Progress in Automating Formalization

Josef Urban   Jiří Vyskočil

Czech Technical University in Prague

AITP 2017, Obergurgl
March 27, 2017

Two Obstacles to Strong Computer Support for Math

1. Low reasoning power of automated reasoning methods, particularly over large complex theories
2. Lack of computer understanding of current human-level (math and exact science) knowledge

- The two are related: human-level math may require nontrivial reasoning to become fully explained. Fully explained math gives us a lot of data for training AITP systems.
- And we want to train AITP on human-level proofs too. Thus getting interesting formalization/ATP/learning feedback loops.
- In 2014 we have decided that the AITP/hammer systems are getting strong enough to try this. And we started to combine them with statistical translation of informal-to-formal math.
- We are pretty cautious, but this really seems possible.

Favorable developments in the last decade

- Reasonably big formal corpora of common math are coming
- Reasonably strong proving methods over them are developed
- Large part of the latter was thanks to learning methods (40–50% of Mizar theorems automatically provable today)
- We are even getting some aligned informal/formal corpora:
- Flyspeck, Compendium of Continuous Lattices, Feit-Thompson
- So let's use what works:
- Statistical machine translation combined with strong learning-assisted automated reasoning over large libraries providing the common reasoning background!

Formal, Informal and Semiformal Corpora

- HOL Light and Flyspeck: some 25,000 theorems
- The Mizar Mathematical Library: some 60,000 theorems (most of them rather small lemmas), 10,000 definitions
- Coq: several large projects (Feit-Thompson theorem, ...)
- Isabelle, seL4 and the Archive of Formal Proofs
- Arxiv.org: 1M articles collected over some 20 years (not just math)
- Wikipedia: 25,000 articles in 2010 - collected over 10 years only
- Proofwiki - LaTeX but very semantic, re-invented the Mizar proof style

Experiments with Informalized Flyspeck

- 22000 Flyspeck theorem statements informalized
  - 72 overloaded instances like "+" for `vector_add`
  - 108 infix operators
  - forget all "prefixes"
    - `real_, int_, vector_, nadd_, hreal_, matrix_, complex_`
    - `ccos, cexp, clog, csin, ...`
    - `vsum, rpow, nsum, list_sum, ...`
  - Deleting all brackets, type annotations, and casting functors
    - `Cx` and `real_of_num` (which alone is used 17152 times).

Statistical Parsing of Informalized HOL

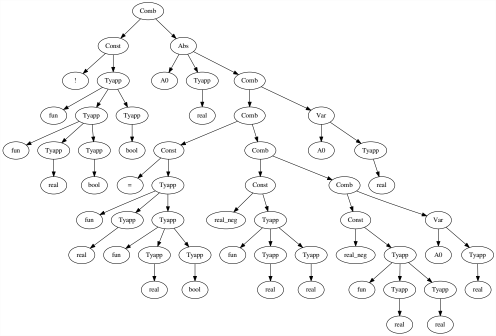
- Experiments with Stanford parser and CYK chart parser
- Examples (treebank) exported from Flyspeck formulas
  - Along with their informalized versions
- Grammar parse trees
  - Annotate each (nonterminal) symbol with its HOL type
  - Also “semantic (formal)” nonterminals annotate overloaded terminals
  - guiding analogy: word-sense disambiguation using CYK is common
- Terminals exactly compose the textual form, for example:
- REAL_NEGNEG: $\forall x. - -x = x$

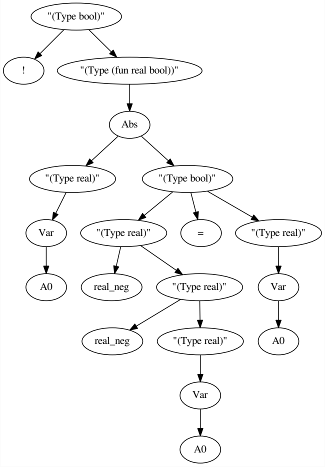
```
(Comb (Const "!" (Tyapp "fun" (Tyapp "fun" (Tyapp "real") (Tyapp "bool"))
(Tyapp "bool"))) (Abs "A0" (Tyapp "real") (Comb (Comb (Const "=" (Tyapp "fun"
(Tyapp "real") (Tyapp "fun" (Tyapp "real") (Tyapp "bool")))) (Comb (Const
"real_neg" (Tyapp "fun" (Tyapp "real") (Tyapp "real"))) (Comb (Const
"real_neg" (Tyapp "fun" (Tyapp "real") (Tyapp "real"))) (Var "A0" (Tyapp
"real"))))) (Var "A0" (Tyapp "real")))))
```

- becomes

```
("(Type bool)" ! ("(Type (fun real bool))" (Abs ("(Type real)"
(Var A0)) ("(Type bool)" ("(Type real)" real_neg ("(Type real)"
real_neg ("(Type real)" (Var A0)))) = ("(Type real)" (Var A0))))))
```

Example grammars



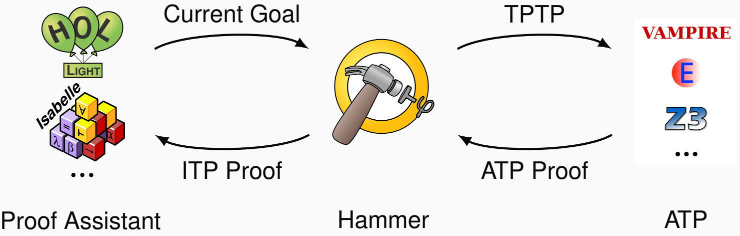


CYK Learning and Parsing

- Induce PCFG (probabilistic context-free grammar) from the trees
  - Grammar rules obtained from the inner nodes of each grammar tree
  - Probabilities are computed from the frequencies
- The PCFG grammar is binarized for efficiency
  - New nonterminals as shortcuts for multiple nonterminals
- CYK: dynamic-programming algorithm for parsing ambiguous sentences
  - input: sentence – a sequence of words and a binarized PCFG
  - output: N most probable parse trees
- Additional semantic pruning
  - Compatible types for free variables in subtrees
- Allow small probability for each symbol to be a variable
- Top parse trees are de-binarized to the original CFG
  - Transformed to HOL parse trees (preterms, Hindley-Milner)

Things that type-check are still not too good

Why not use today's AI/ATP ("hammers")?



Online parsing system

- "`sin ( 0 * x ) = cos pi / 2`"
- produces 16 parses
- of which 11 get type-checked by HOL Light as follows
- with all but three being proved by HOL(y)Hammer

```
sin (&0 * A0) = cos (pi / &2) where A0:real
sin (&0 * A0) = cos pi / &2 where A0:real
sin (&0 * &A0) = cos (pi / &2) where A0:num
sin (&0 * &A0) = cos pi / &2 where A0:num
sin (&(0 * A0)) = cos (pi / &2) where A0:num
sin (&(0 * A0)) = cos pi / &2 where A0:num
csin (Cx (&0 * A0)) = ccos (Cx (pi / &2)) where A0:real
csin (Cx (&0) * A0) = ccos (Cx (pi / &2)) where A0:real^2
Cx (sin (&0 * A0)) = ccos (Cx (pi / &2)) where A0:real
csin (Cx (&0 * A0)) = Cx (cos (pi / &2)) where A0:real
csin (Cx (&0) * A0) = Cx (cos (pi / &2)) where A0:real^2
```

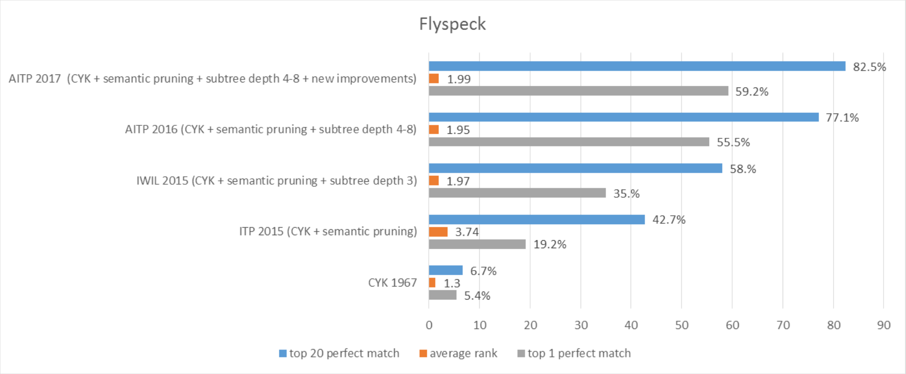
What we can correctly parse

```
! A0 ! A1 ! A2 ! A3 ! A4 FAN vec 0 , V_SY vecmats A4 ,
E_SY vecmats A4 /\ 1 < dimindex UNIV /\ 1 <= A0
/\ A0 <= dimindex UNIV /\ row A0 vecmats A4 = A3
/\ row SUC A0 MOD dimindex UNIV vecmats A4 = A1
/\ row SUC SUC A0 MOD dimindex
   UNIV MOD dimindex UNIV vecmats A4 = A2
/\ ! A5 ! A6 1 <= A5 /\ A5 <= dimindex UNIV /\ 1 <= A6
/\ A6 <= dimindex UNIV
/\ row A5 vecmats A4 = row A6 vecmats A4
==> A5 = A6 ==>
ivs_azim_cycle EE A1 E_SY vecmats A4 vec 0 A1 A2 = A3
```

Typechecking and proving over Flyspeck

- 698,549 of the parse trees typecheck (221,145 do not)
- 302,329 distinct (modulo alpha) HOL formulas
- For each HOL formula we try to prove it with a single AI-ATP method
- 70,957 (23%) can be automatically proved
  - A significant part of them are not interesting because of wrong parenthesizing
- In 39.4% of the 22,000 Flyspeck sentences the correct (training) HOL parse tree is among the best 20 parses
- its average rank: 9.34

Recent Progress on Flyspeck



Betting Slide from IHP'14, Paris

- In 25 years, 50% of the toplevel statements in LaTeX-written Msc-level math curriculum textbooks will be parsed automatically and with correct formal semantics
- Hurry up: I will only accept bets up to 10k EUR total (negotiable)
- More at `http://ai4reason.org/aichallenges.html`

Parsing Mizar – New Features

- More natural-language features than HOL (Andrzej was a linguist too)
- Arbitrary symbols, heavily overloaded
- Declarative natural-deduction style (re-invented in ProofWiki)
- Adjectives and their Prolog-style propagation (registrations)
- Dependent types
- Hidden arguments (derived from the context)
- Syntactic macros (synonyms, antonyms, expandable modes)
- This is all closer to LaTeX, but also a big challenge

Parsing Mizar – Phase0: Treebank Creation

- New transformation of the Mizar internal XML based on the HTML-izer
- The main trick: instead of hyperlinking, use the links as disambiguating nonterminals
- This is followed by using symbolic AI (ATP in our case) for mapping the syntax to the semantic layer
- Example: `RCOMP_1:5` in Mizar, Lisp, “semantic” TPTP and “syntactic” TPTP
- `for s, g being real number holds [.s,g.]  is closed`
- `(Bool "for" (Varlist (Set (Var "s")) "," (Varlist (Set (Var "g")))) "being" (Type ($#nv1_xreal_0 "real" ) ($#nm1_ordinal1 "number" ) ) "holds" (Bool (Set ($#nk1_rcomp_1 "[." ) (Set (Var "s")) "," (Set (Var "g")) ($#nk1_rcomp_1 ".]" ) ) "is" ($#nv2_rcomp_1 "closed" ) ))`
- `![A]: v1_xreal_0(A) => !  [B]: (v1_xreal_0(B) => v2_rcomp_1(k1_rcomp_1(A, B)))`
- `![A]: ![B]: ( ( nm1_ordinal1(A) & nv1_xreal_0(A) & nm1_ordinal1(B) & nv1_xreal_0(B) ) => nv2_rcomp_1(nk1_rcomp_1(A,B)) ) )).`

Examples of Mizar's Advanced Syntactic Mechanisms

```
definition
  let P,R be set;
  func P(#)R -> Relation means
  [x,y] in it iff ex z st [x,z] in P & [z,y] in R;
end;
notation synonym P*R for P(#)R; end;
definition
  let X,Y1,Y2,Z be set;
  let P be Relation of X,Y1;
  let R be Relation of Y2,Z;
  redefine func P*R -> Relation of X,Z;
end;
notation
  let f,g be Function;
  synonym g*f for f*g;
end;
```

Examples of How This Is Translated

```
::   synonym g*f for f*g;
fof(dt_nk3_funct_1, axiom,(![A,B]:(((v1_relat_1(A) &
v1_funct_1(A))  & (v1_relat_1(B) & v1_funct_1(B)))
=> nk3_funct_1(B, A)=nk6_relat_1(B,A)))).

:: synonym P*R for P(#)R;
fof(dt_nk6_relat_1, axiom,(![A,B]:((v1_relat_1(A)
& v1_relat_1(B))  => nk6_relat_1(A, B)=nk5_relat_1(A, B)))).
```

Parsing Mizar – Phase1: Statistical Parsing and Translation to Prolog/TPTP

- the most probable parses for an ambiguous Mizar-like sentence
- `for s, g being real number holds [.s,g.]  is closed`
- becomes
- `(Bool "for" (Varlist (Set (Var "s")) "," (Varlist (Set (Var "g")))) "being" (Type ($#nv1_xreal_0 "real" ) ($#nm1_ordinal1 "number" ) ) "holds" (Bool (Set ($#nk1_rcomp_1 "[." ) (Set (Var "s")) "," (Set (Var "g")) ($#nk1_rcomp_1 ".]" ) ) "is" ($#nv2_rcomp_1 "closed" ) ))`
- which is postprocessed (Lisp-to-TPTP) into the “syntactic TPTP”:
- `![A]: ![B]: ( ( nm1_ordinal1(A) & nv1_xreal_0(A) & nm1_ordinal1(B) & nv1_xreal_0(B) ) => nv2_rcomp_1(nk1_rcomp_1(A,B)) ) )).`

Parsing Mizar – Phase2: ATP connects the layers

- About 13000 Prolog-style formulas encoding the relation between user-level syntax and the semantic (MPTP) encoding
- Also the full set of Mizar typing rules needed for this!
- Altogether about 30000 background knowledge rules used for the mapping - pretty bad
- We try to prove that the syntactic form is implied by the semantic form
- Relatively non-trivial task for ATPs, requires premise selection and good ATP strategies
- Vampire: about 40% proved in 60s
- Targeted E strategies invented automatically on the corpus by our BliStrTune system: about 50% proved (by 14 strategies)

Parsing Mizar – Phase2: Example ATP problem

```
include('../stdincl').
fof(t5_rcomp_1,axiom,( ! [A] : ( v1_xreal_0(A) => ! [B] :
( v1_xreal_0(B) => v2_rcomp_1(k1_rcomp_1(A,B)) ) ) )).
fof(c_t5_rcomp_1,conjecture,( ! [A] : ! [B] : (
( nm1_ordinal1(A) & nv1_xreal_0(A) & nm1_ordinal1(B) &
nv1_xreal_0(B) ) => nv2_rcomp_1(nk1_rcomp_1(A,B)) ) )).
```

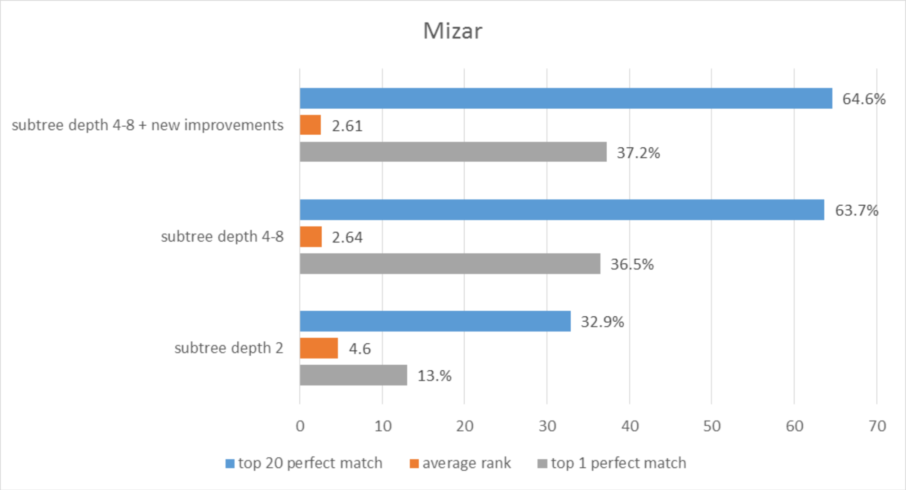
Further Improvements of the Parsing – Three Pass Algorithm

- bottom-up pass, we use CYK to compute only the set of reachable nonterminals for every cell in the parsing chart (instead of computing all partial parses)
- top-down pass we prune from the chart all nonterminals that cannot be reached from the top
- (bottom-up) parse we run the standard (full) CYK, however avoiding the unreachable nonterminals detected before
- => about 30% speedup on Mizar dataset

Further Improvements of the Parsing – Occam's Razor

- Occam's Razor to prefer simpler parses, where simpler means that the parse was constructed using fewer parsing rules
- this discourages e.g. from formulas that parse the very overloaded symbol + in many different ways
- $\frac{\texttt{probability of a standard partial parse}}{\texttt{num of all parsing rules of a partial parse}}$

First Mizar Results (100-fold Cross-validation)



Future Work

- Starting to look at full Mizar proofs and their alignment to ProofWiki
- Tighter integration of probabilistic parsing with semantic pruning (simple congruence closure already in)
- More corpora $\rightarrow$ more alignments $\rightarrow$ more knowledge $\rightarrow$ ...
- Smarter parsing methods
- Looping self-teaching systems:
- train on some data $\rightarrow$ parse $\rightarrow$ typecheck/prove the parses ...
- ... and thus get more data to train on $\rightarrow$ loop ...
- merge with other AI/ATP self-improving systems (MaLARea, concept alignment)

Thanks for listening!

- Questions?