HYPNO: Theorem Proving with Hypersequent Calculi for Non-normal Modal Logics (System Description)

Tiziano Dalmonte[1], Nicola Olivetti[1], and Gian Luca Pozzato[2](✉)

[1] Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
{tiziano.dalmonte,nicola.olivetti}@lis-lab.fr

[2] Dipartimento di Informatica, Universitá degli Studi di Torino, Turin, Italy
gianluca.pozzato@unito.it

Abstract. We present HYPNO (HYpersequent Prover for NOn-normal modal logics), a Prolog-based theorem prover and countermodel generator for non-normal modal logics. HYPNO implements some hypersequent calculi recently introduced for the basic system **E** and its extensions with axioms M, N, and C. It is inspired by the methodology of leanT^AP, so that it does not make use of any ad-hoc control mechanism. Given a formula, HYPNO provides either a proof in the calculus or a countermodel, directly built from an open saturated hypersequent. Preliminary experimental results show that the performances of HYPNO are very promising with respect to other theorem provers for the same class of logics.

Keywords: Non-normal modal logics · Hypersequent calculi · Prolog

1 Introduction

Non-Normal Modal Logics (NNMLs for short) are a generalization of ordinary modal logics that do not satisfy some axioms or rules of minimal normal modal logic **K**. They have been studied since the seminal works by C.I. Lewis, Scott, Lemmon, and Chellas (for an introduction see [3]), and along the years have gained interest in several areas such as epistemic, deontic, and agent reasoning among others [1,7,12–14]. NNMLs are characterised by the neighbourhood semantics. In [4,6], a variant of it is presented, called bi-neighbourhood semantics, this variant is more suitable for logics lacking the monotonicity property, although equivalent to the standard one.

This work has been partially supported by the ANR project TICAMORE ANR-16-CE91-0002-01 and by the INdAM project GNCS 2019 "METALLIC #2".

Not many theorem provers for NNMLs have been developed so far.[1] In [8] optimal decision procedures are presented for the whole cube of NNMLs; these procedures reduce a validity/satisfiability checking in NNMLs to a set of SAT problems and then call an efficient SAT solver. Despite undoubtably efficient, they do not provide explicitly "proofs", nor countermodels. A theorem prover for logic **EM** based on a tableaux calculus very similar to the one in [10], is presented in [9]: the system, is implemented in ELAN and handles also more complex Coalition Logic and Alternating Time Temporal logic. In [11] it is presented a Prolog implementation of a NNML containing both the $[\forall\forall]$ and the $[\exists\forall]$ modality; its $[\exists\forall]$ fragment coincides with the logic **EM**, also covered by HYPNO. Finally in [5] it is presented PRONOM, a theorem prover for the whole range of NNMLs, which implements *labelled* sequent calculi in [6]; PRONOM provides both proofs and countermodels in the mentioned bi-neighbourhood semantics.

In this paper we describe HYPNO (HYpersequent Prover for NOn-normal modal logics) a Prolog theorem prover for the whole cube of NNMLs. The prover HYPNO implements the optimal calculi for NNMLs recently introduced in [4]. These calculi handle hypersequents, where a hypersequent represents intuitively a metalogical disjunction of sequents; sequents in themselves can be interpreted as formulas of the language. While the hypersequent structure is not strictly needed for proving formulas, it ensures a direct computation of a countermodel from *one* failed proof-branch. Consequently, the prover takes as input a formula and either returns a proof or a countermodel of it in the bi-neighbourhood semantics mentioned above. The Prolog implementation closely corresponds to the calculi: each rule is encoded by a Prolog clause of a `prove` predicate. This correspondence ensures in principle both the soundness and completeness of the theorem prover. Termination of proof search is obtained by preventing redundant application of rules. Although there are no benchmarks in the literature, the performance seems promising, in particular it outperforms the theorem prover PRONOM based on labelled calculi.

The program HYPNO as well as all the Prolog source files, including those used for the performance evaluation, are available for free usage and download at http://193.51.60.97:8000/HYPNO/.

2 Axioms, Semantics, and Hypersequent Calculi

We present first the axiomatization and semantics of NNMLs of the classical cube and then the hypersequent calculi implemented by HYPNO.

Given a countable set of propositional variables Atm, the formulas of the language $\mathcal{L}$ of NNMLs are built as follows: $A ::= p \mid \bot \mid \top \mid A \vee A \mid A \wedge A \mid A \rightarrow A \mid \Box A$, where $p \in \mathsf{Atm}$. The minimal NNML **E** is defined in $\mathcal{L}$ by extending classical propositional logic with the rule RE below. The systems of the classical cube are then obtained by adding to **E** any combination of axioms M, C, and N. We obtain in this way eight distinct logics (see the *classical cube*, below on the right), where the top system **EMCN** coincides with normal modal logic **K**.

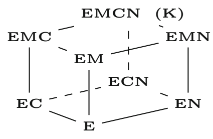
[1] We only mention here *implemented* systems, for a discussion on proof systems for NNMLs we refer to [4,6] and references therein.

$$\text{RE}\ \frac{A \leftrightarrow B}{\Box A \leftrightarrow \Box B}$$

M $\Box(A \wedge B) \to \Box A$

C $\Box A \wedge \Box B \to \Box(A \wedge B)$

N $\Box\top$



Coming to the semantics, we consider the bi-neighbourhood models [6]. As a difference with standard neighbourhood semantics, in the bi-neighbourhood one, worlds are equipped with sets of *pairs* of neighbours which can be thought as lower and upper approximations of neighbourhood in the standard semantics.

Definition 1. *A* bi-neighbourhood model *is a tuple* $\mathcal{M} = \langle \mathcal{W}, \mathcal{N}_b, \mathcal{V} \rangle$*, where* $\mathcal{W}$ *is a non-empty set of worlds,* $\mathcal{V}$ *is a valuation function, and* $\mathcal{N}_b$ *is a bi-neighbourhood function* $\mathcal{W} \longrightarrow \mathcal{P}(\mathcal{P}(\mathcal{W}) \times \mathcal{P}(\mathcal{W}))$*. We say that* $\mathcal{M}$ *is a* M-model *if* $(\alpha, \beta) \in \mathcal{N}_b(w)$ *implies* $\beta = \emptyset$*, it is a* N-model *if for all* $w \in \mathcal{W}$ *there is* $\alpha \subseteq \mathcal{W}$ *such that* $(\alpha, \emptyset) \in \mathcal{N}_b(w)$*, and it is a* C-model *if* $(\alpha_1, \beta_1), (\alpha_2, \beta_2) \in \mathcal{N}_b(w)$ *implies* $(\alpha_1 \cap \alpha_2, \beta_1 \cup \beta_2) \in \mathcal{N}_b(w)$*. The forcing relation for boxed formulas is as follows:* $\mathcal{M}, w \Vdash \Box A$ *if and only if there is* $(\alpha, \beta) \in \mathcal{N}_b(w)$ *s.t.* $\alpha \subseteq [A] \subseteq \mathcal{W} \setminus \beta$*, where* $[A]$ *denotes the truth set of* A *in* $\mathcal{M}$*.*

Bi-neighbourhood models can be easily transformed into equivalent standard neighbourhood models, and vice versa. Moreover, bi-neighbourhood semantics characterises the whole cube of NNMLs [6], in the sense that a formula A is derivable in $\mathbf{E}(\mathbf{M}/\mathbf{C}/\mathbf{N})$ if and only if it is valid in all bi-neighbourhood (M/N/C)-models of the corresponding class.

The hypersequent calculi for NNMLs implemented by HYPNO are introduced in [4]. Their syntax is as follows: a *block* is a structure $\langle \Sigma \rangle$, where Σ is a multiset of formulas of $\mathcal{L}$. A *sequent* is a pair $\Gamma \Rightarrow \Delta$, where Γ is a multiset of formulas and blocks, and Δ is a multiset of formulas. A *hypersequent* is a multiset $S_1 \mid ... \mid S_n$, where $S_1, ..., S_n$ are sequents. Single sequents can be interpreted into the language as: $i(A_1, ..., A_n, \langle \Sigma_1 \rangle, ..., \langle \Sigma_m \rangle \Rightarrow B_1, ..., B_k) = \bigwedge_{i \leq n} A_i \wedge \bigwedge_{j \leq m} \Box \bigwedge \Sigma_j \to \bigvee_{\ell \leq k} B_\ell$. We say that a sequent S is *valid* in a bi-neighbourhood model $\mathcal{M}$ (written $\mathcal{M} \models S$) if for all $w \in \mathcal{M}$, $\mathcal{M}, w \Vdash i(S)$. Moreover, a hypersequent H is *valid* in $\mathcal{M}$ ($\mathcal{M} \models H$) if $\mathcal{M} \models S$ for some $S \in H$, and it is valid in (M/C/N-)models if it is valid in all models of that kind.

The calculi implemented by HYPNO are a minor variant of the ones in [4]: they contain an additional arrow $\Rrightarrow$ used to represent that the formulas on the left of $\Rrightarrow$ entails the *conjunction* (rather than their disjunction) of the formulas on its right. By this modification, all rules of the calculi are at most binary; the equivalence of the modified calculi with the original ones in [4] is straightforward.

The hypersequent calculi are defined by the rules in Fig. 1 (for propositional rules we only show the initial sequents and the rules for implication). In particular: $\mathcal{H}_{\mathsf{E}}$:= propositional rules + $\Box_{\mathsf{L}}$ + $\Box_{\mathsf{R}}$ + $\Rrightarrow_1$ + $\Rrightarrow_2$; $\mathcal{H}_{\mathsf{EN}}$:= $\mathcal{H}_{\mathsf{E}}$ + N; $\mathcal{H}_{\mathsf{EC}}$:= $\mathcal{H}_{\mathsf{E}}$ + C; $\mathcal{H}_{\mathsf{ECN}}$:= $\mathcal{H}_{\mathsf{E}}$ + C + N; $\mathcal{H}_{\mathsf{M}}$:= propositional rules + $\Box_{\mathsf{L}}$ + ${}_{\mathsf{M}}\Box_{\mathsf{R}}$;

$$\mathsf{init}\ \frac{}{G \mid p, \Gamma \Rightarrow \Delta, p} \qquad \bot_{\mathsf{L}}\ \frac{}{G \mid \bot, \Gamma \Rightarrow \Delta} \qquad \top_{\mathsf{R}}\ \frac{}{G \mid \Gamma \Rightarrow \Delta, \top}$$

$$\to_{\mathsf{L}}\ \frac{G \mid A \to B, \Gamma \Rightarrow \Delta, A \qquad G \mid B, A \to B, \Gamma \Rightarrow \Delta}{G \mid A \to B, \Gamma \Rightarrow \Delta} \qquad \to_{\mathsf{R}}\ \frac{G \mid A, \Gamma \Rightarrow \Delta, A \to B, B}{G \mid \Gamma \Rightarrow \Delta, A \to B}$$

$$\Box_{\mathsf{L}}\ \frac{G \mid \langle A \rangle, \Box A, \Gamma \Rightarrow \Delta}{G \mid \Box A, \Gamma \Rightarrow \Delta} \qquad {}_{\mathsf{M}}\Box_{\mathsf{R}}\ \frac{G \mid \langle \Sigma \rangle, \Gamma \Rightarrow \Delta, \Box B \mid \Sigma \Rightarrow B}{G \mid \langle \Sigma \rangle, \Gamma \Rightarrow \Delta, \Box B}$$

$$\Box_{\mathsf{R}}\ \frac{G \mid \langle \Sigma \rangle, \Gamma \Rightarrow \Delta, \Box B \mid \Sigma \Rightarrow B \qquad G \mid \langle \Sigma \rangle, \Gamma \Rightarrow \Delta, \Box B \mid B \Rrightarrow \Sigma}{G \mid \langle \Sigma \rangle, \Gamma \Rightarrow \Delta, \Box B}$$

$$\Rrightarrow_1\ \frac{G \mid A \Rightarrow B}{G \mid A \Rrightarrow B} \qquad \Rrightarrow_2\ \frac{G \mid A \Rightarrow B \qquad G \mid A \Rrightarrow \Sigma}{G \mid A \Rrightarrow B, \Sigma}\ |\Sigma| \geq 1$$

$$\mathsf{N}\ \frac{G \mid \langle \top \rangle, \Gamma \Rightarrow \Delta}{G \mid \Gamma \Rightarrow \Delta} \qquad \mathsf{C}\ \frac{G \mid \langle \Sigma, \Pi \rangle, \langle \Sigma \rangle, \langle \Pi \rangle, \Gamma \Rightarrow \Delta}{G \mid \langle \Sigma \rangle, \langle \Pi \rangle, \Gamma \Rightarrow \Delta}$$

Fig. 1. Rules of $\mathcal{H}_{\mathsf{E}^\star}$.

$\mathcal{H}_{\mathsf{MN}} := \mathcal{H}_{\mathsf{M}} + \mathsf{N}$; $\mathcal{H}_{\mathsf{MC}} := \mathcal{H}_{\mathsf{M}} + \mathsf{C}$; and $\mathcal{H}_{\mathsf{MCN}} := \mathcal{H}_{\mathsf{M}} + \mathsf{C} + \mathsf{N}$. In the following, we denote by $\mathcal{H}_{\mathsf{E}^\star}$ any extension of $\mathcal{H}_{\mathsf{E}}$.

3 Design of HYPNO

The prover HYPNO implements the calculi of Fig. 1. It is inspired by the "lean" methodology of leanT^AP [2], even if it does not follow its style in a rigorous manner. The program comprises a set of clauses, each one of them implementing a rule or an axiom of the mentioned calculi. The proof search is provided for free by the mere depth-first search mechanism of Prolog, without any additional ad hoc mechanism. Before presenting in details the code of the theorem prover, let us discuss a general design choice.

As mentioned, HYPNO searches for a derivation of an input formula and in case of failure, on demand, it produces a countermodel of it. The proof search procedure is implemented by a predicate `terminating_proof_search` which tries to generate a derivation of the given input formula. In case it fails, on demand by the user, another predicate `build_saturated_branch` is invoked that computes an open saturated branch from which a countermodel is extracted. The predicate `build_saturated_branch` is in some sense "dual" of the proof search one. We have chosen to implement a two-phase computation, instead of a single one taking care of both tasks, for the following reason: a single-phase procedure would need to carry over all information for extracting a countermodel anyway; this information would be completely useless in case of a successful derivation and would unacceptably overload proof-search. As matter of fact, the time spent to "recompute" the saturated branch is negligible with respect to the overload of a proof-search procedure storing also information for countermodel extraction. By this choice we get a simpler and more readable code, and of course, more suited for countermodel generation only "on demand".

HYPNO represents an hypersequent with a Prolog list whose elements are Prolog terms of the form `singleSeq([Gamma,Delta],Additional)`, each one representing a sequent in the hypersequent. `Gamma`, `Delta`, and `Additional` are in turn Prolog lists: `Gamma` and `Delta` represent the left side and the right side of the single sequent, respectively, whereas `Additional` keeps track of the rules already applied to each sequent in order to ensure termination by avoiding multiple redundant applications of the same rule to a given hypersequent. Elements of `Gamma` and `Delta` are either formulas or Prolog lists representing blocks. Symbols $\top$ and $\bot$ are represented by constants `true` and `false`, respectively, whereas connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, and $\Box$ are represented by `-`, `^`, `?`, `->`, and `box`. The symbol of provability $\Rrightarrow$ in systems with axiom C is represented by `=>`. Propositional variables are represented by Prolog atoms. As an example, the Prolog list

```
[singleSeq( [[box (a ^ c), [true], [a,c]], [a, b, a -> b, box b]],
 [n, right(a -> b), apdR([a,c],b)]), singleSeq ([[P], [P]] ,[ ])]
```

is used to represent the hypersequent $\Box(A \wedge C), \langle\top\rangle, \langle A, C\rangle \Rightarrow A, B, A \vee B, \Box B \mid P \Rightarrow P$, to which the rules N, $\vee_{\mathsf{R}}$ and $\Box_{\mathsf{R}}$ have been already applied, the last one by using the block $\langle A, C\rangle$ and the formula $\Box B$ as the principal formulas. In turn, no rule has been applied to $P \Rightarrow P$ (the list `Additional` is empty).

Given a NNML formula F represented by the Prolog term `f`, HYPNO executes the main predicate of the prover, called **prove**[2], whose only two clauses implement the functioning of HYPNO: the first clause checks whether F is valid and, in case of a failure, the second one enables the graphical interface to invoke a predicate called `counter` to compute a model falsifying F. In detail, the predicate `prove` first checks whether the formula is valid by executing the predicate:

```
terminating_proof_search(Hyper, ProofTree).
```

This predicate succeeds if and only if the hypersequent represented by the list `Hyper` is derivable in $\mathcal{H}_{\mathsf{E}^\star}$. When it succeeds, the output term `ProofTree` matches with a representation of the derivation found by the prover. As an example, in order to prove that the sequent $\Box(A \wedge (B \vee C)) \Rightarrow \Box((A \wedge B) \vee (A \wedge C))$ is valid in **E**, one queries HYPNO with the goal:

```
terminating_proof_search([singleSeq([[box (a ^ (b ? c))], [box ((a ^ b) ?
                         (a ^ c))]], [  ]), ProofTree).
```

Each clause of `terminating_proof_search` implements an axiom or rule of the sequent calculi $\mathcal{H}_{\mathsf{E}^\star}$. To search for a derivation of a sequent $\Gamma \Rightarrow \Delta$, HYPNO proceeds as follows. First of all, if $\Gamma \Rightarrow \Delta$ is an instance of an axiom, then the goal will succeed immediately by using one of the clauses implementing the axioms. As an example, the clause implementing init is as follows:

```
terminating_proof_search(Hyper,tree(axiom,PrintableHyper,no,no)):-
   member(singleSeq([Gamma,Delta],_),Hyper),
   member(P,Gamma), member(P,Delta),!,
   extractPrintableSequents(Hyper,PrintableHyper).
```

[2] The user can run HYPNO without using the interface of the web application. To this aim, he just needs to invoke the goal `prove(f)`.

The auxiliary predicate `extractPrintableSequents` is used just for a graphical rendering of the hypersequent. If $\Gamma \Rightarrow \Delta$ is not an instance of the axioms, then the first applicable rule will be chosen, e.g. if `Gamma` contains a list `Sigma` representing a block $\langle \Sigma \rangle \in \Gamma$, and `Delta` contains `box b` representing that $\Box B \in \Delta$, then the clause for $\Box_{\mathsf{R}}$ will be chosen, and HYPNO will be recursively invoked on its premises. HYPNO proceeds in a similar way for the other rules. The ordering of the clauses is such that the application of branching rules is postponed as much as possible. As an example, here is the clause implementing $\Box_{\mathsf{R}}$:

```
1. terminating_proof_search(Hyper,tree(rbox,PrintableHyper,Sub1,Sub2)):-
2.   select(singleSeq([Gamma,Delta],Additional),Hyper,NewHyper),
3.   member(Sigma,Gamma), is_list(Sigma),member(box B,Delta),
4.   list_to_ord_set(Sigma,SigmaOrd), \+member(apdR(SigmaOrd,B),Additional),!,
5.   terminating_proof_search([singleSeq([Sigma,[B]],[])|
      [singleSeq([Gamma,Delta],[apdR(SigmaOrd,B)|Additional])|NewHyper]],Sub1),
6.   terminating_proof_search([singleSeq([[],[B => Sigma]],[])|
      [singleSeq([Gamma,Delta],[apdR(SigmaOrd,B)|Additional])|NewHyper]],Sub2),
7.   extractPrintableSequents(Hyper,PrintableHyper).
```

Line 3 checks whether `Gamma` contains an item `Sigma` which is a list representing a block and if a box formula `box B` belongs to the list `Delta`. Line 4 implements the restriction on the application of the rule used in order to ensure a terminating proof search: if the `Additional` list contains the Prolog term `apdR(SigmaOrd,B)`[3], this means that the rule $\Box_{\mathsf{R}}$ has been already applied on that sequent by using $\Box B$ and the block Σ, and HYPNO does no longer apply it. Otherwise, the predicate `terminating_proof_search` is recursively invoked on the two premises of the rule (lines 5 and 6), by introducing $\Sigma \Rightarrow B$ and $B \Rrightarrow \Sigma$ respectively. Since the rule is invertible, Prolog cut `!` is used in line 4 to eventually block backtracking.

When the predicate `terminating_proof_search` fails, then the initial formula is not valid. On user demand, as recalled at the beginning of this section, HYPNO extracts a model falsifying such a formula from an open saturated branch, following the model extraction method described in [4]. The model is computed by executing the predicate:

```
build_saturated_branch(Hyper, Model).
```

When this predicate succeeds, the variable `Model` matches a description of an open saturated branch obtained by applying the rules of $\mathcal{H}_{\mathsf{E}^*}$ to the initial formula. Since the very objective of this predicate is to build an open saturated hypersequent in the sequent calculus, its clauses are essentially the same as the ones for the predicate `terminating_proof_search`, however rules introducing a branching in a backward proof search are implemented by *pairs* of (disjoint) clauses, each one attempting to build an open saturated hypersequent from the corresponding premise. As an example, the following clauses implement the saturation in presence of a block Σ in the left hand side and of a boxed formula $\Box B$ in the right hand side of a sequent:

[3] The predicate `list_to_ord_set` is used in order to check the applicability of the rule by ignoring the order of the formulas in the block.

```
build_saturated_branch(Hyper,Model):-
  select(singleSeq([Gamma,Delta],Additional),Hyper,NewHyper),
  member(Sigma,Gamma),is_list(Sigma),member(box B,Delta),
  list_to_ord_set(Sigma,SigmaOrd),\+member(apdR(SigmaOrd,B),Additional),
  build_saturated_branch([singleSeq([Sigma,[B]],[])|
   [singleSeq([Gamma,Delta],[apdR(SigmaOrd,B)|Additional])|NewHyper]],Model).
build_saturated_branch(Hyper,Model):-
  select(singleSeq([Gamma,Delta],Additional),Hyper,NewHyper),
  member(Sigma,Gamma),is_list(Sigma),member(box B,Delta),
  list_to_ord_set(Sigma,SigmaOrd),\+member(apdR(SigmaOrd,B),Additional),
  build_saturated_branch([singleSeq([[],[B => Sigma]],[])|
   [singleSeq([Gamma,Delta],[apdR(SigmaOrd,B)|Additional])|NewHyper]],Model).
```

HYPNO will first try to build a countermodel by considering the left premise of $\Box_{\mathsf{R}}$, whence recursively invoking the predicate `build_saturated_branch` on the premise with the sequent $\Sigma \Rightarrow B$. In case of a failure, it will carry on the saturation process by using the right premise of $\Box_{\mathsf{R}}$ with the sequent $B \Rrightarrow \Sigma$.

Clauses implementing axioms for the predicate `terminating_proof_search` are replaced by the last clause, checking whether the current sequent represents an open and saturated hypersequent:

```
build_saturated_branch(Hyper,model(Hyper)):-\+instanceOfAnAxiom(Hyper).
```

Since this is the very last clause of `build_saturated_branch`, it is considered by HYPNO only if no other clause is applicable, then the hypersequent is saturated. The auxiliary predicate `instanceOfAnAxiom` checks whether the hypersequent is open by proving that it is not an instance of the axioms. The second argument matches a term `model` representing the countermodel extracted from `Hyper`.

The implementation of the calculi for extensions of **E** is very similar: given the modularity of the calculi $\mathcal{H}_{\mathsf{E}^*}$, each system is obtained by just adding clauses for both the predicates `terminating_proof_search` and `build_saturated_branch` corresponding to the specific axioms/rules. However, we provide a different Prolog file for each system of the cube. This choice is justified by two reasons: first of all readiness of the code: one may be interested only in one specific system, wishing to have all the rules in a stand-alone file. Second and more important, an implementation of calculi for a family of logic cannot be completely modular: the computation (both proof-search and countermodel extraction) is sensitive to the order of application of the rules, so that the insertion of different rules may result in different orders of application of the whole set of rules.

HYPNO can be used on any computer or mobile device through a web interface implemented in php, which allows the user to choose the modal logic. When a formula is valid, HYPNO builds a pdf file showing a derivation in the corresponding calculus, as well as the LaTeX source file. Otherwise, a countermodel falsifying the initial formula is displayed. Prolog source codes are also available.

4 Performance of HYPNO

We have compared the performance of HYPNO with those of the prover PRONOM [5], which deals with the same set of logics, obtaining promising results. We have tested it by running SWI-Prolog, version 7.6.4, on an Apple MacBook Pro, 2.7 GHz Intel Core i7, 8 GB RAM machine. First, we have tested HYPNO over hundred valid formulas in **E** and considered extensions obtained by generalizing schemas of valid formulas by varying some crucial parameters, like the modal degree (the level of nesting of the $\Box$ connective), already used for testing PRONOM. For instance, we have considered the schemas (valid in all systems):

$$(\Box(\Box(A_1 \wedge (B_1 \vee C_1)) \wedge \cdots \wedge \Box(A_n \wedge (B_n \vee C_n)))) \rightarrow (\Box(\Box((A_1 \wedge B_1) \vee (A_1 \wedge C_1)) \wedge \cdots \wedge \Box((A_n \wedge B_n) \vee (A_n \wedge C_n)))$$

$$(\Box^n C_1 \wedge \cdots \wedge \Box^n C_j \wedge \Box^n A) \rightarrow (\Box^n A \vee \Box^n D_1 \vee \cdots \vee \Box^n D_k)$$

obtaining encouraging results: Table 1 reports the number of timeouts of HYPNO and PRONOM over a set of valid formulas in system **E**.

Table 1. Percentage of timeouts over valid formulas in **E**.

System	0,1 ms	1 ms	100 ms	1 s	5 s
HYPNO	91,50%	78,91%	28,23%	9,52%	5,78%
PRONOM	85,71%	77,55%	57,82%	31,16%	19,80%

HYPNO is able to answer in less than one second on more than the 90% of the tests, whereas PRONOM is not even if we extend the time limit to 5 s.

We have also tested HYPNO on randomly generated formulas, fixing different time limits, numbers of propositional variables, and levels of nesting of connectives. We have compared the performances of HYPNO with those of PRONOM, obtaining the results in Table 2: in each pair, the first number is the percentage of timeouts of HYPNO, the second number is the percentage of timeouts of PRONOM given the fixed time limit.

Table 2. Percentage of timeouts in 5000 random tests (system **E**).

Vars/Depth	1 ms	10 ms	1 s	10 s
3 vars - depth 5	4–5,58%	0,78–1,76%	0,02–0,48%	0–0,22%
3 vars - depth 7	23,78–25,18%	10,86–20,16%	3,16–14,40%	2,02–12%
7 vars - depth 10	45,22–44,94%	34,36–42,36%	19,06–30,30%	16,06–20,34%

Also in case of formulas generated from 3 different atomic variables and with a higher level of nesting (7), HYPNO is able to answer in more than 96% of the cases within 1 s, against the 85% of PRONOM. We have repeated the experiments also for all the extensions of **E** considered by HYPNO: complete results can be found at http://193.51.60.97:8000/HYPNO/#experiments. Moreover, we are planning to perform more accurate tests following the approach of [8], where randomly generated formulas can be obtained by selecting different degrees of probability about their validity.

5 Conclusions

We have presented HYPNO, a prover for the cube of NNMLs based on some hypersequent calculi for these logics recently introduced. HYPNO produces both proofs and countermodels in the bi-neighbourhood semantics. Although no specific optimisation has been implemented, the performances of HYPNO are promising. In the future we intend to extend possible optimisation, in particular to minimize the size of countermodels. Moreover we intend to extend it to other non-normal modal logics in the realm of deontic and agent-ability logics.

Acknowledgements. We thank the reviewers for their careful reading that helped us to improve this paper. We are currently developing a new version of HYPNO taking into account all the suggestions about its implementation.

References

1. Askounis, D., Koutras, C.D., Zikos, Y.: Knowledge means 'all', belief means 'most'. J. Appl. Non-Class. Log. **26**(3), 173–192 (2016). https://doi.org/10.1080/11663081.2016.1214804
2. Beckert, B., Posegga, J.: LeanTAP: lean tableau-based deduction. J. Autom. Reasoning **15**(3), 339–358 (1995). https://doi.org/10.1007/BF00881804
3. Chellas, B.F.: Modal Logic. Cambridge University Press, Cambridge (1980)
4. Dalmonte, T., Lellmann, B., Olivetti, N., Pimentel, E.: Countermodel construction via optimal hypersequent calculi for non-normal modal logics. In: Artemov, S., Nerode, A. (eds.) LFCS 2020. LNCS, vol. 11972, pp. 27–46. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-36755-8_3
5. Dalmonte, T., Negri, S., Olivetti, N., Pozzato, G.L.: PRONOM: proof-search and countermodel generation for non-normal modal logics. In: Alviano, M., Greco, G., Scarcello, F. (eds.) AI*IA 2019. LNCS (LNAI), vol. 11946, pp. 165–179. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-35166-3_12
6. Dalmonte, T., Olivetti, N., Negri, S.: Non-normal modal logics: Bi-neighbourhood semantics and its labelled calculi. In: Proceedings of AiML 12, pp. 159–178. College Publications (2018)
7. Elgesem, D.: The modal logic of agency. Nord. J. Philos. Log. **2**(2), 1–46 (1997)
8. Giunchiglia, E., Tacchella, A., Giunchiglia, F.: SAT-based decision procedures for classical modal logics. J. Autom. Reasoning **28**(2), 143–171 (2002). https://doi.org/10.1023/A:1015071400913
9. Hansen, H.: Tableau games for coalition logic and alternating-time temporal logic-theory and implementation. Master's thesis, University of Amsterdam (2004)
10. Lavendhomme, R., Lucas, T.: Sequent calculi and decision procedures for weak modal systems. Studia Logica **66**, 121–145 (2000). https://doi.org/10.1023/A:1026753129680
11. Lellmann, B.: Combining monotone and normal modal logic in nested sequents – with countermodels. In: Cerrito, S., Popescu, A. (eds.) TABLEAUX 2019. LNCS (LNAI), vol. 11714, pp. 203–220. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29026-9_12
12. Pacuit, E.: Neighborhood Semantics for Modal Logic. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-319-67149-9

13. Pauly, M.: A modal logic for coalitional power in games. J. Log. Comput. **12**(1), 149–166 (2002)
14. Vardi, M.Y.: On epistemic logic and logical omniscience. In: Theoretical Aspects of Reasoning About Knowledge, pp. 293–305. Elsevier (1986)