UNIWERSYTET
WARSZAWSKI

Wydział Matematyki, Informatyki i Mechaniki
Uniwersytet Warszawski

Artificial Intelligence in Theorem Proving

(Sztuczna inteligencja w dowodzeniu)

Cezary Kaliszyk

MIMUW, March 2020

Overview

Last Lectures

- Premise selection
- Adaptations of standard machine learning algorithms
- Deep Math
- Unification

Today

- How do we characterize the prover / proof state?
- Syntactic and semantic features
- ML-evaluation

How to do meaningful characterization?

First: Characterization of what? We need to characterize either a proof state, a premise, a conjecture, or something similar. All of these essentially correspond to characterizing a mathematical formula (in the given logic)

We have initially used basic syntactic features (symbols that appear in a formula). We have seen that deep learning can do minimally better than that. But maybe with the knowledge of math we can characterize things more meaningfully

Lets first extend our features to all that we can extract syntactically.

Syntactic features used for learning

Symbols

constant, function, predicate, type-instance of a symbol

Types

type constants, type constructors, and type classes

Term as graph

subterms, paths in the term graph, these need to be considered with various variable normalizations

Meta Information

theory name, presence in various databases (e.g. used by the simplifier or as an induction rule)

Terms as graphs

Given the term
$f(a, g(b, c), h(d))$

Path features in this term would be

```
f g h  a b c d
f-a f-b f-c f-d f-g f-h g-b g-c h-d
f-g-b f-g-c f-h-d
```

Model features

In first order logic a formula is true if it is true in all models/interpretations.

Could we define features based on models? There are first-order logic tools to find counter models (so called counter model finders, e.g. Paradox)

To use such a counter model finder, we can take various formulas that appear in our dataset, and their subformulas (e.g. anticedents) and run them through the tools to get models. For example 1000 interesting models which spit a set of formulas nicely.

The validity of any new formula in these found 1000 models gives 1000 nice new features. These are very good, but these are only finite models, and they are restricted to the known symbols and the evaulation of these features is very slow.

Can we do better?

Requirements for Semantic Features

Express important semantic relations

Equivalence would be nice, but we need efficiency
Practical: Matching, Unification, ...

Use optimized ATP indexing trees

- discrimination trees
- substitution trees

High-level motivation

- internal nodes in such trees have useful semantic content (generalization)
- the trees only contain generalizations that are relevant
  (otherwise combinatorial explosion)

Substitution Trees

A substitution tree is a term indexing structure that stores terms in the form of a tree, where nodes are terms and edges are substitutions.

The tree starts with a variable as the root, and stored terms are obtained by subsequently applying substitutions going from the root to the leaf

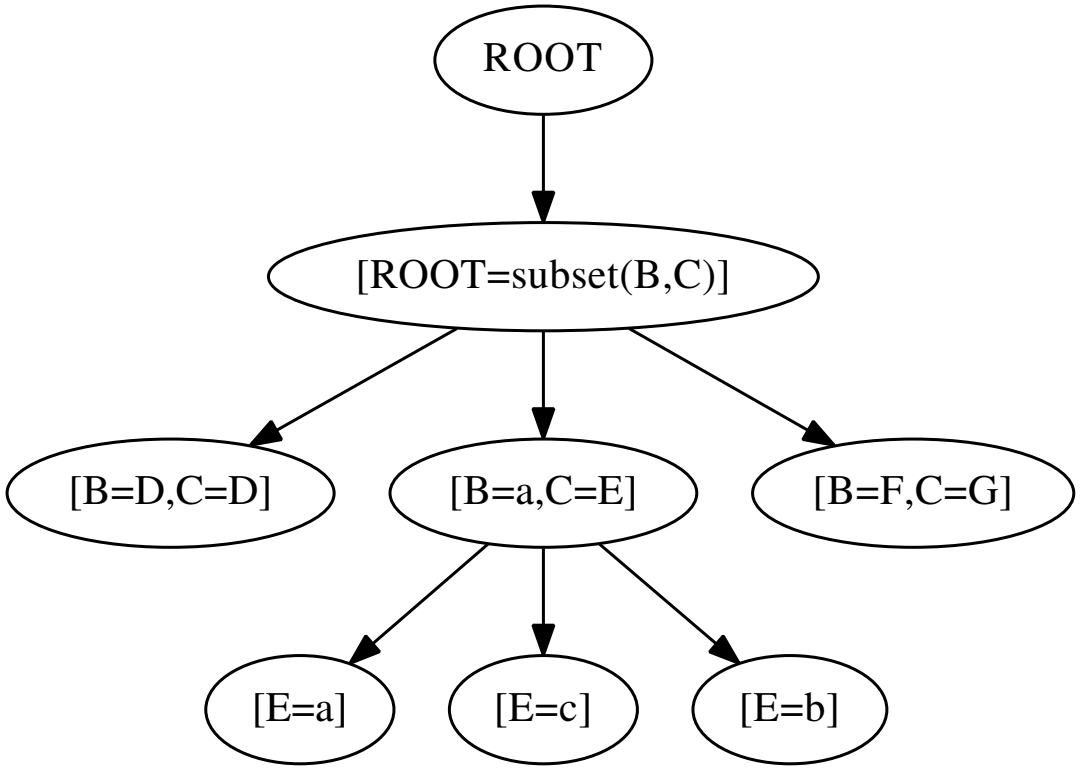
Finding terms in a substitution tree requires checking compatibility with the substitution and inserting terms may involve factoring a substitution

The further details are only given by example

Substitution Tree example

The tree that contains the terms on the left is as follows:

```
subset(A,B)
subset(a,b)
subset(a,c)
subset(C,C)
subset(a,a)
```



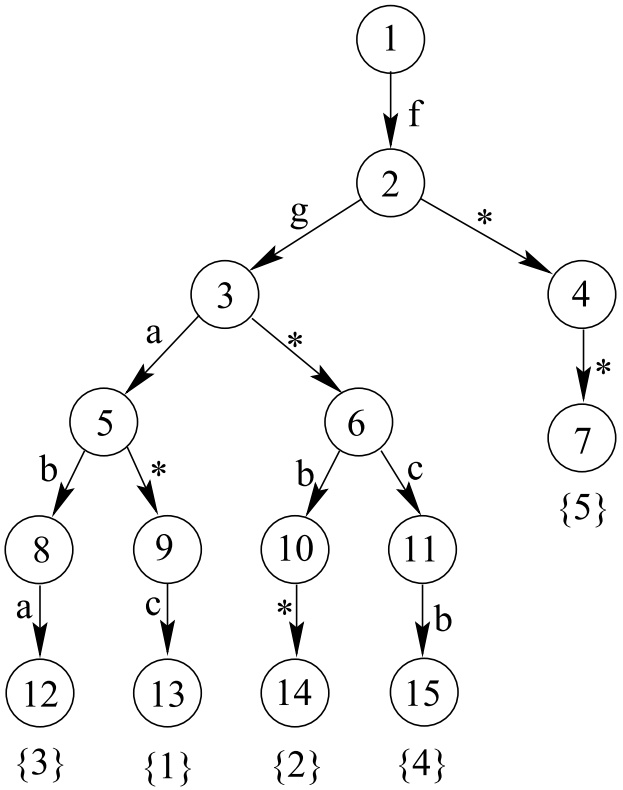
Discrimination Tree

A discrimination tree is an indexing structure that also stores terms and allows the recovery of terms that unify or match a given one

It is a variant of a trie, where the terms are input in the order of their reading. This means that a term $f(g(a, b), a)$ is represented as a path from the root $f - g - a - b - a$. Additionally variables are represented as $*$ and when looking up terms or inserting them subparts of terms need to be skipped.

Discrimination Tree example

For terms $f(g(a,*),c)$, $f(g(*,b),*)$, $f(g(a,b),a)$, $f(g(*,c),b)$, and $f(*,*)$



Exercise: How to find same / matching / unifying terms?

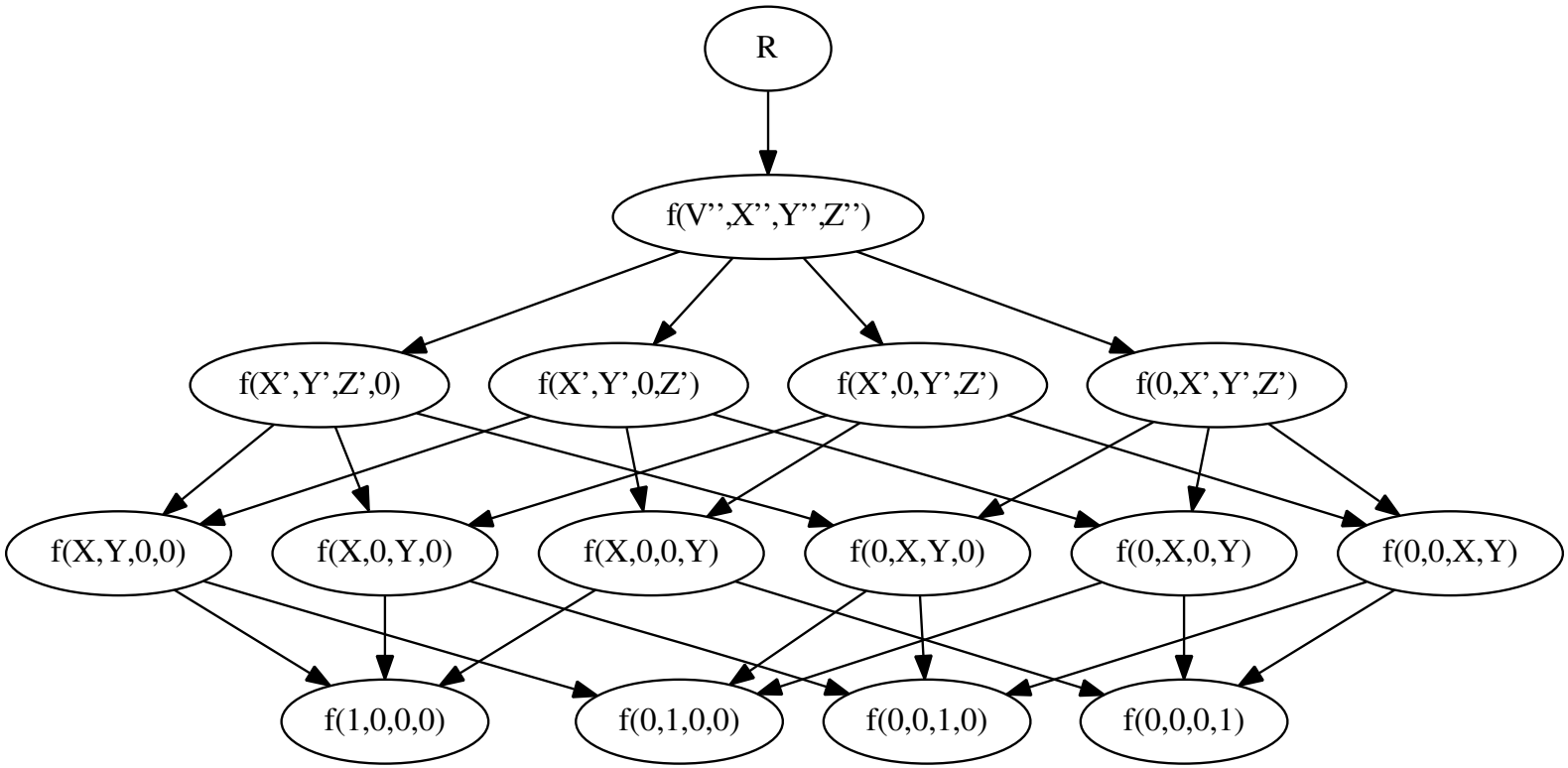
Discrimination Tree (2/2)

The way a discrimination tree is represented in HOL Light, where terms can additionally be lambda-terms is as follows.

```
type term_label = Vnet
                | Lcnet of (string * int)
                | Cnet of (int * int)
                | Lnet of int
type net = Netnode of net TermLabelMap.t
                * term option
```

Indexing trees do not guarantee all generalizations

- $1 + 2 = 2 + 1$ and $3 + 7 = 7 + 3$ are both generalized by $X + Y = Y + X$
- but the common generalization node is only guaranteed if $X + Y = Y + X$ is already in the library
- solution: heuristic generalization
- Note that full generalization explodes, as seen below:



Generalizations of f(a,g(b,c),h(d))

Generalizing the right argument (repeatedly) gives the following terms:

```
V a b c d h(V) h(d) g(V,V) g(b,V) g(b,c) f(V,V,V)
f(a,V,V) f(a,g(V,V),V) f(a,g(b,V),V) f(a,g(b,c),V)
f(a,g(b,c),h(V)) f(a,g(b,c),h(d))
```

Left:

```
V a b c d h(V) h(d) g(V,V) g(V,c) g(b,c) f(V,V,V)
f(V,V,h(V)) f(V,V,h(d)) f(V,g(V,V),h(d))
f(V,g(V,c),h(d)) f(V,g(b,c),h(d))
```

Generalization at every position gives:

```
V a b c d h(V) h(d) g(V,c) g(b,V) g(b,c)
f(V,g(b,c),h(d)) f(a,V,h(d)) f(a,g(V,c),h(d))
f(a,g(b,V),h(d)) f(a,g(b,c),V) f(a,g(b,c),h(V))
f(a,g(b,c),h(d))
```

Combinations of the above (but not all, not to get an expontential)

Summary of Features Used

Name	Description
SYM	Constant and function symbols
TRM_0	Subterms, all variables unified
TRM_α	Subterms, de Bruijn normalized
$\mathrm{MAT}_\varnothing$	Matching terms, no generalizations
MAT_r	Repeated gener. of rightmost innermost constant
MAT_l	Repeated gener. of leftmost innermost constant
MAT_1	Gener. of each application argument
MAT_2	Gener. of each application argument pair
$\mathrm{MAT}_\cup$	Union of all above generalizations
PAT	Walks in the term graph
ABS	Substitution tree nodes
UNI	All unifying terms

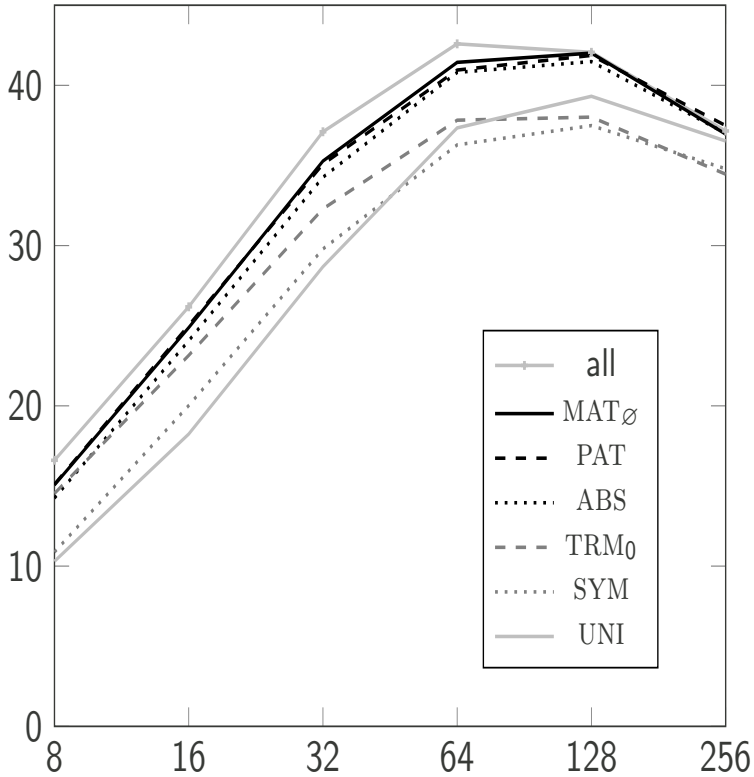
Feature Statistics

- Mizar based datasets: MPTP2078 and MML1147
- 4.5k and 150k formulas respectively

Method	Speed (sec)		Number of features		Learning and prediction (sec)	
	MPTP2078	MML1147	total	unique	knn	naive Bayes
SYM	0.25	10.52	30996	2603	0.96	11.80
TRM_α	0.11	12.04	42685	10633	0.96	24.55
TRM_0	0.13	13.31	35446	6621	1.01	16.70
$\text{MAT}_\varnothing$	0.71	38.45	57565	7334	1.49	24.06
MAT_r	1.09	71.21	78594	20455	1.51	39.01
MAT_l	1.22	113.19	75868	17592	1.50	37.47
MAT_1	1.16	98.32	82052	23635	1.55	41.13
MAT_2	5.32	4035.34	158936	80053	1.65	96.41
$\text{MAT}_\cup$	6.31	4062.83	180825	95178	1.71	112.66
PAT	0.34	64.65	118838	16226	2.19	52.56
ABS	11	10800	56691	6360	1.67	23.40
UNI	25	N/A	1543161	6462	21.33	516.24

ATP evaluation on MPTP2078 (E-prover / k-NN)

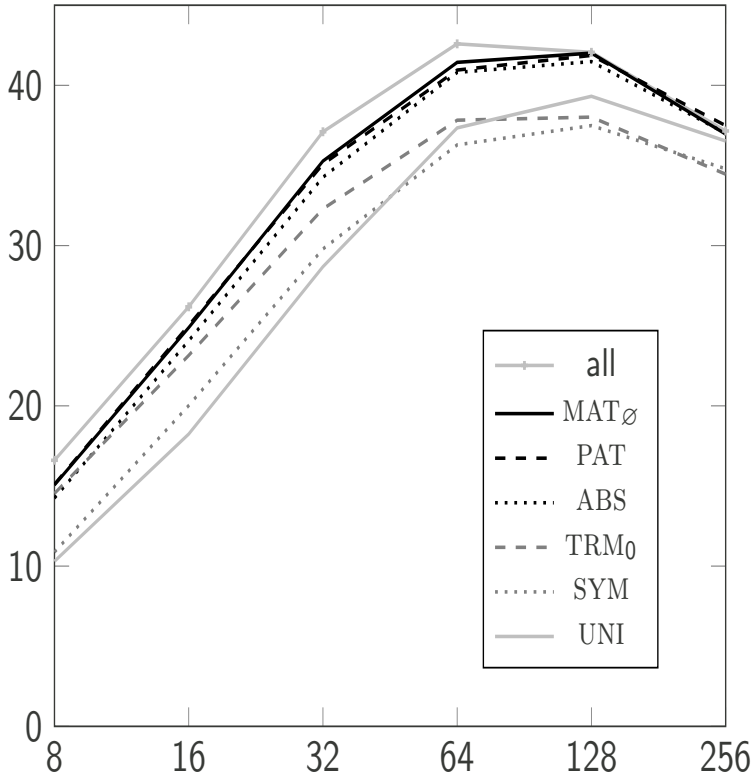
The left graph gives the ATP performance based on the number of predicted premises. The right table gives the union across all premise numbers



Method	Proved (%)	Theorems
$\text{MAT}_{\varnothing}$	54.379	1130
MAT_{r}	54.331	1129
MAT_{l}	54.283	1128
PAT	54.235	1127
$\text{MAT}_{\cup}$	53.994	1122
MAT_{1}	53.994	1122
MAT_{2}	53.898	1120
ABS	53.802	1118
TRM_{0}	50.529	1050
UNI	50.241	1044
SYM	48.027	998
TRM_{α}	43.888	912
$\text{SYM}\vert\text{TRM}_{0}\vert\text{MAT}_{\varnothing}\vert\text{ABS}$	55.486	1153

ATP evaluation on MPTP2078 (E-prover / k-NN)

The left graph gives the ATP performance based on the number of predicted premises. The right table gives the union across all premise numbers



Method	Proved (%)	Theorems
$\text{MAT}_{\varnothing}$	54.379	1130
MAT_{r}	54.331	1129
MAT_{l}	54.283	1128
PAT	54.235	1127
$\text{MAT}_{\cup}$	53.994	1122
MAT_{1}	53.994	1122
MAT_{2}	53.898	1120
ABS	53.802	1118
TRM_{0}	50.529	1050
UNI	50.241	1044
SYM	48.027	998
TRM_{α}	43.888	912
$\text{SYM}\vert\text{TRM}_{0}\vert\text{MAT}_{\varnothing}\vert\text{ABS}$	55.486	1153

Different features allow proving different theorems!

Improvement over syntax based features is much more than DeepMath

Machine Learning Evaluation

So far we have evaluated the various learning methods based on ATP evaluation

These are very hardware costly, the feature evaluation on the previous slide required a few CPU-months

Could we use some machine learning evaluation to do it faster?

Do the ML evaluation results actually correspond to the ATP ones?

Machine Learning Evaluation Metrics

- **Rank:** average position of a proof dependency in the sequence of suggestions ordered by their predicted relevance

Machine Learning Evaluation Metrics

- **Rank:** average position of a proof dependency in the sequence of suggestions ordered by their predicted relevance
- **100Cover:** average coverage of the set of dependencies by the first 100 suggestions (%)

Machine Learning Evaluation Metrics

- **Rank:** average position of a proof dependency in the sequence of suggestions ordered by their predicted relevance
- **100Cover:** average coverage of the set of dependencies by the first 100 suggestions (%)
- **100Precision:** ratio of the correctly predicted proof dependencies in the first 100 suggestions

Machine Learning Evaluation Metrics

- **Rank:** average position of a proof dependency in the sequence of suggestions ordered by their predicted relevance
- **100Cover:** average coverage of the set of dependencies by the first 100 suggestions (%)
- **100Precision:** ratio of the correctly predicted proof dependencies in the first 100 suggestions
- **Full Recall:** minimum number of predictions needed to include the whole set of proof dependencies

Machine Learning Evaluation Metrics

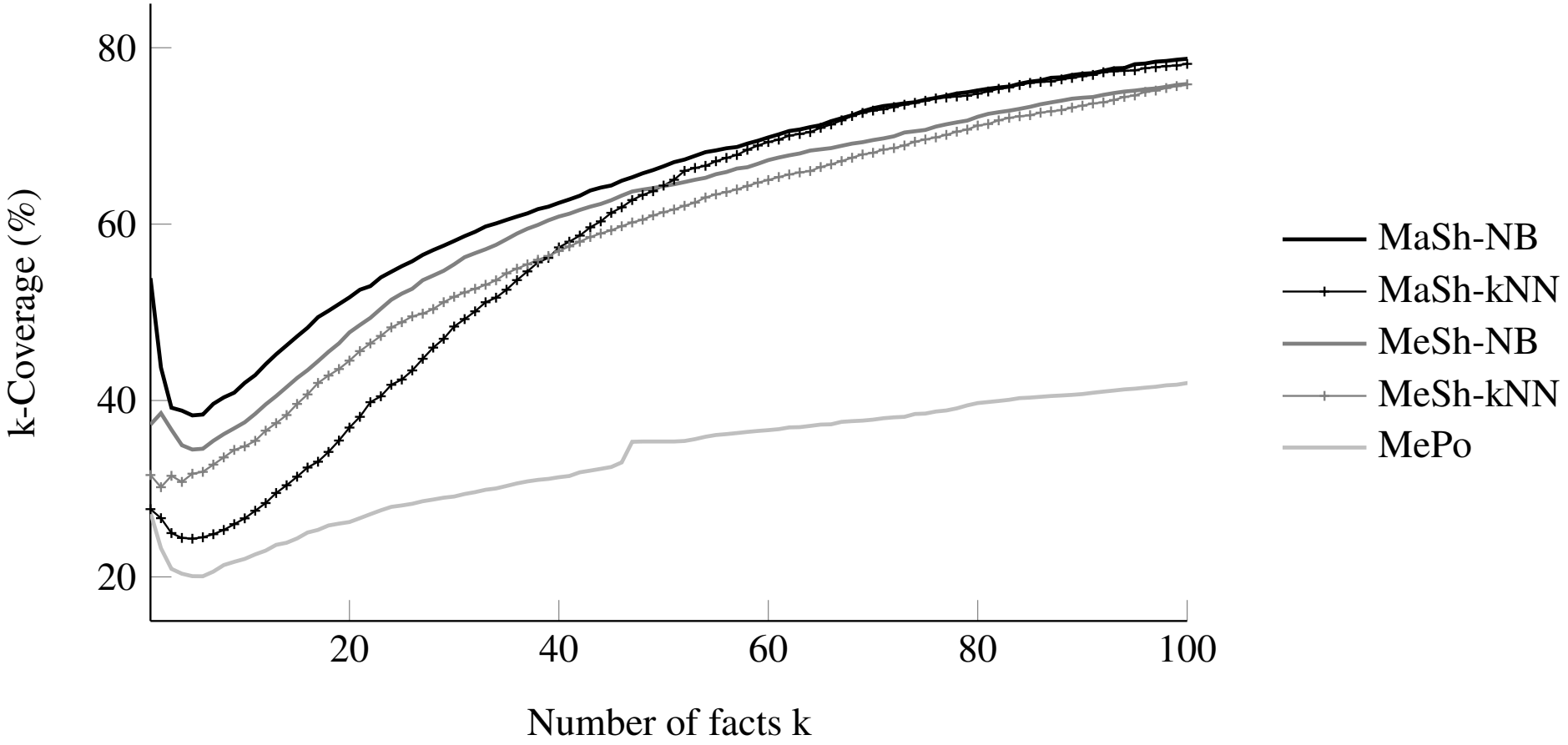
- **Rank:** average position of a proof dependency in the sequence of suggestions ordered by their predicted relevance
- **100Cover:** average coverage of the set of dependencies by the first 100 suggestions (%)
- **100Precision:** ratio of the correctly predicted proof dependencies in the first 100 suggestions
- **Full Recall:** minimum number of predictions needed to include the whole set of proof dependencies
- **AUC (Area under the ROC Curve):** probability that given a randomly drawn used premise and a randomly drawn unused premise, the used premise is ranked higher than the unused premise. Let $x_1, .., x_n$ be ranks of used premises and $y_1, .., y_m$ ranks unused ones.

$$\text{AUC} = \frac{\sum_i^n \sum_j^m 1_{x_i > y_j}}{mn}$$

where $1_{x_i > y_j} = 1$ iff $x_i > y_j$ and zero otherwise.

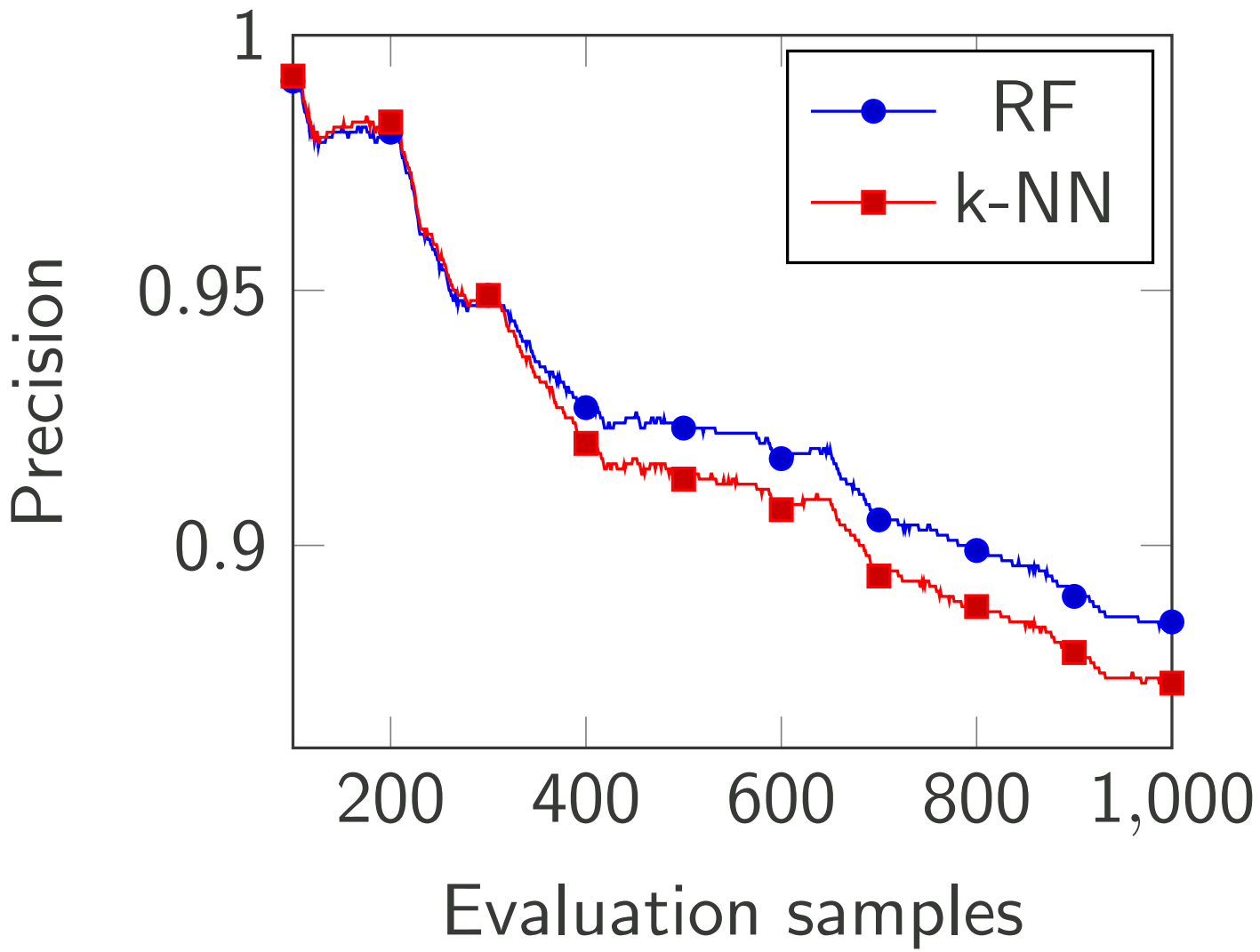
Coverage Example

We have discussed MePo in the second/third lecture, here is the ML evaluation for it:



Precision Example

We have discussed random forests in the second/third lecture, ML evaluation:



Additional Literature (not required)

More on various ATP indexing structures and their use for features

Peter Graf.
Term Indexing, volume 1053 of *Lecture Notes in Computer Science*. Springer, 1996.

Cezary Kaliszyk, Josef Urban, and Jirí Vyskocil.
Efficient semantic features for automated reasoning over large theories.
In Qiang Yang and Michael J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 3084–3090. AAAI Press, 2015.

Summary

This Lecture

- Unification-based features
- ML-evaluation

Next

- ATPs foundations
- deep learning in E prover