Efficient Validation of FOL ID Cyclic Induction Reasoning

Sorin Stratulat

Efficient Validation of $\mathrm{FOL_{ID}}$ Cyclic Induction Reasoning

Sorin Stratulat

Université de Lorraine, CNRS, LORIA, F-57000 Metz, France
sorin.stratulat@univ-lorraine.fr

Abstract

Checking the soundness of the cyclic induction reasoning for first-order logic with inductive definitions ($\mathrm{FOL_{ID}}$) is decidable but the standard checking method is based on a doubly exponential complement operation for Büchi automata. We present a polynomial method 'semi-deciding' this problem; its most expensive steps recall the comparisons with multiset path orderings. In practice, it has been integrated in the Cyclist prover and successfully checked all the proofs included in its distribution.

$\mathrm{FOL_{ID}}$ cyclic proofs may also be hard to certify. Our method helps to represent the cyclic induction reasoning as being well-founded, where the ordering constraints are automatically built from the analysis of the proofs. Hence, it creates a bridge between the two induction reasoning methods and opens the perspective to use the certification methods adapted for well-founded induction proofs.

Introduction. Cyclic pre-proofs for the classical first-order logic with inductive predicates ($\mathrm{FOL_{ID}}$) have been extensively studied in [1, 2, 4]. They are finite sequent-based derivations where some terminal nodes, called *buds*, are labelled with sequents already occurring in the derivation, called *companions*. Bud-companion (BC) relations, graphically represented as *back-links*, are described by an induction function attached to the derivation, such that only one companion is assigned to each bud, but a node can be the companion of one or several buds. The pre-proofs can be viewed as digraphs whose cycles, if any, are introduced by the BC-relations.

It is easy to build unsound pre-proofs, for example by creating a BC-relation between the nodes labelled by the sequents from a stuttering step. The classical soundness criterion is the *global trace condition*. Firstly, the paths are annotated by traces built from inductive antecedent atoms (IAAs) found on the lhs of the sequents in the path, then it is shown that for every infinite path p in the cyclic derivation of a false sequent, there is some trace following p such that all successive steps starting from some point are decreasing and certain steps occurring infinitely often are strictly decreasing w.r.t. some semantic ordering. We say that a *progress point* happens in the trace when a step is strictly decreasing. A *proof* is a pre-proof if every infinite path has an infinitely progressing trace starting from some point.

The standard checking method [2] of the global trace condition is decidable but based on a doubly exponential complement operation for Büchi automata [5]. It has been implemented in the Cyclist prover [3] and experiments showed that the soundness checking can take up to 44% of the proof time. On the other hand, a less costly, polynomial-time, checking method has been presented in [7, 9].[1] The pre-proof to be checked is firstly normalized into a digraph consisting of a set of derivation trees to which is attached an extended induction function. The resulting digraph counts among its roots the companions and the root of the pre-proof to be checked. The normalized pre-proof is a proof if every strongly connected component (SCC) of the digraph satisfies some ordering constraints, similar to those used for certifying cyclic Noetherian induction proofs [8].

[1] [6] tackles a similar question, although from a more theoretical viewpoint.

Implementation. The method has been implemented in Cyclist. Cyclist builds the pre-proofs using a depth-first search strategy that aims at closing open nodes as quickly as possible. Whenever a new cycle is built, model-checking techniques provided by an external model checker are called to validate it. If the validation result is negative, the prover backtracks by trying to find another way to build new cycles. Hence, it may happen that the model checker be called several times during the construction of a pre-proof.

To each root r from the digraph $\mathcal{P}$ of a normalized pre-proof tree-set, the method attaches a measure $\mathcal{M}(r)$ consisting of a multiset of IAAs of the sequent labelling r, denoted by $S(r)$. One of the challenges is to find the good measures such that the ordering constraints be satisfied. A procedure for computing these measure values is given by Algorithm 1.

Algorithm 1 GenOrd($\mathcal{P}$): to each root r of $\mathcal{P}$ is attached a measure $\mathcal{M}(r)$

for all root r **do**
  $\mathcal{M}(r) := \emptyset$
end for
for all rb-path $r \to b$ from a non-singleton SSC **do**
  if there is a trace between an IAA A of $S(b)$ and an IAA A' of $S(r)$ **then**
    add A to $\mathcal{M}(rc)$ and A' to $\mathcal{M}(r)$, where rc is the companion of b
  end if
end for

Firstly, the measures attached to each root are empty sets. Then, for each root-bud (rb) path from a cycle, denoted by $r \to b$, and for every trace along $r \to b$, leading some IAA of $S(r)$ to another IAA of $S(b)$, we add the corresponding IAAs to the measures of r and the companion of b, respectively. Since the number of rb-paths is finite, Algorithm 1 terminates.

Algorithm 1 may compute measure values that do not pass the comparison test for some non-singleton SCCs that are validated by the model checker. For this case, we considered an improvement consisting of the incremental addition of IAAs from a root sequent that are not yet in the measure value of the corresponding root r. Since the validating orderings are multiset extensions of multiset path orderings, such an addition does not affect the comparison value along the rp-paths starting from r. On the other hand, it may affect the comparison tests for the rp-paths ending in the companions of r. This may duplicate some IAAs from the value measure of the roots from the rp-paths leading to these companions. The duplicated IAAs have to be processed as any incrementally added IAA, and so on, until no changes are performed anymore.

Table 1 illustrates some statistics about the proofs of the conjectures considered in Table 1 from [3], checked with the standard as well as our improved method. The IAAs are indexed in Cyclist to facilitate the construction of traces; the way they are indexed influence how the pre-proofs are built. The column labelled 'Time-E' is the proof time measured in milliseconds with our method. Similarly, the 'Time' column displays the proof time using the standard method, while 'SC%' shows the percentage of time taken to check soundness using the model checker. 'Depth' shows the depth of the proof, 'Nodes' the number of nodes in the proof, and 'Bckl.' the number of back-links in the proof. The last column shows the number of calls to the model checker as (calls on unsound proof)/(total calls) when our method is not used. The proofs have been performed on a MacBook Pro featuring a 2,7 GHz Intel Core i7 processor and 16 GB of memory. We can notice that, by using our method, the execution time is reduced by a factor going from 1.43 to 5.

Even when using the improved version of Algorithm 1, the method may propose measure

Theorem	**Time-E**	Time	SC%	Depth	Nodes	Bckl.	Uns./All
$O_1x \vdash Nx$	**2**	7	61	2	9	1	0/1
$E_1x \vee O_2x \vdash Nx$	**4**	11	63	3	19	2	0/4
$E_1x \vee O_1x \vdash Nx$	**2**	9	77	2	13	2	2/5
$N_1x \vdash Ox \vee Ex$	**3**	7	52	2	8	1	0/1
$N_1x \wedge N_2y \vdash Q(x, y)$	**297**	425	40	4	19	3	168/181
$N_1x \vdash Add(x, 0, x)$	**1**	5	76	1	7	1	0/1
$N_1x \wedge N_2y \wedge Add_3(x, y, z) \vdash Nz$	**8**	14	38	2	8	1	4/5
$N_1x \wedge N_2y \wedge Add_3(x, y, z) \vdash Add(x, sy, sz)$	**15**	22	32	2	14	1	9/10
$N_1x \wedge N_2y \vdash R(x, y)$	**266**	484	48	4	35	5	149/170

Table 1: Statistics for CYCLIST proofs checked with the standard and our methods.

values that do not pass the comparison tests. Indeed, this was happened once, while proving $N_1x \wedge N_2y \vdash R(x, y)$. Hopefully, the prover backtracked and finally found the same proof as that built using the model checker. The source code of the implementation can be downloaded at `https://members.loria.fr/SStratulat/files/e-cyclist.zip`

References

[1] J. Brotherston. Cyclic proofs for first-order logic with inductive definitions. In *Proceedings of TABLEAUX-14*, volume 3702 of *LNAI*, pages 78–92. Springer-Verlag, 2005.

[2] J. Brotherston. *Sequent Calculus Proof Systems for Inductive Definitions*. PhD thesis, University of Edinburgh, November 2006.

[3] J. Brotherston, N. Gorogiannis, and R. L. Petersen. A generic cyclic theorem prover. In *APLAS-10 (10th Asian Symposium on Programming Languages and Systems)*, volume 7705 of *LNCS*, pages 350–367. Springer, 2012.

[4] J. Brotherston and A. Simpson. Sequent calculi for induction and infinite descent. *Journal of Logic and Computation*, 21(6):1177–1216, 2011.

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

[6] R. Nollet, A. Saurin, and C. Tasson. Pspace-completeness of a thread criterion for circular proofs in linear logic with least and greatest fixed points. In S. Cerrito and A. Popescu, editors, *TABLEAUX'2019*, pages 317–334. Springer International Publishing, 2019.

[7] S. Stratulat. Cyclic proofs with ordering constraints. In R. A. Schmidt and C. Nalon, editors, *TABLEAUX 2017 (26th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods)*, volume 10501 of *LNAI*, pages 311–327. Springer, 2017.

[8] S. Stratulat. Mechanically certifying formula-based Noetherian induction reasoning. *Journal of Symbolic Computation*, 80, Part 1:209–249, 2017.

[9] S. Stratulat. Validating back-links of $\mathrm{FOL}_{\mathrm{ID}}$ cyclic pre-proofs. In S. Berardi and S. van Bakel, editors, *CL&C'18 (Seventh International Workshop on Classical Logic and Computation)*, number 281 in EPTCS, pages 39–53, 2018.