

Connections: Markov Decision Processes for Classical, Intuitionistic, and Modal Connection Calculi

Fredrik Rømming¹, Jens Otten² and Sean B. Holden¹

¹University of Cambridge, The Old Schools, Trinity Lane, Cambridge CB2 1TN, United Kingdom

²University of Oslo, Problemveien 11, 0313 Oslo, Norway

Abstract

This paper introduces a framework for integrating Reinforcement Learning (RL) with proof search in connection calculi for classical, intuitionistic, and modal logic. We specify a mapping from the relevant connection calculi to Markov Decision Processes (MDPs), and provide a Python library implementing such MDPs.

Keywords

Reinforcement Learning, Automated Reasoning, Connection Calculus, Intuitionistic Logic, Modal Logic

1. Introduction

Automated Theorem Proving (ATP) is concerned with determining whether a given formula is *valid* in a specific logic. Complementary to classical logic, intuitionistic logic is used within interactive proof assistants such as NuPRL [1] and Coq [2], while modal logics [3, 4] have applications in planning, natural language processing and program verification. The complexity of ATP is higher in these non-classical logics than in classical logic, and applications would benefit from improved reasoning tools. One approach to dealing with the non-classical logics is to encode their *Kripke semantics* with *labels* or *prefixes* [5, 6]. Two powerful ATP systems—ileanCoP [7] and MleanCoP [8]—use prefixes and are based on (*clausal*) *connection calculi* for non-classical logics [9, 10, 11].

Combining ATP and *Machine Learning (ML)* can enhance existing ATP systems (or *theorem provers*) [12, 13, 14, 15]. Using ML to guide the proof search clearly has the potential to lead to more efficient ATP systems, while preserving their ability to provide formal proofs. ML can be used in fully automated ATP systems for *premise selection*, *strategy choice* [16] and *inference choice*. Whereas the first two approaches use ML in a pre-processing step, in the third approach ML is tightly integrated into the proof search.

This paper introduces a framework for integrating Reinforcement Learning (RL) with proof search in connection calculi for classical, intuitionistic, and modal first-order logic. There are two


ARECCa 2023: Automated Reasoning with Connection Calculi. International Workshop, TABLEAUX 2023. 18 September 2023. Prague, Czech Republic

✉ fr409@cam.ac.uk (F. Rømming); jeotten@ifi.uio.no (J. Otten); sbh11@cl.cam.ac.uk (S. B. Holden)

🌐 <http://www.cl.cam.ac.uk/~fr409> (F. Rømming); <http://jens-otten.de/> (J. Otten); <http://www.cl.cam.ac.uk/~sbh11> (S. B. Holden)

🆔 0000-0001-7545-4662 (F. Rømming); 0000-0002-4331-8698 (J. Otten); 0000-0001-7979-1148 (S. B. Holden)

© 2023 For the purpose of open access, the author has applied a Creative Commons Attribution (CC BY) licence to any Author Accepted Manuscript version arising from this submission

 CEUR Workshop Proceedings (CEUR-WS.org)

main contributions:

1. The discussion and definition of a mapping from classical and non-classical connection calculi to Markov Decision Processes (MDPs).
2. A Python library implementing such MDPs providing seamless integration with the ML ecosystem facilitating RL experiments for in-prover guidance.

We introduce the connection calculi and ML techniques in Section 2. Section 3 specifies a mapping from proof search to the RL setting. Section 4 describes the implementation of the library. The paper concludes with a summary and a plan for future work.

2. Preliminaries

2.1. Classical, Intuitionistic and Modal Connection Calculi

The following methods are based on *uniform* clausal connection calculi. We provide a short overview of these calculi; more details can be found in [17, 9, 18, 8].

An *atomic formula* (denoted by A) is built up from predicate symbols (denoted by P, Q, R), function symbols and term variables (x, y). A *(first-order) formula* (denoted by F) is built up from atomic formulae, the connectives $\neg, \wedge, \vee, \Rightarrow$, and the first-order quantifiers \forall and \exists . A *modal formula* might also include the modal operators \Box and \Diamond . A *literal* L has the form A or $\neg A$. In the (clausal) connection calculus a formula is represented as a matrix, which is a representation of the formula in a (prefixed) clausal form.

2.1.1. Classical Logic

The *classical matrix* $M(F)$ of a formula F is its representation as a set of clauses, where each clause is a set of literals. It is the representation of F in *disjunctive normal form*. Skolemization of the Eigenvariables is done in the usual way. In the *graphical representation* of a matrix, its clauses are arranged horizontally, while the literals of each clause are arranged vertically. In contrast to sequent and tableau calculi, the connection calculus uses a *connection-driven* search to find a proof for the validity of a matrix $M(F)$ for a given formula F .

A *connection* is a set $\{A_1, \neg A_2\}$ of literals with the same predicate symbol but different polarities. A *term substitution* σ_T assigns terms to variables. A connection is *σ_T -complementary* iff $\sigma_T(A_1) = \sigma_T(A_2)$. The axiom and the three rules of the (clausal) *connection calculus* are given in Figure 1 ($:p_1$ and $:p_2$ are to be ignored for classical logic) [17]. The words of the calculus are tuples “ $C, M, Path$ ”, where M is a matrix, C is a (subgoal) clause or ε and (the active) $Path$ is a set of literals or ε . A *copy of a clause* C is made by renaming all variables in C . The *rigid* term substitution $\sigma = \sigma_T$ is calculated by using one of the well-known *term unification* algorithms whenever a connection is identified. A *connection proof* of M is a proof of $\varepsilon, M, \varepsilon$ with a substitution $\sigma = \sigma_T$.

2.1.2. Non-Classical Logics

For *intuitionistic* and *modal logic*, the matrix and the calculus are extended by prefixes, representing world paths in the Kripke semantics; see [19, 5, 6]. A *prefix* p is a string consisting

<i>Axiom (A)</i>	$\frac{}{\{\}, M, Path}$	<i>Start (S)</i>	$\frac{C_2, M, \{\}}{\varepsilon, M, \varepsilon}$ and C_2 is copy of $C_1 \in M$
<i>Reduction (R)</i>	$\frac{C, M, Path \cup \{L_2 : p_2\}}{C \cup \{L_1 : p_1\}, M, Path \cup \{L_2 : p_2\}}$		and $\{L_1 : p_1, L_2 : p_2\}$ is σ -complementary
<i>Extension (E)</i>	$\frac{C_2 \setminus \{L_2 : p_2\}, M, Path \cup \{L_1 : p_1\}}{C \cup \{L_1 : p_1\}, M, Path}$	$\frac{C, M, Path}{\{L_1 : p_1, L_2 : p_2\}}$	and C_2 is a copy of $C_1 \in M$, $L_2 : p_2 \in C_2$, $\{L_1 : p_1, L_2 : p_2\}$ is σ -complementary

Figure 1: The (clausal) connection calculus for classical, intuitionistic and modal logic

of variables (denoted by U, V, W) and constants (denoted by a, b) and assigned to each literal. Skolemization is not only used for the (first-order) Eigenvariables, but extended to prefix constants [9]. Using the occurs-check during unification this ensures that the reduction ordering is acyclic [6]. The *intuitionistic* and *modal matrix* $M(F)$ of a formula F is a representation of F in standard clausal form, in which each literal L is marked with its prefix $:p$ [9, 10, 11].

A *prefix substitution* σ_P assigns strings to prefix variables and is calculated by a *prefix unification* that depends on the specific non-classical logic [9, 10, 8]. In intuitionistic and modal logic, a connection $\{L_1 : p_1, L_2 : p_2\}$ is σ -*complementary* iff both its literals *and* prefixes can be unified under a combined substitution $\sigma = (\sigma_T, \sigma_P)$; that is, additionally $\sigma_P(p_1) = \sigma_P(p_2)$ must hold. An *intuitionistic/modal connection proof* of M is a proof of $\varepsilon, M, \varepsilon$ in the calculus in Figure 1 substitution $\sigma = (\sigma_T, \sigma_P)$ [19, 6].

Example 1. Consider the formula $F_1 := ((P \vee \forall x \neg(Qx \Rightarrow Qc)) \wedge R) \Rightarrow (P \wedge R)$. It has the following intuitionistic (prefixed) matrix $M := M(F_1)$

$$\{\{P^0 : Ua_2^*, R^0 : Ub_2^*\}, \{P^1 : UV_1, Q^0 x : UVW a_1^* b_1^*\}, \{P^1 : UV_2, Q^1 c : UVW a_1^* V_3\}, \{R^1 : UV_4\}\}$$

where $a_1^* := a_1(U, V, x, W)$, $b_1^* := b_1(U, V, x, W)$, $a_2^* := a_2(U)$, $b_2^* := b_2(U)$.¹ M has the following graphical representation and (graphical) connection proof with the substitutions $\sigma_T(x) = c$ and $\sigma_P(V_1) = a_2^*$, $\sigma_P(V_2) = a_2^*$, $\sigma_P(V_3) = b_1^*$, $\sigma_P(V_4) = b_2^*$; literals of each connection are connected with a line.

$$\left[\begin{array}{c} \left[\begin{array}{c} P^0 : Ua_2^* \\ R^0 : Ub_2^* \end{array} \right] \left[\begin{array}{c} P^1 : UV_1 \\ Q^0 x : UVW a_1^* b_1^* \end{array} \right] \left[\begin{array}{c} P^1 : UV_2 \\ Q^1 c : UVW a_1^* V_3 \end{array} \right] \left[R^1 : UV_4 \right] \end{array} \right]$$

The *formal* connection proof (where prefixes are omitted) is shown in Figure 2.

2.2. MDPs and Reinforcement Learning

We now provide an introduction to MDPs and RL—details can be found in [21].

Most proof procedures are search algorithms: there is an initial state, states can be modified by actions, and the goal is to find a proof state. The use of *heuristics* is crucial for performance. For

¹*Polarities* ($\in \{0, 1\}$) are used to mark negated and non-negated literals (see [20, 6]).

3. Connection Calculi as Markov Decision Processes

As described in the previous section, RL concerns agents interacting with environments. In the case of proof procedures, one can consider an agent deciding which choice to make at each point in the proof search. Hence, to apply RL we need to define the proof search environment and its choice points. We now address the description of proof search procedures in connection calculi using MDPs.

While the reader familiar with the connection calculus might be tempted to see the relationship between proof and MDP as straightforward, it is in fact more subtle than is apparent at first glance, and requires some care to define correctly. While the connection calculi define procedures whereby sequential decisions are made to find proofs, they do not directly define MDPs. For confluent proof calculi such as resolution, one can treat the words of the calculus as observations and inference rules as actions, since all information about the state of the proof is carried in the most recently generated word. However, this cannot be done with connection calculi, because it is unclear how to handle branching and backtracking. The words of the connection calculi do not carry enough information to know whether the current state is a goal state (a proof), or to know what inferences have or have not been attempted. Allowing dead-end states that are not proofs would be too restrictive, so backtracking is needed.

We now specify the *state space*, *action space*, *transition space*, and *reward function* tuple $(S, A, \mathcal{S}, \mathcal{R})$ for *CC-MDP*, an RL environment for proof search in connection calculi.

Definition 1 (CC-MDP State Space). *The state space S is defined as the set of all possible derivations (a derivation is an incomplete proof in which some leaves are not closed by axioms) in the connection calculus together with the substitution $\sigma = \sigma_T$. Further, to keep track of the open backtracking choices, each node in the connection derivation is marked with the possible inference steps that have not been attempted so far.*

For efficiency reasons, the rigid prefix substitution σ_P (for the *non-classical* logics) is only calculated after a classical proof has been found. It is therefore *not* part of the states in S (see also the description of the implementation in Section 4).

Example 2. Consider the formula $((P \vee \forall x \neg(Qx \Rightarrow Qc)) \wedge R) \Rightarrow (P \wedge R)$ from Example 1 and its classical matrix $M = \{\{P, R\}, \{\neg P, Qx\}, \{\neg P, \neg Qc\}, \{\neg R\}\}$. Figure 3 shows a (possible) state $s \in S$ in the proof search of M . s includes a derivation of M together with the substitution σ_T , and for each node a list of non-attempted inference steps.

$$\begin{array}{c}
 I_3 \frac{\frac{\frac{\{\neg P\}, M, \{P, Qx'\}}{I_2} \quad \{\}, M, \{P\}}{I_1} E \quad \{R\}, M, \{\}}{E} \\
 \frac{\{Qx'\}, M, \{P\}}{I_2} \\
 \frac{\{P, R\}, M, \{\}}{I_1} S \\
 \varepsilon, \{\{P, R\}, \{\neg P, Qx\}, \{\neg P, \neg Qc\}, \{\neg R\}\}, \varepsilon
 \end{array}
 \quad
 \begin{array}{l}
 \sigma_T = \{x' \setminus c\} \\
 \text{Non-attempted:} \\
 I_3 : \{\} \\
 I_2 : \{(E, \{\neg P, \neg Qc\})\} \\
 I_1 : \{\}
 \end{array}$$

Figure 3: A CC-MDP state for the matrix M

Definition 2 (CC-MDP Action Space). *The action space A consists of rule application actions $A_{inferences}$ and a backtracking action $a_{backtrack}$. There is a rule application action for each rule in the connection calculus. Hence, a rule application action $a_{r,x} \in A_{inferences}$ is specified by the rule name $r \in \{S, R, E\}$ (for Start, Reduction or Extension) and the associated clause and/or literal x . Specifically, an action $a_{r,x}$ can have one of the following forms: a_{S,C_2} for the Start rule, a_{R,L_2} for the Reduction rule and $a_{E,C_2/L_2}$ for the Extension rule.*

Example 3. To get from the initial state $\varepsilon, M, \varepsilon$ to the state in Figure 3 one can take the actions: $a_{S,\{P,R\}}$, $a_{E,\{-P,Qx'\}/\neg P}$, $a_{E,\{\neg Qc,\neg P\}/\neg Qc}$. These actions are a *start* step with the clause $\{P, R\}$, followed by two extension steps connecting the leftmost literal² in the leftmost open subgoals of the proof tree to the literal P in the clause $\{-P, Qx'\}$ and the literal $\neg Qc$ in the clause $\{\neg Qc, \neg P\}$.

We say that action a is *valid* in state s if action $a = a_{backtrack}$ or a is a rule application action $a_{r,x} \in A_{inferences}$ denoting a valid rule application to the leftmost literal in the leftmost open subgoal in the proof tree of s . A valid rule application takes the rigid term substitution σ_T into account, so $\sigma_T(L_1) = \sigma_T(L_2)$. As for the states in the state space S , the rigid prefix substitution σ_P is not taken into account (and updated) for the *non-classical* logics. For the application of the *reduction* and the *extension* rule, we only require that the prefixes p_1 and p_2 of the connection can be unified under a (new) local prefix substitution σ'_P (using a so-called *weak prefix unification*). To handle proper backtracking, when a rule application action a is taken from state s to s' , a is no longer counted as a non-attempted inference for the node in the tableau of s' corresponding to the principle node of a . The special action $a_{backtrack}$ backtracks the state's derivation from the leftmost literal in the leftmost subgoal to the previous choice point, which still has non-attempted inferences.

In general an MDP models state transitions stochastically—performing action a in state s leads to a new state $s' \sim \mathcal{S}(S'|s, a)$. In a connection prover the transition is deterministic, in the sense that performing action a in state s leads reliably to a single outcome state s' . This gives rise to the following deterministic state transition function.

Definition 3 (CC-MDP Transition Function). *The transition distribution \mathcal{S} is defined as*

$$\mathcal{S}(S' = s'|s, a) = \begin{cases} 1 & \text{if } a \text{ is valid in } s \text{ and } s' \text{ is the (deterministic) result} \\ & \text{of applying } a \text{ to } s \\ 0 & \text{otherwise.} \end{cases}$$

Notice that the transition function is necessarily deterministic, as any probabilistic transition function would not accurately describe the dynamics of the underlying system.

We only consider the first literal of the leftmost open subgoal for rule application. This is because all subgoals need to be closed, so we do not consider alternative subgoals and literals within subgoals as choice points for the MDP.

Definition 4 (CC-MDP Reward Function). *To remain faithful to the underlying problem, we consider the following relatively sparse reward function*

$$\mathcal{R}(s'; s, a) = \begin{cases} 1 & \text{if } s' \text{ is a proof} \\ 0 & \text{otherwise} \end{cases}$$

²All clauses (including subgoal clauses) are treated as ordered sets of literals.

where s' is a proof iff the derivation of s' is a proof under the unique (combined) substitution $\sigma = \sigma_T$ (for classical logic) or $\sigma = (\sigma_T, \sigma_P)$ of s' .

This reward function is the simplest function accurately describing the goal while preserving optimality of solutions.

Example 4. Figure 4 shows the graph representation of a part of CC-MDP.

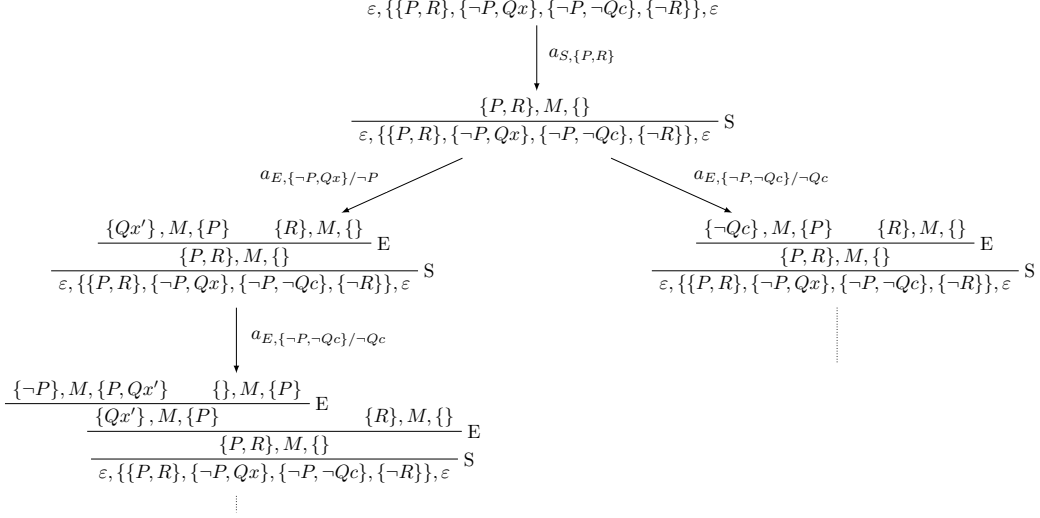


Figure 4: Part of the CC-MDP graph (omitting σ_S and non-attempted inferences)

Proof search in the classical and non-classical connection calculi can be framed as an agent interacting with the CC-MDP environment, giving the necessary theoretical framework for applying RL to the proof search in these connection calculi.

4. Implementation

4.1. Connection Calculi as MDPs in Python

Connections [22] is a Python library of connection calculi implemented as MDPs, providing environments for proof search in connection calculi. It provides OpenAI Gym/Gymnasium-like [23] environments for proof search in connection calculi for classical, intuitionistic, and modal first-order logic. It currently supports the modal logics D, T, S4, and S5, each for the constant, cumulative, and varying domains.

Connections implements the basic calculi for classical, intuitionistic, and modal logic as described above, enhanced by *regularity* [24]. The observation and action spaces are as described in Section 3, treating literals and (first-order) terms as objects associated with locations in a matrix (represented as a list of lists of literals) and as a tableau-like proof tree. For intuitionistic and modal logic, literals and terms have an extra field for their prefixes represented as (first-order) terms. Connections is implemented natively in Python with no dependencies. As Python is the de-facto language for ML, the library provides an accessible and reproducible way to conduct RL experiments with provers based on connection calculi. Using standardized frameworks

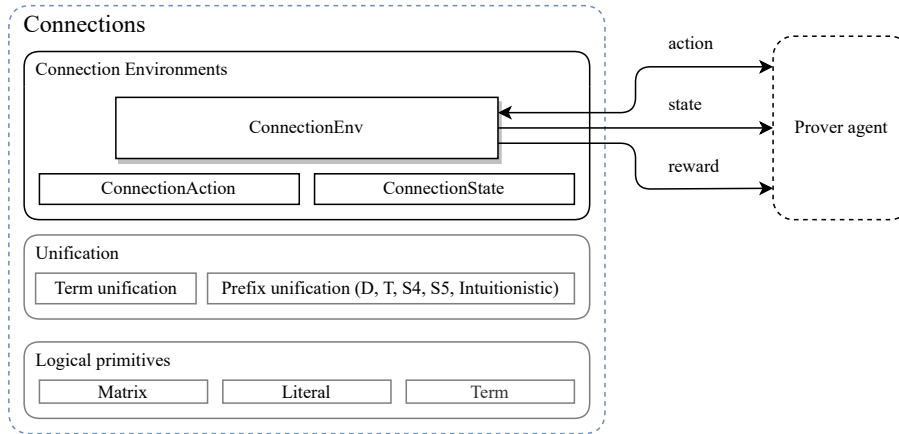


Figure 5: Overview of the Connections Python library and its main modules

increases confidence in the correctness of the implementation, and the environments can easily be incorporated into the rest of the ML ecosystem alongside frameworks such as RLlib [25], Stable Baselines [26], PyTorch [27], Tensorflow [28] and others. Figure 5 gives an overview of Connections and how it fits into the RL setting.

Compared to conducting learning experiments with external calls to Prolog and OCaml implementations of leanCoP, using the Connections environments drastically reduces the complexity needed to control the prover. This is due to eliminating the need for remote procedure calls, and to the imperative basis of the environments, which allow fine-grained control while respecting abstraction levels. That is, since Connections is imperative (logic + control, explicit state space) while Prolog implementations are declarative (logic + no control, implicit state space), modifications to control such as the restriction of backtracking can easily be made in ways that are much trickier if you try to use the Prolog cut. (See also [29].)

The non-classical Connections environments inherit from the classical environment, adding logic-specific prefixes and prefix unification algorithms. The non-classical provers based on Connections perform a classical proof search, in which the prefixes of the literals in each connection are collected. After a classical proof is found, these prefixes are unified by a prefix unification algorithm to ensure that the classical proof is also a valid non-classical proof.

Besides the basic calculi, the Connections environments implement two additional optimizations, significantly reducing the underlying search space while preserving completeness. The start clause C_1/C_2 of the start rule is restricted to *positive start clauses* (those with polarity 0) and the *regularity condition* [24] is employed. The translation into a (prefixed) matrix is done in a pre-processing step. If the problem contains explicit axiom and conjecture formulas, the standard/naive translation into clausal form is performed for the axiom formulas, while a *definitional translation* [18] is performed for the conjecture formula.

The Connections environments are not provers by themselves, they expose an interface that agents can use to train and make inferences, completing the RL agent-environment interaction loop, as shown in Figure 5. A prover in this context is an agent making consecutive steps in a Connections environment until it has found proof or timed out.


```

env = ConnectionEnv("problem_path")
observation, info = env.reset()
while True:
    action = env.action_space[0] # Always choose first available action
    observation, reward, terminated, truncated, info = env.step(action)
    if terminated or truncated:
        break

```

Figure 6: Python code for the pyCoP prover based on Connections

4.2. Python Connection Provers for Classic and Non-classical Logics

The Connections environments can be used to build both *learning* and *non-learning* connection provers, depending on the agent used. For example, by using non-learning agents that always choose the first available action, we obtain standard Python connection provers in an elegant and straightforward way—the provers emerge from the interaction between the “always-first” agent and a Connections environment. The complete Python code implementing such a prover is shown in Figure 6. Depending on the environment used (`ConnectionEnv`, `IConnectionEnv` or `MConnectionEnv`) this results in three (stand-alone) Python provers [22] for classical, intuitionistic and modal logics, called `pyCoP`, `ipyCoP` and `mpyCoP` respectively. These are based on the same connection calculi as the `leanCoP` family of theorem provers implemented in Prolog [17, 18, 9, 7, 10, 8]. By design, the `pyCoP` provers mimic the classical proof steps of `leanCoP` 1.0, using the positive start clause technique and iterative deepening on the size of the active path. However, the `pyCoP` provers do not reorder clauses during proof search, and integrate an enhanced regularity check [24]. This corresponds to version 1.0f of `leanCoP` [22].

While the main purpose of the Connections environments is to facilitate easy implementation of learned provers for classical, intuitionistic, and modal logic using the MDP + agent view of connection proof search, the `pyCoP` provers highlight the general (learning and non-learning) capabilities of the Connections environments and give confidence in the correctness of their implementation by showing that they can be used to emulate `leanCoP`, `ileanCoP`, and `MleanCoP`.

5. Conclusion

We present a Python library providing a framework for ML in connection calculi for classical and non-classical logics, and with specific emphasis on facilitating experiments using RL to guide proof search. Aside from its ML-centric component, this also represents the first non-Prolog implementation of provers based on the clausal non-classical connection calculi, and using prefix unification to capture the Kripke semantics of intuitionistic and modal first-order logics.

We are at present using this library to experiment with RL methods in an attempt to improve the performance of the unmodified provers, and we hope that the library inspires and facilitates others to explore their own ideas within this space.

In future work we intend to extend the library to allow us more fully to address restricted backtracking, refutation techniques and iterative deepening, and to include both further modal logics, and non-clausal methods such as those of `nanoCoP` [30, 31].

References

- [1] R. L. Constable, et al., *Implementing Mathematics with the NuPRL proof development system*, Prentice–Hall, Englewood Cliffs, NJ, 1986.
- [2] Y. Bertot, P. Castéran, *Interactive Theorem Proving and Program Development Coq’Art: The Calculus of Inductive Constructions*, EATCS Series, Springer, Heidelberg, 2004.
- [3] P. Blackburn, J. van Benthem, F. Wolter, *Handbook of Modal Logic*, Elsevier, Amsterdam, 2006.
- [4] M. Fitting, R. L. Mendelsohn, *First-Order Modal Logic*, Kluwer, Dordrecht, 1998.
- [5] A. Waaler, *Connections in nonclassical logics*, in: A. Robinson, A. Voronkov (Eds.), *Handbook of Automated Reasoning*, Elsevier Science, Amsterdam, 2001, pp. 1487–1578.
- [6] L. A. Wallen, *Automated Deduction in Non-Classical Logics*, MIT Press, Cambridge, 1990.
- [7] J. Otten, *leanCoP 2.0 and ileanCoP 1.2: High performance lean theorem proving in classical and intuitionistic logic*, in: A. Armando, P. Baumgartner, G. Dowek (Eds.), *IJCAR 2008*, volume 5195 of *LNAI*, Springer, Heidelberg, 2008, pp. 283–291.
- [8] J. Otten, *MleanCoP: A connection prover for first-order modal logic*, in: S. Demri, D. Kapur, C. Weidenbach (Eds.), *IJCAR 2014*, volume 8562 of *LNAI*, Springer, Heidelberg, 2014, pp. 269–276.
- [9] J. Otten, *Clausal connection-based theorem proving in intuitionistic first-order logic*, in: *TABLEAUX 2005*, volume 3702 of *LNAI*, Springer, Heidelberg, 2005, pp. 245–261.
- [10] J. Otten, *Implementing connection calculi for first-order modal logics*, in: E. Ternovska, K. Korovin, S. Schulz (Eds.), *9th International Workshop on the Implementation of Logics (IWIL 2012)*, volume 22 of *EPIC*, EasyChair, 2012, pp. 18–32.
- [11] J. Otten, W. Bibel, *Advances in connection-based automated theorem proving*, in: M. Hinchey, J. P. Bowen, E.-R. Olderog (Eds.), *Provably Correct Systems, NASA Monographs in Systems and Software Engineering*, Springer, Cham, 2017, pp. 211–241.
- [12] J. Urban, J. Vyskočil, P. Štěpánek, *MaLeCoP Machine Learning Connection Prover*, in: K. Brünnler, G. Metcalfe (Eds.), *Automated Reasoning with Analytic Tableaux and Related Methods, Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, 2011, pp. 263–277.
- [13] G. Irving, C. Szegedy, A. A. Alemi, N. Eén, F. Chollet, J. Urban, *DeepMath - Deep Sequence Models for Premise Selection*, in: *Advances in Neural Information Processing Systems*, volume 29, Curran Associates, Inc., 2016.
- [14] C. Kaliszyk, J. Urban, H. Michalewski, M. Olšák, *Reinforcement Learning of Theorem Proving*, in: *Advances in Neural Information Processing Systems*, volume 31, Curran Associates, Inc., 2018.
- [15] Z. Zombori, J. Urban, C. E. Brown, *Prolog Technology Reinforcement Learning Prover*, in: N. Peltier, V. Sofronie-Stokkermans (Eds.), *Automated Reasoning, Lecture Notes in Computer Science*, Springer International Publishing, Cham, 2020, pp. 489–507.
- [16] C. Mangla, S. B. Holden, L. Paulson, *Bayesian ranking for strategy scheduling in automated theorem provers*, in: J. Blanchette, L. Kovács, D. Pattinson (Eds.), *Proceedings of the 11th International Joint Conference on Automated Reasoning (IJCAR)*, volume 13385 of *Lecture Notes in Artificial Intelligence*, Springer, 2022, pp. 559–577. 19 pages.
- [17] J. Otten, W. Bibel, *leanCoP: lean connection-based theorem proving*, *Journal of Symbolic*

Computation 36 (2003) 139–161.

- [18] J. Otten, Restricting backtracking in connection calculi, *AI Commun.* 23 (2010) 159–182.
- [19] J. Otten, Non-clausal connection calculi for non-classical logics, in: R. Schmidt, C. Nalon (Eds.), *TABLEAUX 2017*, volume 10501 of *LNAI*, Springer, Cham, 2017, pp. 209–227.
- [20] R. M. Smullyan, *First-Order Logic*, *Ergebnisse der Mathematik und ihrer Grenzgebiete*, Springer-Verlag, Berlin, Heidelberg, New York, 1968.
- [21] R. S. Sutton, A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd edition ed., MIT Press, 2018.
- [22] F. Rømming, *Connections*, 2023. URL: <https://github.com/fredrrom/connections>.
- [23] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, OpenAI Gym, 2016. ArXiv:1606.01540 [cs].
- [24] R. Letz, G. Stenz, Model elimination and connection tableau procedures, in: *Handbook of Automated Reasoning*, Elsevier Science Publishers, Amsterdam, 2001, pp. 2015–2112.
- [25] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, I. Stoica, RLLib: Abstractions for Distributed Reinforcement Learning, in: *Proceedings of the 35th International Conference on Machine Learning*, PMLR, 2018, pp. 3053–3062. ISSN: 2640-3498.
- [26] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, N. Dormann, Stable-baselines3: Reliable reinforcement learning implementations, *Journal of Machine Learning Research* 22 (2021) 1–8.
- [27] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., Pytorch: An imperative style, high-performance deep learning library, *Advances in neural information processing systems* 32 (2019).
- [28] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015. Software available from tensorflow.org.
- [29] S. B. Holden, Connect++: a fast, flexible and modifiable connection prover to support machine learning, in: *Proceedings of the Workshop on Automated Reasoning with Connection Calculi (AReCCa)*, 2023.
- [30] J. Otten, A non-clausal connection calculus, in: K. Brunnler, G. Metcalfe (Eds.), *TABLEAUX 2011*, volume 6793 of *LNAI*, Springer, Heidelberg, 2011, pp. 226–241.
- [31] J. Otten, nanoCoP: A non-clausal connection prover, in: N. Olivetti, A. Tiwari (Eds.), *IJCAR 2016*, volume 9706 of *LNAI*, Springer, Heidelberg, 2016, pp. 302–312.