Learning Guided Automated Reasoning: A Brief Survey

Lasse Blaauwbroek[1,2], David Cerna[3], Thibault Gauthier[1], Jan Jakubův[1,4], Cezary Kaliszyk[4], Martin Suda[1], and Josef Urban[1]

[1] Czech Technical University in Prague
[2] Radboud University Nijmegen
[3] Czech Academy of Sciences Institute for Computer Science
[4] University of Innsbruck

Abstract. Automated theorem provers and formal proof assistants are general reasoning systems that are in theory capable of proving arbitrarily hard theorems, thus solving arbitrary problems reducible to mathematics and logical reasoning. In practice, such systems however face large combinatorial explosion, and therefore include many heuristics and choice points that considerably influence their performance. This is an opportunity for trained machine learning predictors, which can guide the work of such reasoning systems. Conversely, deductive search supported by the notion of logically valid proof allows one to train machine learning systems on large reasoning corpora. Such bodies of proof are usually correct by construction and when combined with more and more precise trained guidance they can be boostrapped into very large corpora, with increasingly long reasoning chains and possibly novel proof ideas.
In this paper we provide an overview of several automated reasoning and theorem proving domains and the learning and AI methods that have been so far developed for them. These include premise selection, proof guidance in several settings, AI systems and feedback loops iterating between reasoning and learning, and symbolic classification problems.

1 Introduction

No one shall drive us from the semantic AI paradise of computer understandable math and science!

– AGI'18 [154]

Automated Reasoning (AR) [126] and Automated Theorem Proving (ATP) systems are general AI systems that are in principle capable of solving arbitrary mathematical and reasoning problems. Their theoretical *completeness* means that *any* solvable problem, regardless of its difficulty, will be eventually solved.

In practice, today's AR and ATP systems however soon encounter combinatorial explosion, typically preventing them from solving hard open problems in a reasonable time. Regardless of which proof calculi they implement, they have to make many decisions concerning which theories and lemmas to use, which inference steps or tactics to choose, which instantiations to apply, how to divide

the problems, split the clauses, propose useful lemmas, etc. These choices do not typically influence the theoretical correctness and completeness of the proof mechanisms but rather the systems' efficiency and practical performance.

AR/ATP practitioners have often *manually designed and optimized* a spectrum of heuristics, efficiency and calculus improvements based on their practical and theoretical insights. Such manual designs can sometimes be very successful, as witnessed, e.g., by the large performance boosts brought by using the right term orderings [61], literal selection function [53] and age-weight ratios [98] in ATPs, sophisticated indexing methods for ATPs [121] and SAT solvers [102], as well as by human-designed improvements of the underlying calculi such as CDCL [143,69] in SAT solving, ordering-based constraints [105], and AVATAR-style splitting [159,124] in saturation-based ATP.

On the other hand, general mathematics is undecidable and arbitrarily complicated, and it seems increasingly hard to manually design more complex heuristics for more complex domains and problems. At the same time, *automated design and optimization* of parameters, heuristics, functions and algorithms has been a major topic in AI since its beginnings. Especially in the last decades the field of machine learning (ML) has produced a number of interesting *data-driven* [142] methods that can be used in AR and ATP. Perhaps the most interesting area of such research is how to combine and interleave the AR and ML methods, creating feedback loops and meta-systems that can continuously improve their skills and keep finding—for long time—harder and harder proofs and explanations.

In this survey, we cover the development of such methods starting from the early AI/TP systems and tasks such as high-level knowledge selection, to the today's spectrum of architectures and tasks that include guidance at various levels, a variety of learning-based methods with various speed/accuracy tradeoffs, and a variety of combinations of the ML and AR methods. Automated Reasoning is however a large field and this brief survey does not make claims to be exhaustive. We mainly focus here on the fields of Automated and Interactive Theorem Proving. For some related AR fields similar surveys and overviews have been written recently. In particular, we recommend the recent exhaustive overview [57] discussing ML methods in the context of SAT and QSAT solving.

2 Early History

According to Davis [26], in the beginning of AR and ATP, two research directions emerged: (i) *heuristic*/AI emulation of human thought processes, such as Newell's and Simon's Logic Theorist [104] and later Bledsoe's work [14], and (ii) design of crisp algorithms based on logic transformations and calculi, such as the early proof procedures by Davis and Putnam [27], Gilmore [46] and Robinson [125]. The latter (*logicist*) approach largely prevailed in ATP in the first decades, thanks to major advances such as resolution, DPLL and paramodulation. Interesting examples of the heuristic/AI systems at that time include Lenat's Automated Mathematician (AM) [93], Langley's Bacon [92] and Colton's HR [22], which propose concepts and conjectures, thus also qualifying as symbolic ML.

First interesting combinations of state-of-the-art ATPs with ML methods (both statistical and symbolic) were designed in the 90s by the Munich AR group [31,30]. The methods and systems included, for example, the invention of *tree (recursive) neural networks (TNNs)* for classifying logical expressions by Goller and Kuchler [50], and the development of the E prover [137] by Schulz, which allowed proof guidance by symbolic patterns abstracted from related proof searches [135]. Related to that was the *hints* proof guidance in *Otter* developed for attacking open conjectures by Veroff [158], again based on symbolic abstraction (subsumption) of lemmas in related proofs.

At the same time, large ITP (*interactive theorem proving*) libraries started to appear, with the *Mizar* project [6] producing over 700 formal mathematical articles by 2001. Such large libraries have since become a natural target for combining ML and AR. Urban's 1998 MSc thesis [148] was perhaps the first attempt to learn symbolic heuristics over such formal libraries with the use of *inductive logic programming* (ILP – a symbolic ML approach). Already before that, the ILF project [25] started to work on translations between Mizar and ATPs. This was continued by the MPTP project [149,150] which in 2003 released a dataset of about 30000 Mizar ATP problems, and reported about the first large ATP and ML experiments over it. This included training a *naive Bayes* (Section 4.2) predictor over the Mizar library (Mizar Proof Advisor) to select suitable library facts (premises) for proving the next Mizar problems. In 2004, the Isabelle/Sledgehammer was developed [99], including the first heuristic (non-learning) premise selection methods (MePo) [100]. Such *hammer* systems connecting ITPs with ATPs have become an AR topic of its own, see [12].

Since efficient premise selection is a critical task in hammers, a lot of the initial AI/TP work focused on it. The 2007 MPTP Challenge[5] followed by the 2008 CASC LTB (large theory batch) competition introduced benchmarks of large-theory problems and suitable settings for AI/TP system training and evaluation. This quickly led to fast non-learning methods such as SInE [54] integrated in ATPs, as well as larger meta-systems such as MaLARea [152,156] that interleave proving with learning of premise selection.

The early non-learning systems typically focus on symbols and symbolic features in the formulas, with MePo and SInE using metrics such as symbolic overlap (Jaccard index) and symbol rarity, starting with the conjecture symbols and recursively adding the most related formulas until a certain size of the set of premises is reached. The Mizar Proof Advisor and *MaLARea* instead initially trained naive Bayes to associate the conjecture symbols (features) with the premise names, later adding more complicated syntactic features such as term walks and also semantic model-based features (Section 3.3), obtained by evaluating both conjectures and premises in a growing set of finite models.

These first approaches turned out quite successful, with MePo, SInE and MaLARea considerably increasing the performance of the underlying ATPs in large theories. This has (re-)opened several research topics, such as (i) how to suitably characterize mathematical formulas and objects, (ii) what are the suit-

[5] https://www.tptp.org/MPTPChallenge/

able ML methods, (iii) on what level should the ML guidance be applied, and (iv) how to construct larger feedback loops and meta-systems combining ML and AR. In the next sections we discuss some of these topics.

.

3 Characterization of Mathematical Knowledge

How knowledge is represented is often essential to understanding and gleaning deeper insights about the subject encompassing it. To take a prominent example, consider Fermat's Last Theorem. The theorem statement concerns what most would classify as number theory, yet the proof lives in the world of *elliptic curves*. There may be an elementary number-theoretic proof of the theorem. However, given the state of mathematics at the time, the shortest path was through a, at least to the mathematics dilettante, seemingly unrelated area. When developing methods for classifying mathematical expressions, analogously, proper representation is essential to extracting the necessary semantic notions.

3.1 Syntactic Features

As mentioned above, the work on premise selection started with syntactic characterizations of conjectures and premises. The early methods extend the extraction of symbols (already mentioned in the discussion of SInE and MePo in the previous section) by using subterms as additional features, applying various normalizations to them to increase the feature matching between the formulas, and using syntactic representation of types and their connections. The aim of such early investigations was to provide HOL Light with automation (a hammer) for proof development in the Flyspeck project. For example, [72] included features based on the subterms normalized by replacing variables by their types, their de Bruijn numbers, or merging all variables. As an example, the HOL theorem `DISCRETE_IMP_CLOSED` with the HOL Light statement:

```
∀s:real^N→bool e.
        &0 < e ∧ (∀x y. x IN s ∧ y IN s ∧ norm(y − x) < e ⟹ y = x)
        ⟹ closed s
```

is characterized by the following set of strings that represent unique features.

```
"real", "num", "fun", "cart", "bool", "vector_sub", "vector_norm",
"real_of_num", "real_lt", "closed", "_0", "NUMERAL", "IN", "=", "&0",
"&0 < Areal", "0", "Areal", "Areal^A", "Areal^A - Areal^A",
"Areal^A IN Areal^A->bool", " Areal^A->bool", "_0", "closed Areal^A->bool",
"norm (Areal^A - Areal^A)", "norm (Areal^A - Areal^A) < Areal"
```

The above approach resulted in many unique features, for which various feature weighting schemes were explored for efficient use with the premise selection predictors. The most efficient schemes were based on the linguistic TF-IDF [71], raising considerably the performance of the best k-nearest neighbor predictors.

In TF-IDF, a term t, present in a collection of documents D, is weighted by the logarithm of the inverse of the term's frequency within D, that is:

$$\mathrm{IDF}(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

A further improvement to the syntactic characterization of terms was introduced in the work on machine learning for Sledgehammer [86], where walks through term graphs were considered. Adding such features again allows the abstraction of the global term structure and better sharing of such automatically created concepts/features between the statements.

3.2 ENIGMA Syntactic Features

Syntactic features are also heavily used by ENIGMA systems, where clauses are represented by finite numeric *feature vectors*. ENIGMA (***E**fficient Lear**n**ing-Based **I**nference **G**uiding **Ma**chine*) is a state-of-the-art machine learning guidance system for the ATP E [138]. In the first ENIGMA [62], the feature vector is constructed by traversing the clause syntax tree and collecting all top-down oriented symbol paths of length 3. For example, given the unit clause $\mathit{plus}(X, \mathit{nul}) = X$, we obtain triples $(\oplus, =, \mathit{plus})$, $(\oplus, =, X)$, $(=, \mathit{plus}, X)$, and $(=, \mathit{plus}, \mathit{nul})$, where $\oplus$ signifies the root node of the syntax tree of positive literals. Additionally, to abstract from variable names and to deal with possible collisions of Skolem symbols, all variables are replaced by a special name $\odot$ and all Skolem symbols by $\circledast$. After this renaming, the triples contain only the symbols from the problem signature Σ and 4 special symbols $\{\oplus, \ominus, \odot, \circledast\}$, where $\ominus$ is used as the root node of negative literals. This allows exhaustive enumeration of all possible triples, assigning each triple a unique number smaller than $(|\Sigma| + 4)^3$. This number is used as an index in the feature vector, and the vector value specifies the number of occurrences of the corresponding triple in the clause.

While the first version of ENIGMA yielded encouraging results, it was not yet ready to scale to benchmarks with larger signatures. This led to the second ENIGMA [63] with enhanced feature vectors. Instead of an exhaustive enumeration of all possible symbol triples, only the triples appearing in the training data were enumerated. This significantly reduced the vector length as many triples do not appear in the provided training data, and many can not appear at all, for example, $(=, =, =)$. This enumeration must be stored together with the trained model. The second ENIGMA additionally introduced the following additional clause features.

Count Features extend the feature vector with the clause length, and the counts of positive/negative literals. Moreover, for each symbol f we added the number of occurrences of f in positive/negative literals, together with the maximal depth of f in positive/negative literals. Count features allowed us to drop the clause length multiplier γ and to use the model prediction directly as the clause weight.

Horizontal Features provide a more accurate representation of clauses by feature vectors. For every term $f(t_1, \ldots, t_n)$, a new feature $f(s_1, \ldots, s_n)$ was introduced, where s_i is the top-level symbol of t_i. The number of occurrences of each horizontal feature is stored in the feature vector. Again, only the horizontal features that appear in the training data are considered. For example, the unit clause $plus(X, nul) = X$, yields horizontal features $= (plus, \circledast)$ and $plus(\circledast, nul)$, each occurring once in the clause.

Conjecture Features embed the conjecture to be proved in the feature vector. The first ENIGMA simply recommended the same clauses independently of the conjecture being proved. In the second ENIGMA, conjecture features were appended to the vector, making the vector twice the size. Thusly, the second ENIGMA was able to provide goal specific predictions, which was essential for experiments on Mizar problems which are much more heterogeneous than AIM benchmarks.

Another important step towards large data support in ENIGMA, was the implementation of *feature hashing*. This significantly reduced the feature vector size. ENIGMA uses a generic purpose string hashing function *sdbm*.[6] Each feature is represented by a unique string identifier, for example, $(=, plus, X)$ becomes "`|=|plus|*|`", and $plus(\circledast, nul)$ becomes "`.plus.*.nul.`". This string is passed through the hashing function, computed with a fixed-length data type representation (64 bit unsigned). The string hash modulo the selected *hash base* is used as the feature index. The hash base is intended to directly limit the vector size, at the price of occasional feature collisions.

In order to abstract from specific symbol names also in the context of features, the next ENIGMA [60] introduced a very simple method of *symbol anonymization*. During the extraction of clause features, all symbol names are replaced by symbol arities, keeping only the information whether the symbol is a function or a predicate. For example, the binary function symbol *plus* becomes simply *f2*, the ternary predicate symbol *ite* becomes *p3*, and so on. In this way, a decision tree classifier does not depend on symbol names, for the price of symbol collisions. While this rather trivial symbol anonymization was initially implemented mainly as a baseline for the graph neural networks, it performed surprisingly well in practice and it became a useful ENIGMA option. The symbol collisions reduce the size of training data, which is a favorable side effect of name anonymization.

3.3 More Semantic Features

The syntactic structure of mathematical statements does not always easily capture their intended meaning. For examples, the syntactic features of two complex formulas ϕ and $\neg\phi$ differ only very little, making them very close in various feature metrics. On the other hand, in the semantic Tarski-Lindenbaum algebra,

[6] Given the (code of the) i-th character s_i: $h_i = s_i + (h_{i-1} \ll 6) + (h_{i-1} \ll 16) - h_{i-1}$ with $h_0 = 0$.

these formulas are each other's complements, i.e., in some sense they are as distant as possible. To accommodate this, mechanisms for extracting features that are based on the meaning of formulae have been proposed.

The most common way of specifying the semantics of logical expressions in first-order logic is by considering which *models satisfy the expressions*. However, a formula may have many satisfying models. Thus, only *interesting* models that distinguish between formulas, are useful for characterization purposes. MaLARea-SG1 [156] used finite model finders to continuously search for countermodels for problems with too few recommended premises. The notion of model's interestingness is thus defined dynamically, based on the current state of the learned knowledge (the trained premise selector), with which it co-evolves. This led to considerable improvement of MaLARea's performance.

Another approach to obtain more semantic features considers the *unifiability of two formulas*. For most first-order automated reasoning calculi, the unifiability of formulas correlates with their use as principal formulas of an inference step. This means that first-order indexing structures [78] can be used as another source of more semantic features. The nodes of indexing structures, such as a substitution tree or discrimination tree, correspond to sets of unifiable formulas present in the given reasoning problem, while paths correspond to similar term structures. Thus, features corresponding to these nodes can characterize whether the given formulas can be combined for inferencing within a given calculus.

Latent semantic analysis (LSA) [29] is a method that creates low-rank vectors (*embeddings*) characterizing terms and documents based on the terms' co-occurrence in the documents. LSA-based features were used with some success in premise selection systems [77] and hammers [75]. LSA-based embeddings precede related more recent methods, which use the same idea of characterizing words by their context, such as Word2Vec [101] and neural embeddings.

3.4 Characterization Using Neural Networks

Already the early tree neural networks by Goller and Kuchler attempt to learn the representation of the symbols as neural sub-networks. More recently, starting with Word2Vec (a shallow neural network), a number of neural approaches have been experimented with for obtaining useful sentence and term embeddings.

For example, encoder-decoder neural frameworks attempt to summarize an input text (e.g., in English) in a vector that can be then decoded into suitable output (e.g., French text with corresponding meaning). A recent encoder-decoder approach was developed by Sutton et al. [3]. It focused on developing an embedding capturing semantic equivalence, i.e., formulas that are negations of each other are adequately distinguished. This research direction is connected to the work outlined in Section 8, where approximate reasoning and semantically rich encoding are used to build systems for various synthesis problems. Purgał [118] later developed an autoencoder for predicate logic formulas by training a network to decode a given formula's top symbol and its children. Another example is [166], which builds a representation of tactics by their meanings, i.e., how the tactic transforms a given state into several subsequent states.

An alternative to vectorization of symbolic expression as simple sequences of symbols is to use neural architectures that capture the structure of the expressions. This includes the pioneering work on Tree NNs by [51], capturing the recursive nature and tree-like structure of logical expressions. Related investigation is done in [35] for recognizing propositional logical entailment through evaluation of formulas in 'possible worlds', which is also related to the earlier model-based features in MaLARea. A related investigation [18] models expressions using a similar network architecture but instead aims to recognize properties such as *validity* through a top-down evaluation of the expression. The model thus tries to approximate decompositional reasoning.

An illustrative example of Tree NNs modeling symbolic expressions for the purpose of guiding an automated reasoner is *ENIGMA-NG* [20]. The authors provide a vector embedding that associates each predicate (function) symbol with a learned function $R^n \times \cdots \times R^n \to R^n$ where the number of arguments matches the symbol's arity. Thus, the term structure is encoded by the composition of these functions. Certain expressions whose semantic content cannot be directly extracted from the structure of the expression are grouped together. For example, all variables share the same encoding function. Clauses, rather than being considered as a composition of several instances of a binary *or* function, are handled by a separate RNN model capturing their set-like nature.

While Tree NNs capture the structure of symbolic expressions, a few issues remain. For example, the encoding used by *ENIGMA-NG* [20] requires the construction of a learnable function for each *symbol*, while all variables use a single learnable function. Graph neural networks (GNNs) [131] can provide an architecture able to abstract away the names used for the representation of symbolic expressions and, to some extent, distinguish variables and their occurrences. Early uses of GNNs for premise selection, such as FORMULANET [160], did not improve these deficiencies but provided direction towards a more semantic-preserving architecture. Olsak [106] then proposed a GNN-based name-invariant embedding using a hypergraph of clauses and terms, where none of the names appear and a symbol's meaning can only be inferred from the subgraph of its properties. While some loss of structure may still occur, for example, $f(t_1, t_2, t_1)$ and $f(t_2, t_1, t_2)$ are identically encoded, the embedding is a vast improvement over previous approaches, naturally treating also clauses as sets.

Such embeddings allow the neural architecture to draw analogies between different mathematical domains. Many algebraic operators are associative, commutative, and/or distributive, yet they have different names. The framework presented in [106] would recognize the local graph structure as analogous regardless of the different names used. While such neural architectures provide better generalization and improved cross-domain predictions, compared to simpler methods such as decision trees their predictions may be too confident. This problem was observed when integrating the above-mentioned GNN architecture into the inference selection mechanism of *plcop* [168]. The authors introduced *entropy regularization* to normalize the model's confidence and improve accuracy.

4 Premise Selection

The success of modern ATPs is partially due to their ability to select a small number of facts that are relevant to the conjecture. The act of selecting these relevant facts is referred to as *premise selection* [1], which can also be seen as a step in a more general abstraction-refinement framework [95]. Without good premise selection, ATPs can be easily overwhelmed by a number of possible deductions, which holds even for relatively simple conjectures. Non-learning ATP approaches include heuristics such as MePo and SInE, mentioned in Section 2. In this section, we cover the typically more precise learning approaches.

4.1 k-Nearest Neighbors (k-NN)

This approach to selection requires a measure of distance, more appropriately referred to as a *similarity relation*, between two facts. This similarity relation, in simple cases, is defined by the distance between two facts computed from a set of binary features. To increase precision, weights and a scaling factor are added to the features. Thus, the similarity relation between two facts is defined as follows:

$$s(a,b) = \sum\nolimits_{f \in F(a) \cap F(b)} w(f)^{\tau_1}$$

where F is the feature vector, w is the weight vector, and τ_1 a scaling factor.

We then realize that if a proof relies on more dependencies, each of them is less valuable, so we divide by the number of dependencies. We additionally add a factor for the dependencies themselves. Given N the set of the k nearest neighbours, and $D(b)$ the dependencies of fact b, and a scaling factor τ_2, the relevance of fact a for goal g is:

$$\left(\tau_2 \sum\nolimits_{b \in N \mid a \in D(b)} \frac{s(b,g)}{|D(b)|}\right) + \begin{cases} s(a,g) & \text{if } a \in N \\ 0 & \text{otherwise} \end{cases}$$

This approach was later extended to adaptive k, i.e., considering more or less neighbours depending on the requested number of best premises [75].

4.2 Naive Bayes

Given a fact a and a goal to prove g, we try to estimate the probability it is useful based on the features $f_1, ..., f_n$ of the goal g:

$$\begin{aligned} & P(a \text{ is relevant for proving } g) \\ = \; & P(a \text{ is relevant} \mid g\text{'s features}) \\ = \; & P(a \text{ is relevant} \mid f_1, \ldots, f_n) \\ \propto \; & P(a \text{ is relevant}) \Pi_{i=1}^{n} P(f_i \mid a \text{ is relevant}) \\ \propto \; & \#a \text{ is a proof dependency} \cdot \Pi_{i=1}^{n} \frac{\#f_i \text{ appears when } a \text{ is a proof dependency}}{\#a \text{ is a proof dependency}} \end{aligned}$$

This approach can be improved by considering features not present in the goal when a was used. An issue with considering the negative case is that there are too many possible features to take into account. So, instead of looking at all features not present when a was used, we only consider the so called *extended features* of a fact, namely the features that do not appear and are related to those that do appear [86]. This allows considering the probabilities of features not appearing in the goal while a dependency was used. These probabilities can be computed/estimated efficiently. Early versions of MaLARea used the implementation in the SNoW [15] toolkit. Kaliszyk later [71,72] implemented a custom naive Bayes for Flyspeck and other experiments.

4.3 Decision Trees

A *decision tree* is a binary tree with nodes labeled by conditions on the values of the feature vectors. Such trees and their *ensembles* are today among the strongest ML methods, which can also be very efficient. Initially, Färber [36] improved on k-NN approaches [71] with *random forests* (ensembles of decision trees) that used k-NN as a secondary classifier. With further modifications, random forests can also be applied directly as, e.g., in [112] where they are used for premise selection in Lean. *Gradient-boosted trees*, as implemented by XGBoost [17] and LightGBM [81] are useful both for premise selection and for efficient ATP guidance as discussed in the next section. Since they work in a binary (positive/negative) setting, they require (pseudo-)negative examples for training. Piotrowski's *ATPBoost* [113] defined an infinite MaLARea-style loop that interleaves their training with proving, producing increasingly better positive and negative data for the training. Quite surprisingly, *hashing* the large number (over millions) of sparse symbolic features (such as term walks) into much smaller space (e.g. 32000) allowed efficient training of these toolkits over very large libraries such as full MML, with practically no performance penalties [20,65].

4.4 Neural Methods

The first analysis of the performance of deep neural networks for premise selection was done by the *DeepMath* [2] project. Despite requiring significantly more resources, there was only limited improvement over the simpler methods. However, the work also employed the above mentioned *binary setting*, where conjectures and the potential premises were evaluated together. This allows to meaningfully evaluate premises that have not been seen yet. Methods such as k-NN and naive Bayes typically do not allow that without further data-augmentation tricks.[7] As logical formulas naturally have a tree structure, graph neural networks

[7] Such as adding for each premise as a training example its provability by itself, which has indeed been used from the beginnings of ML-based premise selection [150,151]. Data augmentation in general is another very interesting AI/TP topic, see e.g. [74].

were soon proposed for premise selection [160] (see also Section 3.4). Other alternatives such as graph sequence models [56], directed graph networks [122], recurrent neural networks [115] and transformers [155] have been experimented with, allowing to take into account also dependencies among the premises [115].

The signature independent GNN described in [106] seems to be the strongest method today, based on a recent large evaluation of many methods over the Mizar dataset [59]. The study also demonstrates the role of *ensembles and portfolios* of different premise selection methods. An interesting phenomenon observed in [155] is that some of the neural methods provide a smooth transition between premise selection and *conjecturing*, i.e., proposing so far unproved "premises" (intermediate conjectures) that split the problem into two [42]. Such neural conjecturing methods are further explored e.g. in [114,120,110,45,68].

5 Guidance of Saturation-based ATPs

Several decisions that ATPs need to regularly perform are very similar to premise selection. The selection of the next clause in saturation-based provers or of the next extension step in a connection tableaux calculus both correspond to the selection of relevant premises for a given conjecture. For this reason, many ATP guidance techniques are motivated by premise selection, with the additional caveat that efficiency needs to be much more taken into account.

Arguably the most important heuristic choice point in saturation-based theorem proving is *clause selection*. It is the procedure for deciding, at each iteration of the main proving loop, which will be the next clause to activate and thus participate in generating inferences. A perfect clause selection—obviously impossible to attain in practice—would mean selecting just the clauses of the yet-to-be-discovered proof and would thus completely eliminate search from the proving process. Although state-of-the-art clause selection heuristics are far from this goal, experiments show that even small improvements in its quality can have a huge impact on prover performance [139]. This makes clause selection the natural main target for machine-learned prover guidance.

The central idea behind improving clause selection by ML, which goes back at least to the early work of Schulz [134,31,136], is to learn from successes. One trains a binary classifier for recognizing as positive those clauses that appeared in previously discovered proofs and as negative the remaining selected ones. In subsequent runs, clauses classified positively are prioritized for selection.

Particular systems mainly differ in 1) which base prover they attempt to enhance, 2) how they represent clauses for the just described supervised learning task, and 3) which ML technique they use. When aiming to improve a prover in the most realistic, i.e., time constrained, setting, there are intricate trade-offs to be made between faithfulness of the chosen representation, capacity of the trained model and the speed in which the advice can be learned and retrieved. State-of-the-art provers are tightly optimized programs and a higher-quality advice may not lead to the best results if it takes too long to obtain.[8]

[8] This is not only because search and/or backtracking is an indispensable part of any reasoning which is not purely memorized, but also because self-learning systems that

For example, the ENIGMA (***E**fficient Lear**n**ing-Based **I**nference **G**uiding **Ma**chine*) system, extending the ATP E [138], started out with easy-to-compute syntactic clause features (e.g., term-walks, see Section 3.2) and a simple but very *efficient* linear classifier [62,63]. At the same time, Loos et al. [94], first experimenting with integrating state-of-the-art neural networks with E, discovered their models to be too slow to simply replace the traditional clause selection mechanism. A similar phenomenon appears in ML-based guidance of tactical ITPs (Section 7), where the relatively simple but fast predictors used in systems like TacticToe [44] (HOL4) and Tactician [11] (Coq) are hard to beat by more complicated but significantly slower neural guidance in a fair comparison [128]. In later versions of ENIGMA, these trade-offs were further explored by experimenting with gradient boosted trees and tree neural networks over term structure [20], and finally with graph neural networks [60]. In the meantime, Deepire [144,145], an extension of the prover Vampire [85] used tree neural networks but over clause derivation structure. In the rest of this section, we compare these and other systems also from other angles as we highlight some notable aspects of this interesting technology.

Integrating the learned advice. There are several possible ways in which the trained model M can be used to improve clause selection in a guided prover. Typically, one seeks to first turn the advice into a total order on clauses, e.g., for maintaining a queue data structure for a quick retrieval, and, in a second step, to somehow combine this order with the original clause selection heuristic.

In the most general case of a black-box binary classifier, M only suggests whether a clause is good (1) or bad (0). The first ENIGMA [62] used a linear combination of the M's classification and clause's length to come up with an overall score for sorting clauses. Not fully relying on M makes the advice more robust, in particular, it mitigates the damage from having large erroneously positively classified clauses. Another sensible strategy, adopted, for example, by ENIGMA-NG [20], is to have all positively classified clauses precede the negatively classified ones and to use clause generation time-stamps (sometimes referred to as *clause age*) as a tiebreaker on clauses in each of the two classes. This amounts to saying "prefer clauses suggested by M and among those apply the FIFO rule: old clauses first before younger ones". Finally, even a binary classifier often internally uses a continuous domain of assigned values (often referred to as *logits*) which are only turned into a binary decision by, e.g., getting compared against a fixed threshold. This is for instance true for models based on neural networks. ENIGMA anonymous [60], as well as the system by Loos et al. [94], used a simple comparison of clause logits for ordering their neural queue.

---

use faster guidance will produce more data to learn from. In today's ML, a slightly weaker learner trained on much more data will often be better than a slightly stronger learner trained on much less data. If a self-learning system uses a slightly weaker but much faster learner/predictor to do many iterations of proving and learning, the self-learning system will in a fixed amount of time produce much more data and its ultimate performance will thus be higher.

A clause queue ordered with the help of M (in one of the ways just described) can of course be used to simply replace any original clause selection heuristic in the prover. However, it is often better to combine the original heuristic and the learned one [20,144]. The simplest way of doing this is to alternate, in some pre-selected ratio, between selections from the various clause queues (one of them being the one based on M). A *layered clause selection* mechanism, which lead to the best performance of Deepire [144], applies the full original heuristic to clauses classified positively by M and alternates this with the original heuristic applied to all clauses. An advantage of this approach is that it allows for a *lazy evaluation* trick [144], under which not every clause available for selection has to immediately get evaluated by the relative expensive model M. Very recently, in the context of the iProver (instantiation-based) system, guidance using only the trained GNN model M managed to outperform the combination of M with the original heuristics [21]. One of the reasons for that is an improved learning from *dynamic* proof data, taking into account more proof states than the traditional ENIGMA-style training that only learns from the final proof states.

Signature (in-)dependence. First-order clauses, as syntactic objects, are built from predicate and function symbols specified by the input problem's *signature*. A representation of clauses for machine learning may consider this signature fixed, which is often natural and efficient, but ultimately ties the learned guidance to that signature. E.g., if we train our guidance on problems coming from set theory, the model will not be applicable to problems from other domains.

The early pattern-based guidance by Schulz for E was already signature independent, as well as, e.g., the early concept-alignment methods designed for transfer of knowledge between ITP libraries by Gauthier and Kaliszyk [41]. The ENIGMA systems [62,63,20] relied initially on the fixed signature approach. ENIGMA Anonymous [60] started to abstract from specific symbol identities and replace them with arity abstractions of the term walks and property invariant neural embeddings [106], opening doors to knowledge transfer. A detailed comparison of these ENIGMA abstraction mechanisms with the earlier ones by Schulz, Gauthier and Kaliszyk is discussed in [60] (Appendix B).

An interesting approach from this perspective, is taken by Deepire [144,145], which uses recursive neural networks for classifying the generated clauses based solely on their derivation history. Thus Deepire does not attempt to read "what a clause says", but only bases its decisions on "where a clause is coming from". This makes it trivially independent on problem signature, however, it still relies on a fixed initial axiom set over which all problems are formulated.

Building in context. A clause C useful for proving conjecture G_1 can be completely useless for proving a different conjecture G_2. While a great prover performance boost via learning can already be achieved *without* taking the conjecture context into account [145] (and, indeed, the standard clause selection heuristics are of this kind), many systems supply some representation of the conjecture as a secondary input to their model to improve the guidance [94,63,20].

Another, more subtle, kind of context is the information of how far the prover is in completing a particular proof. Intuitively, selecting a certain clause could only make sense if some previous clauses have already been selected. This is explored by ENIGMAWatch [49], where a proof state is approximated by a vector of *completion ratios* of a fixed set of previously discovered proofs. A system called TRAIL goes further and allows every processed clause to influence the score of any unprocessed clause through a multiplicative attention mechanism [24]. This level of generality, however, is computationally quite costly, and TRAIL does not manage to improve over plain E prover [138] under practical time constraints.

Looping and reinforcement. Once we train a model that successfully improves the performance of the base prover, we can collect even more proofs to train on and further improve our guidance in a subsequent iteration. Such re-learning from new proofs, as introduced by MaLARea [156], constitutes a powerful technique for tapping the full potential of a particular guidance architecture [65,145].

Iterative improvement is also at the core of the reinforcement learning (RL) approach [146], which builds on a different conceptual framework than the one we used so far (e.g., agent, action, state, reward – see also Section 6), but in the context of saturation-based provers gave so far rise to systems of comparable design [24,4]. The main reason for this is that even systems based on RL reward (and reinforce the selection of) clauses that appeared in the discovered proofs. The alternative of only rewarding the final proof-finishing step and letting the agent to distinguish the good from the bad through trial-and-error would be prohibitively expensive.

Beyond clause selection. Although clause selection has been the main focus of research on this front, there are many other ways in which machine learning can be used to improve saturation-based provers. For example, it is possible to predict good term orderings for the underlying superposition calculus [9] or symbol weights [10]. In the later case we still aim to influence clause selection via clause weight (one of the standard heuristics), but only indirectly, in an initialization phase, which comes before the proof search starts. Another class of approaches, which we only briefly mention here, use ML-style techniques for synthesis of ATP strategies and suggesting good targeted strategies or strategy schedules based on input problem features [153,133,87,64,55].

ML-based approaches have also been used to prevent interactions between unsuitable clauses [48], and in general to design iterative algorithms (*Split & Merge*) that repeatedly split the problems into separate reasoning components whose results are again merged after some time [19]. Such algorithms can also be seen as soft/learnable alternatives to ATP methods such as *splitting* [159], and to manual design of theory procedures and their combinations in SMT.

6 Guidance of Tableaux and Instantiation-based ATPs

While the saturation-based ATPs are today the strongest, the proofs produced by such systems are often at odds with human intuition. Tableaux-based provers

produce proofs closer in argumentation to human reasoning; this is likely due to the case-based style of the typical tableaux proof system. Essentially, such systems are performing a sort of model elimination rather than a search for contradiction. An advantage of such approaches is that the proof state is compactly represented, and it is easier to control the number of possible actions in comparison to saturation provers, where the number of mutually resolvable clauses can grow quickly. This has resulted in a lot of recent research in adding ML-based guidance in particular to the *connection tableaux calculus* [108].

First, the MaLeCoP (Machine Learning Connection Prover) system [157] used an external and relatively slow evaluation method to select the extension steps in the leanCoP ATP [107]. This showed that with good guidance, one can avoid 90% of the inferences. This was made much faster by integrating an efficient sparse naive Bayes classifier in an ML-guided OCAML re-implementation of leanCoP (FEMaLeCoP) [73]. As an alternative to the direct selection of *extension steps*, Monte Carlo simulations can be used to select the promising branches (MonteCoP) [37]. A major progress was obtained by removing the *iterative deepening* used by default in leanCoP, and instead using an Alpha-Zero-like architecture for guiding connection tableaux in *rlCoP* [76]. Reinforcement learning of *policy* (action, i.e. inference, selection) and *value* (state, i.e., partial tableaux, evaluation) and their use for intelligent subtree exploration yielded after several iterations of proving and learning on a training set a system that solves 40% more test problems than the default leanCoP strategy . The method still produces rather short proofs and policy guidance methods were later investigated for proofs with thousands of inference steps [167]. Extensions of connection tableaux, such as lazy paramodulation, can also be directly guided using similar methods [123]. Further improvements were recently achieved by integrating signature-independent graph neural networks (Section 3.4), e.g. [106,168].

Instantiation-based and SMT (Satisfiability Modulo Theories) systems such as iProver [83], CVC (CVC4/cvc5) [8] and Z3 [28] combine the use of SAT solvers for checking ground (un)satisfiability with various methods for producing suitable ground instantiations of the first-order problems. This approach goes back to the early days of ATP (Section 2) and methods such as Gilmore's [46], however they have recently become much more relevant thanks to today's powerful CDCL [143] based SAT solving and other calculus improvements.

In SMT, ML has so far been mainly used for tasks such as portfolio and strategy optimization [140,111,5]. More recent work [66,32,13] has also explored fast non-neural ML guiding methods based on decision trees and manual features. Very recently, the first neural methods for guiding the cvc5 SMT system have started to be developed [109]. Due to the large number of possible instantiations in such settings, this is typically more involved than guiding the clause selection as in the saturation based systems. Interestingly, iProver's instantiation-based calculus is also using the given clause loop, and an efficient neural guidance of its Inst-Gen [84] procedure has recently led to doubling of its performance on the Mizar corpus [21].

7 Tactic Based ITP Guidance

In most interactive theorem provers, proofs are written using meta-programs consisting of *tactics*. A tactic can analyze the current state of the proof and generate a sequence of kernel inference steps or a partial proof term to advance the proof. Actions performed by tactics can range from simple inference steps, to decision procedures, domain-specific heuristics, and even a generic proof search. As an alternative to ATP guidance of low-level steps, it is possible to recommend such tactics and explore the proof space using them. Recommendations are made by analyzing existing tactical proofs (either written by users or found automatically) and learning to predict which tactic performs a useful action on a given proof state. This *mid-level* guidance task falls anywhere between premise selection and the ATP guidance, allowing ML methods that may be slower than those used for ATPs.

7.1 Overview

Early systems in this field include ML4PG [82] for Coq, which gives tactic suggestions by clustering together various statistics extracted from interactive proofs, without trying to finish the proofs automatically. SEPIA [52] provides tactic predictions and also proof search for Coq, which is however only based on tactic traces without considering the proof state. The first system that considers proof states is TacticToe [43], which uses k-NN selection to predict the most likely tactics that complete a goal in the HOL4 proof assistant. Its later versions combined the prediction of promising tactics with Monte-Carlo tree search giving a very powerful method (66% of the library proved) for proof automation [44].

Similar systems have been created for Coq (Tactician [11]) and HOL Light (HOList [7]), as well as frameworks for the exploration of the tactical proof space in Lean [165]. The early PaMpeR [103] system for Isabelle also considers the proof states, however, it only recommends one command for each proof state, leaving its execution to the user (i.e., it lacks the search component). Further (sometimes experimental) systems for Coq include GamePad [58] and CoqGym [163], using deep neural networks and slow (600s) evaluation mode, as well as Proverbot9001 [129], which was shown to perform well on CompCert. Further Coq-oriented systems (TacTok, Diva, Passport) [40,39,130] are based on the dataset provided by CoqGym. For the Lean proof assistant, further systems include LeanDojo [164] and [89].

7.2 Advantages of Tactic-based ITP Guidance

There are several advantages associated with using tactics for proof search.

Adaptivity. Tactics provide a higher level and more flexible base for proof search. Generating kernel inference steps or partial proof terms directly is a task that needs high precision. On the other hand, tactics are often adaptive and can perform a sensible high-level action, such as search or decision procedures, on a wider variety of proof states.

Specialization. In highly specialized branches of mathematics, tactics are often specialized to the domain by experts. That is, when the default set of tactics provided by a proof assistant is not satisfactory, a new set of tactics may be written by end-users. This is particularly useful when the mathematical domain requires a deep embedding of a custom logic. An example of this is the Iris separation logic framework [70] for Coq. It includes a set of tactics specifically crafted for working with separation logic. Another example is the CakeML project [88] for HOL using custom tactics, which provides a formally verified compiler for the ML language. Some tactic-based proof search methods, such as Tactician's k-NN model for Coq [11] can learn to use new tactics in real time, which has proven rather powerful [128].

7.3 Challenges in Tactic-based ITP Guidance

Potential Incompleteness. Contrary to performing basic inference steps for a logic (as e.g. in the ATP calculi), tactics might not represent a minimal and complete set of inference rules. A set of tactics may not be guaranteed to be able to prove every theorem.[9]

Overlap in functionality. The actions taken by different tactics may have a high degree of overlap. Systems like Proverbot9001 [129] and Tactician [11] attempt to mitigate tactic overlap by *decomposing* and *normalizing* tactic expressions, attempting to eliminate duplicate tactics.

In TacticToe, the *orthogonalization* process is introduced to eliminate redundant tactics [44]. In more detail, TacticToe maintains a database of goal-tactic pairs used for training and this database is subject to the orthogonalization process which is intertwined with the learning. Orthogonalization works as follows: First, each time a new tactic-goal pair (t, g) is extracted from a tactic proof and about to be recorded in the database, we consider if there already exists a better tactic for the goal g in the database. To this end, we organize a competition between the k tactic-goal pairs that are closest[10] to the pair (t,g) (including it). The winner (which is ultimately stored in the database and trained on) is the tactic that subsumes the original tactic t on the goal g and that appears in the largest number of tactic-goal pairs in the database. As a result, already successful tactics with a large coverage are preferred, and new tactics are considered only if they provide a different contribution.

Diverse Tactic Behavior. Due to the diverse range of tactic behaviors it can be difficult to tune proof search to appropriately exploit each class of tactics. For example, while simple tactics are executed rather quickly, more sophisticated tactics may require multiple seconds of execution time before their action completes. Deciding which tactic to execute and the appropriate allocation of resources to it is a major challenge.

[9] This is a theoretical concern that usually is not a problem in practice.

[10] He we already use the learned notion of proximity on the database constructed so far.

Tactics Are Designed for Humans. Because tactics are designed for use by humans, they usually come with a complex language of tactic combinators, higher-order language features, and syntactic sugar. Typically, in machine learning and reinforcement learning we however prefer to have a relatively simple set of actions (commands in the ITP setting). Therefore one would like to decompose the proof scripts into sequences (or more generally trees) of simpler commands. This is a difficult preprocessing step.

Representation. Simpler tactic-based systems make predictions solely based on surface-level syntax features of proof states. This makes it difficult to extract deeper knowledge about the previously introduced concepts. Also, straightforward neural encodings are typically insufficient when new concepts and lemmas are added on the fly (which is very common in ITP), because it is expensive to always adapt a large neural model after the addition of such new items. The most recent work on Graph2Tac [128] learns representations of all definitions in its dataset and can generate representations for unseen definitions on-the-fly. This provides more accurate representations of proof states and tactic arguments, resulting in a 50% improvement over baselines that do not incorporate such background information.

8 Related Symbolic Classification Problems

The majority of investigations discussed above concern the classification of symbolic expressions with the goal of informing a symbolic system, which of a variety of actions is most likely to result in success. A few exceptions were discussed in Section 3.4, for example [35,18]. Both papers present an embedding and a neural architecture solving a classification problem but with no intention of integration within a symbolic system. While this positions such investigations quite far from the core topic of this Survey, note that these works motivated the approach presented in [20] where the authors used a tree NN to guide clause selection. Thus, it is likely that future developments improving precise selection and guidance will be motivated by approaches developed for other symbolic classification problems.

Relatively recent investigations have considered the introduction of probabilities into logic programs [38], i.e., some facts are associated with probability. Such logic programs allow the introduction of predicates whose definition is a neural network. The authors of *DeepProbLog* [96,97], introduce an approach to train so-called *neural predicates* within the context of a probabilistic logic program. In some sense, this can be viewed as a form of *symbolic guidance* of the training procedure of a statistical model. While the majority of this Survey, including the section on neural classification, focuses on guidance, a few investigations have considered *end-to-end* neural theorem proving [127], that perform a soft unification operation over a vector representation of the proof state and atoms to unify. Investigations have also considered the use of *generative adversarial networks* to train a prover and a teacher simultaneously with the goal of teaching the prover to solve the problems presented by the teacher [117].

ILP is a form of symbolic machine learning whose goal is to derive explanatory hypotheses from sets of examples (denoted E^+ and E^-) together with background knowledge (denoted BK) [23]. Essentially, it is a form of inductive synthesis. Early approaches to providing a statistical characterization of ILP include *nFOIL* [90], which models the search procedure and stopping conditions of FOIL [119] using Naive Bayes. Essentially, nFOIL attempts to maximize the probability that a hypothesis covers the examples. The stopping condition is the score function reaching a user-defined threshold. A similar approach was investigated by integrating Kernel methods and FOIL [91].

More recently, Evans et al. [34] introduced *δILP* which considers ILP as a satisfiability problem where each proposition denotes a pair of clauses constructable using the *background knowledge*. These clause pairs are used as the definition *predicated template*. The hypothesis consists of a user-defined number of predicated templates. Essentially, a propositional model of this SAT problem is a hypothesis. As a next step, the authors introduce a type of soft inferencing used to compute the evaluations of the propositions composing the SAT problem. Investigations based on the δILP have considered a hierarchical structure of templating [47] and massive predicate invention [116]. Recent work by some of the Authors of δILP illustrates that such *neuro-symbolic* system can be interleaved with a symbolic solver, allowing them to provide feedback during training [33].

While δILP requires learning a propositional model for a SAT encoding of an ILP instance through soft inferencing, the framework is not directly viable as a general *neural* SAT solver. NeuroSAT [141] embeds a SAT instance into a GNN that learns which propositions are required for satisfiability. A more recent approach to neural SAT solving is the SATNet [161,16,147] where satisfiability is formulated as a semidefinite program.

9 Conclusion

In this work, we discussed the main contemporary areas of combining automated reasoning and especially automated theorem proving with machine learning. This includes the early history, characterization of mathematical knowledge, premise selection, ATPs that use machine learning, feedback loops between proving and learning, and some related symbolic classification problems.

Recently, (large) language models (LMs/LLMs) have shown the ability to generate (not necessarily correct) informal math texts. Beyond high-school tasks or mathematics olympics tasks (solutions for which are abundant on the web), their ATP performance in fair-resource evaluations has, however, so far been questionable compared to targeted architectures such as signature-independent graph neural networks [128]. Other research questions currently debated in the context of LLMs are, e.g., the (lack of) *emergence* [132] and the *memorization* of all the benchmark problems, proofs and their informal presentations available on the web [68].[11] Perhaps the most promising uses of (not necessarily large)

[11] In particular, while an exact formal proof of a solved problem P may or may not be available on the web or GitHub (and thus in the LLM training data), it is still quite

language models today seem to be the conjecturing tasks mentioned in Section 4.4 (e.g., [155,120,45,68]), and in autoformalization [80,79,162,67].

On the other hand, systems like *ChatGPT* have recently convinced a lot of lay and expert audience about the potential of AI systems trained over a lot of informal knowledge. It is likely that this will lead to increased efforts in training AI/TP systems, which not only absorb a lot of knowledge, but also learn to use it correctly and are capable of self-improvement and new discoveries without hallucination, thanks to the ground logical layer. The brief history of the AI/TP field so far demonstrates that perhaps the most interesting systems and research directions emerge when the deductive, search and symbolic methods are in nontrivial ways combined with the learning, inductive and statistical methods, leading to complex and novel AI architectures. In this sense, the future of AI/TP research seems to be bright and very open.

Acknowledgments

We thank Herman Geuvers for creating an environment in his group in Nijmegen that allowed starting and continuing several of the discussed research directions. We also thank the people in Nijmegen with whom we have over the years collaborated on these topics and discussed them, including Tom Heskes, Daniel Kühlwein, Twan van Laarhoven, Evgeni Tsivtsivadze, Jelle Piepenbrock, Jan Heemstra, Freek Wiedijk, Robbert Krebbers and Henk Barendregt.

Finally, we thank the whole AITP community involved in developing these topics, and especially our AI/TP collaborators and colleagues (often from Prague and Innsbruck), including Bob Veroff, Stephan Schulz, Konstantin Korovin, Sean Holden, Jasmin Blanchette, Chad Brown, Mikoláš Janota, Karel Chvalovský, Jiří Vyskočil, Petr Pudlák, Petr Štěpánek, Mirek Olšák, Zsolt Zombori, Geoff Sutcliffe, Christian Szegedy, Tom Hales, Adam Pease, Moa Johansson, Sarah Winkler, Sara Loos, Ramana Kumar, Michael Douglas, Mario Carneiro, Bartosz Piotrowski, Yutaka Nagashima, Zar Goertzel, Michael Faerber, Shawn Wang, Jan Hůla, Filip Bártek and Liao Zhang.

This work has been partially supported by the COST Action CA20111 (CK, DC, MS), ERC PoC grant no. 101156734 *FormalWeb3* (CK, JJ), project RICAIP no. 857306 under the EU-H2020 programme (MS), project CORESENSE no. 101070254 under the Horizon Europe programme (MS), ERC-CZ grant no. LL1902 POSTMAN (JJ, TG, JU), Amazon Research Awards (LB, TG, JU), the EU ICT-48 2020 project TAILOR no. 952215 (LB, JU), the ELISE EU ICT-48 project's (no. 951847) Mobility Program (LB), and the Czech Science Foundation project 24-12759S (MS).

likely that an informal proof with the essential proof ideas for P has been seen during the LLM training. In such cases, the LLM is more likely doing translation (autoformalization) rather than the usual proving of previously unseen problems. Similar issues appear with methods such as (pre-)training on "synthetic" problems, which may be generated in ways that make them close to the target "unseen" problems.

References

1. J. Alama, T. Heskes, D. Kühlwein, E. Tsivtsivadze, and J. Urban. Premise selection for math by corpus analysis and kernel methods. *JAR*, 52(2):191–213, 2014.
2. A. A. Alemi, F. Chollet, N. Eén, G. Irving, C. Szegedy, and J. Urban. DeepMath - deep sequence models for premise selection. In *NIPS 2016*, pages 2235–2243, 2016.
3. M. Allamanis, P. Chanthirasegaran, P. Kohli, and C. Sutton. Learning continuous semantic representations of symbolic expressions. In *ICML 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 80–88. PMLR, 2017.
4. E. Aygün, A. Anand, L. Orseau, X. Glorot, S. M. McAleer, V. Firoiu, L. M. Zhang, D. Precup, and S. Mourad. Proving theorems using incremental learning and hindsight experience replay. In *ICML 2022*, volume 162, pages 1198–1210, 2022.
5. M. Balunovic, P. Bielik, and M. T. Vechev. Learning to solve SMT formulas. In *NeurIPS*, pages 10338–10349, 2018.
6. G. Bancerek, C. Bylinski, A. Grabowski, A. Kornilowicz, R. Matuszewski, A. Naumowicz, K. Pak, and J. Urban. Mizar: State-of-the-art and beyond. In *CICM*, volume 9150 of *LNCS*, pages 261–279. Springer, 2015.
7. K. Bansal, S. M. Loos, M. N. Rabe, C. Szegedy, and S. Wilcox. Holist: An environment for machine learning of higher order logic theorem proving. In *ICML 2019*, volume 97, pages 454–463. PMLR, 2019.
8. H. Barbosa, C. W. Barrett, M. Brain, G. Kremer, H. Lachnitt, M. Mann, A. Mohamed, M. Mohamed, A. Niemetz, A. Nötzli, A. Ozdemir, M. Preiner, A. Reynolds, Y. Sheng, C. Tinelli, and Y. Zohar. cvc5: A versatile and industrial-strength SMT solver. In *TACAS (1)*, volume 13243. Springer, 2022.
9. F. Bártek and M. Suda. Neural precedence recommender. In *CADE 28*, volume 12699 of *LNCS*, pages 525–542. Springer, 2021.
10. F. Bártek and M. Suda. How much should this symbol weigh? A GNN-advised clause selection. In *LPAR 2023*, volume 94 of *EPiC*, pages 96–111. EasyChair, 2023.
11. L. Blaauwbroek, J. Urban, and H. Geuvers. The Tactician - a seamless, interactive tactic learner and prover for Coq. In *CICM 2020*, volume 12236 of *LNCS*, pages 271–277. Springer, 2020.
12. J. C. Blanchette, C. Kaliszyk, L. C. Paulson, and J. Urban. Hammering towards QED. *J. Formalized Reasoning*, 9(1):101–148, 2016.
13. J. C. Blanchette, D. E. Ouraoui, P. Fontaine, and C. Kaliszyk. Machine Learning for Instance Selection in SMT Solving. In *AITP 2019 - 4th Conference on Artificial Intelligence and Theorem Proving*, Obergurgl, Austria, Apr. 2019.
14. W. W. Bledsoe, R. S. Boyer, and W. H. Henneman. Computer proofs of limit theorems. *Artificial Intelligence*, 3:27–60, 1972.
15. A. Carlson, C. Cumby, J. Rosen, and D. Roth. The SNoW learning architecture, vol. 5 technical report. UIUCDCS-R-99-2101, UIUC Computer Science Department, 1999.
16. O. Chang, L. Flokas, H. Lipson, and M. Spranger. Assessing SATNet's ability to solve the symbol grounding problem. In *NeurIPS 2020*, volume 33, pages 1428–1439, 2020.
17. T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Knowledge Discovery and Data Mining 2016*, pages 785–794. ACM, 2016.

18. K. Chvalovský. Top-down neural model for formulae. In *ICLR 2019*. OpenReview.net, 2019.
19. K. Chvalovský, J. Jakubův, M. Olsák, and J. Urban. Learning theorem proving components. In *TABLEAUX*, volume 12842 of *LNCS*, pages 266–278, 2021.
20. K. Chvalovský, J. Jakubův, M. Suda, and J. Urban. ENIGMA-NG: efficient neural and gradient-boosted inference guidance for E. In *CADE 27*, volume 11716 of *LNCS*, pages 197–215. Springer, 2019.
21. K. Chvalovský, K. Korovin, J. Piepenbrock, and J. Urban. Guiding an instantiation prover with graph neural networks. In *LPAR 2023*, volume 94 of *EPiC Series in Computing*, pages 112–123. EasyChair, 2023.
22. S. Colton, A. Bundy, and T. Walsh. Automatic concept formation in pure mathematics. In *IJCAI*, pages 786–793. Morgan Kaufmann, 1999.
23. A. Cropper and S. Dumancic. Inductive logic programming at 30: A new introduction. *J. Artif. Intell. Res.*, 74:765–850, 2022.
24. M. Crouse, I. Abdelaziz, B. Makni, S. Whitehead, C. Cornelio, P. Kapanipathi, K. Srinivas, V. Thost, M. Witbrock, and A. Fokoue. A deep reinforcement learning approach to first-order logic theorem proving. In *AAAI 2021*, pages 6279–6287.
25. I. Dahn and C. Wernhard. First order proof problems extracted from an article in the MIZAR Mathematical Library. In *Int. Workshop on First-Order Theorem Proving (FTP'97)*, pages 58–62, 1997.
26. M. Davis. The early history of automated deduction. *Handbook of automated reasoning*, 1:3–15, 2001.
27. M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM (JACM)*, 7(3):201–215, 1960.
28. L. M. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In C. R. Ramakrishnan and J. Rehof, editors, *TACAS*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.
29. S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *JASIS*, 41(6):391–407, 1990.
30. J. Denzinger, M. Fuchs, C. Goller, and S. Schulz. Learning from previous proof experience. Technical Report AR99-4, Institut für Informatik, TUM, 1999.
31. J. Denzinger and S. Schulz. Learning domain knowledge to improve theorem proving. In *CADE 13*, number 1104 in LNAI, pages 62–76, 1996.
32. D. El Ouraoui. *Méthodes pour le raisonnement d'ordre supérieur dans SMT, Chapter 5.* PhD thesis, Université de Lorraine, 2021.
33. R. Evans, M. Bošnjak, L. Buesing, K. Ellis, D. Pfau, P. Kohli, and M. Sergot. Making sense of raw input. *Artificial Intelligence*, 299:103521, 2021.
34. R. Evans and E. Grefenstette. Learning explanatory rules from noisy data. *J. Artif. Intell. Res.*, 61:1–64, 2018.
35. R. Evans, D. Saxton, D. Amos, P. Kohli, and E. Grefenstette. Can neural networks understand logical entailment? In *ICLR 2018*. OpenReview.net, 2018.
36. M. Färber and C. Kaliszyk. Random forests for premise selection. In *FroCoS 2015*, volume 9322 of *LNCS*, pages 325–340. Springer, 2015.
37. M. Färber, C. Kaliszyk, and J. Urban. Machine learning guidance for connection tableaux. *J. Autom. Reason.*, 65(2):287–320, 2021.
38. D. Fierens, G. V. den Broeck, J. Renkens, D. S. Shterionov, B. Gutmann, I. Thon, G. Janssens, and L. D. Raedt. Inference and learning in probabilistic logic programs using weighted boolean formulas. *Theory Pract. Log. Prog.*, 15(3):358–401, 2015.

39. E. First and Y. Brun. Diversity-driven automated formal verification. In *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22, page 749–761, New York, NY, USA, 2022. Association for Computing Machinery.
40. E. First, Y. Brun, and A. Guha. Tactok: Semantics-aware proof synthesis. *Proc. ACM Program. Lang.*, 4(OOPSLA), nov 2020.
41. T. Gauthier and C. Kaliszyk. Aligning concepts across proof assistant libraries. *J. Symb. Comput.*, 90:89–123, 2019.
42. T. Gauthier, C. Kaliszyk, and J. Urban. Initial experiments with statistical conjecturing over large formal corpora. In *WIP@CIKM'16*, volume 1785 of *CEUR*, pages 219–228, 2016.
43. T. Gauthier, C. Kaliszyk, and J. Urban. TacticToe: Learning to reason with HOL4 tactics. In *LPAR-21*, pages 125–143, 2017.
44. T. Gauthier, C. Kaliszyk, J. Urban, R. Kumar, and M. Norrish. Tactictoe: Learning to prove with tactics. *J. Autom. Reason.*, 65(2):257–286, 2021.
45. T. Gauthier, M. Olsák, and J. Urban. Alien coding. *Int. J. Approx. Reason.*, 162:109009, 2023.
46. P. C. Gilmore. A proof method for quantification theory: its justification and realization. *IBM Journal of research and development*, 4(1):28–35, 1960.
47. C. Glanois, Z. Jiang, X. Feng, P. Weng, M. Zimmer, D. Li, W. Liu, and J. Hao. Neuro-symbolic hierarchical rule induction. In *ICML 2022*, pages 7583–7615.
48. Z. A. Goertzel, K. Chvalovský, J. Jakubův, M. Olšák, and J. Urban. Fast and slow enigmas and parental guidance. In *FroCoS 2021*, volume 12941 of *LNCS*, pages 173–191. Springer, 2021.
49. Z. A. Goertzel, J. Jakubův, and J. Urban. ENIGMAWatch: ProofWatch meets ENIGMA. In *TABLEAUX 2019*, volume 11714 of *LNCS*, pages 374–388, 2019.
50. C. Goller. Learning search-control heuristics with folding architecture networks. In *ESANN 1999*, pages 45–50, 1999.
51. C. Goller and A. Küchler. Learning task-dependent distributed representations by backpropagation through structure. In *ICNN'96*, pages 347–352. IEEE, 1996.
52. T. Gransden, N. Walkinshaw, and R. Raman. SEPIA: search for proofs using inferred automata. In *CADE*, volume 9195 of *LNCS*, pages 246–255, 2015.
53. K. Hoder, G. Reger, M. Suda, and A. Voronkov. Selecting the selection. In *IJCAR 2016*, volume 9706 of *LNCS*, pages 313–329. Springer, 2016.
54. K. Hoder and A. Voronkov. Sine qua non for large theory reasoning. In *CADE*, volume 6803 of *LNCS*, pages 299–314. Springer, 2011.
55. E. K. Holden and K. Korovin. Heterogeneous heuristic optimisation and scheduling for first-order theorem proving. In *CICM 2021*, volume 12833 of *LNCS*, pages 107–123. Springer, 2021.
56. E. K. Holden and K. Korovin. Graph sequence learning for premise selection. *CoRR*, abs/2303.15642, 2023.
57. S. B. Holden. Machine learning for automated theorem proving: Learning to solve SAT and QSAT. *Found. Trends Mach. Learn.*, 14(6):807–989, 2021.
58. D. Huang, P. Dhariwal, D. Song, and I. Sutskever. Gamepad: A learning environment for theorem proving. In *ICLR*, 2019.
59. J. Jakubův, K. Chvalovský, Z. A. Goertzel, C. Kaliszyk, M. Olšák, B. Piotrowski, S. Schulz, M. Suda, and J. Urban. MizAR 60 for Mizar 50. In *ITP 2023*, volume 268 of *LIPIcs*, pages 19:1–19:22, 2023.
60. J. Jakubův, K. Chvalovský, M. Olšák, B. Piotrowski, M. Suda, and J. Urban. ENIGMA Anonymous: Symbol-independent inference guiding machine (system description). In *IJCAR 2020*, volume 12167 of *LNCS*, pages 448–463. Springer, 2020.

61. J. Jakubův and C. Kaliszyk. Unified ordering for superposition-based automated reasoning. In *ICMS 2018*, volume 10931 of *LNCS*, pages 245–254. Springer, 2018.
62. J. Jakubův and J. Urban. ENIGMA: Efficient learning-based inference guiding machine. In *CICM 2017*, volume 10383 of *LNCS*, pages 292–302. Springer, 2017.
63. J. Jakubův and J. Urban. Enhancing ENIGMA given clause guidance. In *CICM 2018*, volume 11006 of *LNCS*, pages 118–124. Springer, 2018.
64. J. Jakubův and J. Urban. Hierarchical invention of theorem proving strategies. *AI Commun.*, 31(3):237–250, 2018.
65. J. Jakubův and J. Urban. Hammering Mizar by learning clause guidance (short paper). In *ITP 2019*, volume 141 of *LIPIcs*, pages 34:1–34:8, 2019.
66. M. Janota, J. Piepenbrock, and B. Piotrowski. Towards learning quantifier instantiation in SMT. In K. S. Meel and O. Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, August 2-5, 2022, Haifa, Israel*, volume 236 of *LIPIcs*, pages 7:1–7:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
67. A. Q. Jiang, S. Welleck, J. P. Zhou, T. Lacroix, J. Liu, W. Li, M. Jamnik, G. Lample, and Y. Wu. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In *ICLR*, 2023.
68. M. Johansson and N. Smallbone. Exploring mathematical conjecturing with large language models. In *NeSy*, volume 3432 of *CEUR Workshop Proceedings*, pages 62–77. CEUR-WS.org, 2023.
69. R. J. B. Jr. and R. Schrag. Using CSP look-back for real-world SAT instances. In *AAAI 97*, pages 203–208. AAAI Press / The MIT Press, 1997.
70. R. Jung, R. Krebbers, J. Jourdan, A. Bizjak, L. Birkedal, and D. Dreyer. Iris from the ground up: A modular foundation for higher-order concurrent separation logic. *J. Funct. Program.*, 28:e20, 2018.
71. C. Kaliszyk and J. Urban. Stronger automation for Flyspeck: Feature weighting and strategy evolution. In *PxTP 2013*, volume 14 of *EPiC Series in Computing*, pages 87–95. EasyChair, 2013.
72. C. Kaliszyk and J. Urban. Learning-assisted automated reasoning with Flyspeck. *J. Autom. Reason.*, 53(2):173–213, 2014.
73. C. Kaliszyk and J. Urban. FEMaLeCoP: Fairly efficient machine learning connection prover. In *LPAR-20 2015*, volume 9450 of *LNCS*, pages 88–96. Springer, 2015.
74. C. Kaliszyk and J. Urban. Learning-assisted theorem proving with millions of lemmas. *J. Symb. Comput.*, 69:109–128, 2015.
75. C. Kaliszyk and J. Urban. MizAR 40 for Mizar 40. *J. Autom. Reason.*, 55(3):245–256, 2015.
76. C. Kaliszyk, J. Urban, H. Michalewski, and M. Olšák. Reinforcement learning of theorem proving. In *NeurIPS 2018*, pages 8836–8847, 2018.
77. C. Kaliszyk, J. Urban, and J. Vyskočil. Machine learner for automated reasoning 0.4 and 0.5. In *PAAR@IJCAR*, volume 31 of *EPiC*, pages 60–66, 2014.
78. C. Kaliszyk, J. Urban, and J. Vyskočil. Efficient semantic features for automated reasoning over large theories. In *IJCAI 2015*, pages 3084–3090. AAAI Press, 2015.
79. C. Kaliszyk, J. Urban, and J. Vyskočil. Automating formalization by statistical and semantic parsing of mathematics. In *ITP*, volume 10499 of *LNCS*, pages 12–27, 2017.
80. C. Kaliszyk, J. Urban, J. Vyskočil, and H. Geuvers. Developing corpus-based translation methods between informal and formal mathematics: Project description. In *CICM 2014*, volume 8543 of *LNCS*, pages 435–439, 2014.

81. G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *NeurIPS 2017*, pages 3146–3154, 2017.
82. E. Komendantskaya, J. Heras, and G. Grov. Machine learning in Proof General: Interfacing interfaces. In *UITP*, volume 118 of *EPTCS*, pages 15–41, 2012.
83. K. Korovin. iProver - an instantiation-based theorem prover for first-order logic (system description). In A. Armando, P. Baumgartner, and G. Dowek, editors, *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings*, volume 5195 of *Lecture Notes in Computer Science*, pages 292–298. Springer, 2008.
84. K. Korovin. Inst-gen - A modular approach to instantiation-based automated reasoning. In *Programming Logics*, volume 7797 of *Lecture Notes in Computer Science*, pages 239–270. Springer, 2013.
85. L. Kovács and A. Voronkov. First-order theorem proving and Vampire. In *CAV*, volume 8044 of *LNCS*, pages 1–35. Springer, 2013.
86. D. Kühlwein, J. C. Blanchette, C. Kaliszyk, and J. Urban. Mash: Machine learning for Sledgehammer. In *ITP 2013*, volume 7998 of *LNCS*, pages 35–50. Springer, 2013.
87. D. Kühlwein and J. Urban. MaLeS: A framework for automatic tuning of automated theorem provers. *J. Autom. Reason.*, 55(2):91–116, 2015.
88. R. Kumar, M. O. Myreen, M. Norrish, and S. Owens. CakeML: A verified implementation of ML. In *Principles of Programming Languages (POPL)*, pages 179–191. ACM Press, Jan. 2014.
89. G. Lample, T. Lacroix, M. Lachaux, A. Rodriguez, A. Hayat, T. Lavril, G. Ebner, and X. Martinet. Hypertree proof search for neural theorem proving. In *NeurIPS*, 2022.
90. N. Landwehr, K. Kersting, and L. D. Raedt. nFOIL: Integrating naïve bayes and FOIL. In *AAAI 2005*, pages 795–800. AAAI Press / The MIT Press, 2005.
91. N. Landwehr, A. Passerini, L. D. Raedt, and P. Frasconi. kFOIL: Learning simple relational kernels. In *AAAI 2006*, pages 389–394. AAAI Press, 2006.
92. P. Langley. BACON: A production system that discovers empirical laws. In *International Joint Conference on Artificial Intelligence*, 1977.
93. D. Lenat. *An Artificial Intelligence Approach to Discovery in Mathematics*. PhD thesis, Stanford University, Stanford, USA, 1976.
94. S. M. Loos, G. Irving, C. Szegedy, and C. Kaliszyk. Deep network guided proof search. In *LPAR-21*, volume 46 of *EPiC Series in Computing*, pages 85–105. EasyChair, 2017.
95. J. C. López-Hernández and K. Korovin. An abstraction-refinement framework for reasoning with large theories. In *IJCAR*, volume 10900 of *LNCS*, pages 663–679, 2018.
96. R. Manhaeve, S. Dumancic, A. Kimmig, T. Demeester, and L. D. Raedt. DeepProbLog: Neural probabilistic logic programming. In *NeurIPS 2018*, pages 3753–3763, 2018.
97. R. Manhaeve, S. Dumancic, A. Kimmig, T. Demeester, and L. D. Raedt. Neural probabilistic logic programming in DeepProbLog. *Artif. Intell.*, 298:103504, 2021.
98. W. McCune. OTTER 2.0. In *CADE 1990*, volume 449 of *LNCS*, pages 663–664. Springer, 1990.
99. J. Meng and L. C. Paulson. Experiments on supporting interactive proof using resolution. In *IJCAR*, volume 3097 of *LNCS*, pages 372–384. Springer, 2004.
100. J. Meng and L. C. Paulson. Lightweight relevance filtering for machine-generated resolution problems. *J. Applied Logic*, 7(1):41–57, 2009.

101. T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.
102. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *DAC 2001*, pages 530–535. ACM, 2001.
103. Y. Nagashima and Y. He. Pamper: proof method recommendation system for isabelle/hol. In M. Huchard, C. Kästner, and G. Fraser, editors, *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, pages 362–372. ACM, 2018.
104. A. Newell and H. Simon. The logic theory machine–a complex information processing system. *IRE Trans. on information theory*, 2(3):61–79, 1956.
105. R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In *Handbook of Automated Reasoning (in 2 volumes)*, pages 371–443. 2001.
106. M. Olšák, C. Kaliszyk, and J. Urban. Property invariant embedding for automated reasoning. In *ECAI 2020*, volume 325, pages 1395–1402. IOS Press, 2020.
107. J. Otten and W. Bibel. leanCoP: lean connection-based theorem proving. *J. Symb. Comput.*, 36(1-2):139–161, 2003.
108. J. Otten and W. Bibel. Advances in connection-based automated theorem proving. In *Provably Correct Systems*, NASA Monographs in Systems and Software Engineering, pages 211–241. Springer, 2017.
109. J. Piepenbrock, M. Janota, J. Urban, and J. Jakubův. First experiments with neural cvc5. http://grid01.ciirc.cvut.cz/~mptp/cvc5-gnn.pdf, 2024.
110. J. Piepenbrock, J. Urban, K. Korovin, M. Olšák, T. Heskes, and M. Janota. Machine learning meets the Herbrand universe. *CoRR*, abs/2210.03590, 2022.
111. N. Pimpalkhare, F. Mora, E. Polgreen, and S. A. Seshia. MedleySolver: Online SMT algorithm selection. In C.-M. Li and F. Manyà, editors, *Theory and Applications of Satisfiability Testing – SAT 2021*, pages 453–470, Cham, 2021. Springer International Publishing.
112. B. Piotrowski, R. F. Mir, and E. W. Ayers. Machine-learned premise selection for Lean. In *TABLEAUX 2023*, volume 14278 of *LNCS*, pages 175–186. Springer, 2023.
113. B. Piotrowski and J. Urban. ATPboost: Learning premise selection in binary setting with ATP feedback. In *IJCAR 2018*, volume 10900 of *LNCS*, pages 566–574, 2018.
114. B. Piotrowski and J. Urban. Guiding inferences in connection tableau by recurrent neural networks. In *CICM 2020*, volume 12236 of *LNCS*, pages 309–314, 2020.
115. B. Piotrowski and J. Urban. Stateful premise selection by recurrent neural networks. In *LPAR 2020*, volume 73 of *EPiC*, pages 409–422. EasyChair, 2020.
116. S. J. Purgal, D. M. Cerna, and C. Kaliszyk. Differentiable inductive logic programming in high-dimensional space. *CoRR*, abs/2208.06652, 2022.
117. S. J. Purgal and C. Kaliszyk. Adversarial learning to reason in an arbitrary logic. In *FLAIRS 2022*, 2022.
118. S. J. Purgal, J. Parsert, and C. Kaliszyk. A study of continuous vector representations for theorem proving. *J. Log. Comput.*, 31(8):2057–2083, 2021.
119. J. R. Quinlan. Learning logical definitions from relations. *Mach. Learn.*, 5:239–266, 1990.
120. M. N. Rabe, D. Lee, K. Bansal, and C. Szegedy. Mathematical reasoning via self-supervised skip-tree training. In *ICLR*. OpenReview.net, 2021.
121. I. V. Ramakrishnan, R. Sekar, and A. Voronkov. Term indexing. In *Handbook of Automated Reasoning (in 2 volumes)*, pages 1853–1964. 2001.

122. M. Rawson and G. Reger. Directed graph networks for logical reasoning (extended abstract). In *PAAR 2020*, volume 2752 of *CEUR Workshop Proceedings*, pages 109–119. CEUR-WS.org, 2020.
123. M. Rawson and G. Reger. lazyCoP: Lazy paramodulation meets neurally guided search. In *TABLEAUX 2021*, volume 12842 of *LNCS*, pages 187–199. Springer, 2021.
124. G. Reger, M. Suda, and A. Voronkov. Playing with AVATAR. In *CADE-25*, volume 9195 of *LNCS*, pages 399–415. Springer, 2015.
125. J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM (JACM)*, 12(1):23–41, 1965.
126. J. A. Robinson and A. Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001.
127. T. Rocktäschel and S. Riedel. End-to-end differentiable proving. In *NeurIPS 2017*, pages 3788–3800, 2017.
128. J. Rute, M. Olsák, L. Blaauwbroek, F. I. S. Massolo, J. Piepenbrock, and V. Pestun. Graph2Tac: Learning hierarchical representations of math concepts in theorem proving. *CoRR*, abs/2401.02949, 2024.
129. A. Sanchez-Stern, Y. Alhessi, L. K. Saul, and S. Lerner. Generating correctness proofs with neural networks. *CoRR*, abs/1907.07794, 2019.
130. A. Sanchez-Stern, E. First, T. Zhou, Z. Kaufman, Y. Brun, and T. Ringer. Passport: Improving automated formal verification using identifiers. *ACM Trans. Program. Lang. Syst.*, 45(2), jun 2023.
131. F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Trans. on Neural Networks*, 20(1):61–80, 2009.
132. R. Schaeffer, B. Miranda, and S. Koyejo. Are emergent abilities of large language models a mirage? In A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 55565–55581. Curran Associates, Inc., 2023.
133. S. Schäfer and S. Schulz. Breeding theorem proving heuristics with genetic algorithms. In *GCAI*, volume 36 of *EPiC*, pages 263–274, 2015.
134. S. Schulz. Explanation based learning for distributed equational deduction. Diplomarbeit in Informatik, Fachbereich Informatik, Univ. Kaiserslautern, 1995.
135. S. Schulz. *Learning Search Control Knowledge for Equational Deduction*, volume 230 of *DISKI*. Infix Akademische Verlagsgesellschaft, 2000.
136. S. Schulz. Learning search control knowledge for equational theorem proving. In *Proc. of the Joint German/Austrian Conference on Artificial Intelligence (KI-2001)*, volume 2174 of *LNAI*, pages 320–334. Springer, 2001.
137. S. Schulz. E - a brainiac theorem prover. *AI Commun.*, 15(2-3):111–126, 2002.
138. S. Schulz, S. Cruanes, and P. Vukmirovic. Faster, higher, stronger: E 2.3. In *CADE 27*, pages 495–507, 2019.
139. S. Schulz and M. Möhrmann. Performance of clause selection heuristics for saturation-based theorem proving. In *IJCAR 2016*, volume 9706 of *LNCS*, pages 330–345, 2016.
140. J. Scott, A. Niemetz, M. Preiner, S. Nejati, and V. Ganesh. MachSMT: A machine learning-based algorithm selector for SMT solvers. In J. F. Groote and K. G. Larsen, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 303–325, Cham, 2021. Springer International Publishing.
141. D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, and D. L. Dill. Learning a SAT solver from single-bit supervision. In *ICLR 2019*. OpenReview.net, 2019.
142. J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA, 2004.

143. J. P. M. Silva and K. A. Sakallah. GRASP - a new search algorithm for satisfiability. In *ICCAD 1996*, pages 220–227. IEEE Computer Society / ACM, 1996.
144. M. Suda. Improving ENIGMA-style clause selection while learning from history. In *CADE 28*, volume 12699 of *LNCS*, pages 543–561. Springer, 2021.
145. M. Suda. Vampire with a brain is a good ITP hammer. In *FroCoS 2021*, volume 12941 of *LNCS*, pages 192–209. Springer, 2021.
146. R. S. Sutton and A. G. Barto. *Reinforcement Learning - An Introduction*. Adaptive computation and machine learning. MIT Press, 1998.
147. S. Topan, D. Rolnick, and X. Si. Techniques for symbol grounding with SATNet. In *NeurIPS 2021*, volume 34, pages 20733–20744. Curran Associates, Inc., 2021.
148. J. Urban. Experimenting with machine learning in automatic theorem proving. Master's thesis, Charles University, Prague, 1998. English summary at https://www.ciirc.cvut.cz/~urbanjo3/MScThesisPaper.pdf.
149. J. Urban. Translating Mizar for first order theorem provers. In *MKM*, volume 2594 of *LNCS*, pages 203–215. Springer, 2003.
150. J. Urban. MPTP - motivation, implementation, first experiments. *J. Autom. Reasoning*, 33(3-4):319–339, 2004.
151. J. Urban. MPTP 0.2: Design, implementation, and initial experiments. *J. Autom. Reasoning*, 37(1-2):21–43, 2006.
152. J. Urban. MaLARea: a metasystem for automated reasoning in large theories. In *ESARLT*, volume 257 of *CEUR*. CEUR-WS.org, 2007.
153. J. Urban. BliStr: The Blind Strategymaker. In *GCAI 2015*, volume 36 of *EPiC*, pages 312–319, 2015.
154. J. Urban. No one shall drive us from the semantic AI paradise of computer-understandable math and science! https://slideslive.com/38909911/no-one-shall-drive-us-from-the-semantic-ai-paradise-of-computerunderstandable-math-and-science, 2018. Keynote at the Artificial General Intelligence Conference (AGI'18).
155. J. Urban and J. Jakubův. First neural conjecturing datasets and experiments. In *CICM 2020*, volume 12236 of *LNCS*, pages 315–323. Springer, 2020.
156. J. Urban, G. Sutcliffe, P. Pudlák, and J. Vyskočil. MaLARea SG1 - machine learner for automated reasoning with semantic guidance. In *IJCAR*, pages 441–456, 2008.
157. J. Urban, J. Vyskočil, and P. Štepánek. MaLeCoP machine learning connection prover. In *TABLEAUX 2011*, volume 6793 of *LNCS*, pages 263–277, 2011.
158. R. Veroff. Using hints to increase the effectiveness of an automated reasoning program: Case studies. *J. Autom. Reasoning*, 16(3):223–239, 1996.
159. A. Voronkov. AVATAR: architecture for first-order theorem provers. In *CAV 2014*, volume 8559 of *LNCS*, pages 696–710. Springer, 2014.
160. M. Wang, Y. Tang, J. Wang, and J. Deng. Premise selection for theorem proving by deep graph embedding. In *NIPS'17*, page 2783–2793, Red Hook, NY, USA, 2017. Curran Associates Inc.
161. P. Wang, P. L. Donti, B. Wilder, and J. Z. Kolter. SATNet: Bridging deep learning and logical reasoning with a differentiable satisfiability solver. In *ICML 2019*, volume 97, pages 6545–6554. PMLR, 2019.
162. Q. Wang, C. Kaliszyk, and J. Urban. First experiments with neural translation of informal to formal mathematics. In *CICM'18*, volume 11006 of *LNCS*, pages 255–270.
163. K. Yang and J. Deng. Learning to prove theorems via interacting with proof assistants. In *ICML-36*, volume 97 of *PMLR*, pages 6984–6994, 09–15 Jun 2019.

164. K. Yang, A. Swope, A. Gu, R. Chalamala, P. Song, S. Yu, S. Godil, R. Prenger, and A. Anandkumar. LeanDojo: Theorem proving with retrieval-augmented language models. *arXiv preprint arXiv:2306.15626*, 2023.
165. K. Yang, A. M. Swope, A. Gu, R. Chalamala, P. Song, S. Yu, S. Godil, R. Prenger, and A. Anandkumar. LeanDojo: Theorem proving with retrieval-augmented language models. *CoRR*, abs/2306.15626, 2023.
166. L. Zhang, L. Blaauwbroek, C. Kaliszyk, and J. Urban. Learning proof transformations and its applications in interactive theorem proving. In *FroCoS 2023*, volume 14279 of *LNCS*, pages 236–254. Springer, 2023.
167. Z. Zombori, A. Csiszárik, H. Michalewski, C. Kaliszyk, and J. Urban. Towards finding longer proofs. In *TABLEAUX 2021*, volume 12842 of *LNCS*, pages 167–186, 2021.
168. Z. Zombori, J. Urban, and M. Olšák. The role of entropy in guiding a connection prover. In *TABLEAUX 2021*, volume 12842 of *LNCS*, pages 218–235. Springer, 2021.