all the inductive predicate symbols are equivalent, hence the

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

weights for the roots are multisets built only from their arguments. The weights 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 recursive path ordering (rpo) [19] built from any precedence over the symbols $\{s, 0\}$. Since every rpo is Noetherian, $<^{wf}$ is Noetherian.

We say that the rb-path $r \to b$ is *valid* if $W(c(b))[\delta] <^{wf}_{mul} W(r)[\theta]$. In the Noetherian induction setting, $W(c(b))[\delta]$ plays the role of the induction hypothesis. In [6, 7], it has been shown that the normalized pre-proof $\mathcal{P}$ is a cyclic proof if every rb-path from each cycle of $\mathcal{P}$ is valid.

In order to certify with Coq the Noetherian induction reasoning from FOL_{ID} pre-proofs, we apply the certification methodology for formula-based Noetherian induction proofs [10]. Firstly, $<^{wf}$ is defined as a rpo and $<^{wf}_{mul}$ as its multiset extension, denoted by *less*, by using the Coccinelle library [20, 21]. Each function symbol from the Coq specification is translated into a Coccinelle function symbol with its 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 index function defines the function symbol precedence used by the rpo ordering.

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*: $\forall$ *x*, (*zero* = *succ x*) $\rightarrow$ *False*.

The normalized pre-proofs can be represented as graphs with cycles, where the cycles occur in the non-singleton strongly connected components (SCCs) of the graphs. Each SCC may be built from a non-empty multiset of connected (sub)trees in the graph. Our procedure defines for each SCC S a conjecture, referred to as `all_true`, stating that all root formulas from the trees building S hold. It can be noticed that, for a given normalized pre-proof $\mathcal{P}$, if all `all_true` conjectures are proved, the proofs of the root sequents from $\mathcal{P}$ do not require induction reasoning; they can be completed by simply following paths leading to leaves or buds whose root companions are labeled by already proved sequents.

Let S_1 and S_2 be two SCCs from a normalized pre-proof. The `all_true` conjecture for S_1 will be proved *before* the `all_true` conjecture for S_2 if S_1 can be reached from S_2. The steps for proving an `all_true` conjecture are illustrated for the unique SCC S from the running example:

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

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

   `Definition` *F_0* : *type_LF_0_4*:= (`fun` *u1 u2* $\Rightarrow$ (*N u1* $\rightarrow$ *N u2* $\rightarrow$ *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* $\rightarrow$ *N u2* $\rightarrow$ *N* (*succ u2*) $\rightarrow$ *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 of the pre-proof belonging to the two trees that built the SCC.[4] 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* $\rightarrow$ $\forall$ *u1*, $\forall$ *u2*, ($\forall$ *F'*, *In F' LF_0_4* $\rightarrow$ $\forall$ *e1*, $\forall$ *e2*, *less* (*snd* (*F' e1 e2*)) (*snd* (*F u1 u2*)) $\rightarrow$ *fst* (*F' e1 e2*)) $\rightarrow$ *fst* (*F u1 u2*).

[4]The full Coq proof script of *F_0* in the main lemma is given in Appendix A.

where *In* and *fst* (resp., *snd*) are the membership predicate and the first (resp. second) projection of a pair.

4. prove the all_true conjecture, here denoted by *all_true_0_4* and stating that all formulas from the list of functionals hold, by using the main lemma and the Noetherian induction principle integrated into Coq:

   `Theorem` *all_true_0_4*: ∀ *F*, *In F LF_0_4* →
   ∀ *u1*: *T*, ∀ *u2*: *T*, *fst* (*F u1 u2*).

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

   `Theorem` *true_0*: ∀ *x y*, *N x* → *N y* → *Q x y*.
   `Theorem` *true_4*: ∀ *u1 u2*, *N u1* → *N u2* → *N* (*succ u2*) →
   Q u1 u2 ∧ *P u1*.

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 E-Cyclist proofs for the rest of the FOL_{ID} examples from the standard distribution of Cyclist and the 2-Hydra problem [12]. For convenience, the axioms for the inductive predicates used in the examples are:

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

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

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

$$ADD\ \{\quad N_1(x) \Rightarrow ADD(0,x,x)\ \mid\ ADD_1(x,y,z) \Rightarrow ADD(s(x),y,s(z))\ \};$$

$$p\ \{\quad true \Rightarrow p(0,0)\ \mid\ true \Rightarrow p(s(0),0)\ \mid\ true \Rightarrow p(x,s(0))\ \mid\ p_1(x,y) \Rightarrow p(s(x),s(s(y)))\ \mid\ p_1(s(y),y) \Rightarrow p(0,s(s(y)))\ \mid\ p_1(s(x),x) \Rightarrow p(s(s(x)),0)\ \};$$

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

Table 1 gives some statistics about the certification of the E-Cyclist proofs about several conjectures involving the above-mentioned inductive predicates. The columns show, from left to right: the problem number as it figures

out in the standard distribution of CYCLIST (except for the 2-Hydra problem listed at the bottom line), the number of the SCCs in the proof, its root sequents and their weights.

An important part of the Coq script built for the running example was shared with the other examples, such as the Coccinelle definition of the $<^{wf}_{mul}$ ordering. Most of the proof effort 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*: $\forall\ x\ y,\ (succ\ x = succ\ y) \rightarrow x = y.$

6 Conclusions and future work

We have applied a general procedure for certifying formula-based Noetherian induction reasoning in order to check FOL_{ID} cyclic proofs using Coq, such as those produced by the E-CYCLIST prover. The output is a Coq script proving a bunch of theorems and lemmas for which the deductive part translates the E-CYCLIST proof steps *without* using proof reconstruction techniques. This approach also allows for finding errors in cyclic proofs in a very precise way, at the level of proof steps. Most steps are converted in a one-to-one translation. Other steps require the application of more than one Coq tactics, such as the `(R.Unf.)` step applied on Node 14 in Fig. 2 to unfold a subsequent conjunct. For the translation of classical proof steps, the axiom of the excluded middle is required because the Coq proofs are constructive. For these reasons, the translation of E-CYCLIST pre-proofs to Coq scripts can be easily automated. On the other hand, the translation of the 'reasoning by induction' part, shown to be more elaborated and therefore more difficult to automatize, is based on the Noetherian induction principle integrated into Coq. We established a bridge between formula-based Noetherian induction and FOL_{ID} cyclic induction, by identifying a class of E-CYCLIST pre-proofs certifiable by Coq and for which the ordering $<^{wf}_{mul}$ is Noetherian and satisfies some ordering constraints. The implementation of our approach was based on Coq, but it can be easily adapted for any certifying proof environment that supports Noetherian induction reasoning, first-order logic, and inductive definitions, such as Isabelle [22].

The procedure has been successfully applied to certify the E-CYCLIST proofs built for all FOL_{ID} examples from the official distribution of CYCLIST, as well as for the 2-Hydra problem. For the Coq users, we plan to integrate the cyclic reasoning *directly* in Coq. In this direction, we intend i) to build a Coq plug-in for identifying the bud-companion relations from the Coq scripts, and ii) to automatically check the Noetherian induction arguments by using an approach similar to [10] that allowed in the past for automatically converting SPIKE proofs to Coq scripts and for defining Coq tactics to automatize the

#	Theorem	# SCC	Root Sequents	Weight
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	$N_1(x) \wedge N_2(y) \wedge ADD_3(x, y, z) \vdash N(z)$	$\{x, z\}$
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.

implicit induction reasoning [13]. We also intend to explore the relationship between the size-change and formula-based Noetherian induction principles in order to apply our approach to the certification of size-change cyclic proofs, such as those produced by the CycleQ prover [16].

Declarations

Funding and/or Conflicts of interests/Competing interests

The author declares that there are no funding and/or conflicts of interests/-competing interests that are relevant to the content of this article.

Data Availability

The datasets generated during and/or analyzed during the current study, in particular, the full Coq specifications and proof scripts, are archived and made available at https://members.loria.fr/SStratulat/files/ECyclist-coq-certification.zip

References

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

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

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

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

[5] Michel, M.: Complementation is more difficult with automata on infinite words. Technical report, CNET (1988)

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

[7] Stratulat, S.: Validating back-links of FOL$_{\mathrm{ID}}$ cyclic pre-proofs. In: Berardi, S., van Bakel, S. (eds.) CL&C'18 (Seventh International Workshop on Classical Logic and Computation). EPTCS, pp. 39–53 (2018). https://doi.org/10.4204/EPTCS.281.4

[8] Stratulat, S.: E-Cyclist: Implementation of an efficient validation of FOL$_{\mathrm{ID}}$ cyclic induction reasoning. In: Kutsia, T. (ed.) 9th International Symposium on Symbolic Computation in Software Science. Electronic Proceedings in Theoretical Computer Science, vol. 342, pp. 129–135 (2021). https://doi.org/10.4204/EPTCS.342.11

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

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

[11] Stratulat, S.: SPIKE, an automatic theorem prover – revisited. In: SYNASC 2020: Proceedings of the 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, pp. 93–96. IEEE Computer Society, New York (2020). https://doi.org/10.1109/SYNASC51798.2020.00025

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

[13] Henaien, A., Stratulat, S.: Performing implicit induction reasoning with certifying proof environments. In: Bouhoula, A., Ida, T., Kamareddine, F. (eds.) Proceedings Fourth International Symposium on Symbolic Computation in Software Science, Gammarth, Tunisia, 15-17 December 2012. Electronic Proceedings in Theoretical Computer Science, vol. 122, pp. 97–108 (2013). https://doi.org/10.4204/EPTCS.122.9

[14] Lee, C.S., Jones, N.D., Ben-Amram, A.M.: The size-change principle for program termination. In: POPL '01: Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, vol. 36, pp. 81–92. ACM Press, New York, NY, USA (2001). https://doi.org/10.1145/360204.360210

[15] Fogarty, S., Vardi, M.Y.: Büchi complementation and size-change termination. In: Kowalewski, S., Philippou, A. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, 15th International Conference, TACAS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5505, pp. 16–30.

Springer, New York (2009). https://doi.org/10.1007/978-3-642-00768-2_2

[16] Jones, E., Ong, C.-L., Ramsay, S.J.: Cycleq: an efficient basis for cyclic equational reasoning. In: Jhala, R., Dillig, I. (eds.) PLDI '22: 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation, San Diego, CA, USA, June 13 - 17, 2022, pp. 395–409. ACM, New York (2022). https://doi.org/10.1145/3519939.3523731

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

[18] Stratulat, S.: 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, pp. 326–352. EasyChair, Manchester (2012). https://doi.org/10.29007/nsx4

[19] Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, Cambridge, England (1998). https://doi.org/10.1017/CBO9781139172752

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

[21] Contejean, E., Paskevich, A., Urbain, X., Courtieu, P., Pons, O., Forest, J.: A3PAT, an approach for certified automated termination proofs. In: Gallagher, J.P., Voigtländer, J. (eds.) PEPM - Proceedings of the 2010 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation, PEPM 2010, Madrid, Spain, pp. 63–72. ACM, New York (2010). https://doi.org/10.1145/1706356.1706370

[22] Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL — A Proof Assistant for Higher-Order Logic. Lecture Notes in Computer Science, vol. 2283. Springer, New York (2002). https://doi.org/10.1007/3-540-45949-9

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, $W(r_1) >_{mul} W(c(b_1))$, and $W(r_2) >_{mul} W(c(b_2))$. We will try to prove that $W(r_1) >_{mul} W(c(b_2))$.

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

We show that there should be a trace along the path $[r_1, \ldots, b_1, r_2, \ldots, b_2]$. Since $W(c(b_2))$ is not empty, let $l_3 \in W(c(b_2))$. By the definition of $<_{mul}$, there exists an IAA $l_2 \in W(r_2)$ such that $l_2 > l_3$ or $l_2 = l_3$. Similar reasoning can be applied to l_2 as we have done for l_3 to conclude that there is $l_1 \in W(r_1)$ 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 $W(r_1) >_{mul} W(c(b_2))$. As previously, for each IAA $l_3 \in W(c(b_2))$ there is an IAA $l_2 \in W(r_2)$ such that $l_2 > l_3$ or $l_2 = l_3$, and an IAA $l_1 \in W(r_1)$ 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 $W(r_1) >_{mul} W(c(b_2))$ when there is at least one $l_3 \in W(c(b_2))$ for which there exists $l_1 \in W(r_1)$ such that $l_1 > l_3$. After the pairwise deletion of equal IAAs from $W(r_1)$ and $W(c(b_2))$, resulting $W(r_1)'$ and $W(c(b_2))'$, we have that $l_3 \in W(c(b_2))'$, $l_1 \in W(r_1)'$ and $l_1 > l_3$. By the definition of $>_{mul}$, we have that $W(r_1) >_{mul} W(c(b_2))$, because the same reasoning can be done for any other IAA from $W(c(b_2))'$ as for l_3.

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

□

B.2 Proof of Theorem 1

Proof Let be a pre-proof whose normalized pre-proof is denoted by $\mathcal{P}$ and for which every rb-path $r \to b$ belonging to a cycle satisfies $W(r) >_{mul} W(c(b))$. Let also p_0 be an infinite path in $\mathcal{P}$. Let p be the infinite path from $\mathcal{P}$ built from p_0 by duplicating the nodes as shown during the normalization process from Section 4. By construction, the path p is built starting from some point only from the concatenations of rb-paths from cycles from $\mathcal{P}$. 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 $W(r_1) >_{mul} W(c(b_1))$. Since $<_{mul}$ is transitive, by Lemma 1, we have that $W(r) >_{mul} W(c(b))$ for each sub-path $[r, \ldots, r, \ldots, b]$. Since $W(r)$ 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 weights for each root of the trace along p' is finite, there is a sub-path $[r, \ldots, b]$ defined as above whose traces have no progress points and satisfies $W(r) >_{mul} W(c(b))$. After the pairwise deletion of equal IAAs from $W(r)$ and $W(c(b))$, to get $W(r)'$ and $W(c(b))'$, we perform a case analysis by considering whether $W(c(b))'$ is empty or not.

- $W(c(b))'$ is not empty. By the definition of $>_{mul}$, there is an IAA $l \in W(c(b))'$ for which there is an IAA $l' \in W(r)'$ such that $l' > l$. Hence, the trace leading l' to l has at least one progress point. Contradiction.
- $W(c(b))'$ is empty. By similar reasoning as given in the proof of Lemma 1, the cardinality of $W(r)$ is greater than that of $W(c(b))$. But $W(r) = W(c(b))$ 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. On the other hand, p_0 can be built from p by deleting the extra nodes added during the normalization process, so it has an infinitely progressing trace starting from some point.

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

□