Learning to Parse on Aligned Corpora

Cezary Kaliszyk Josef Urban Jiří Vyskočil

University of Innsbruck, Austria

Czech Technical University - CIIRC

Outline

Why (and why not) proof assistants

Science Fiction Proof Assistant
- Demo

Informal and Formal Mathematics
- Manual and Automatic Alignment

AI / ATP in parsing and proving

AI / ATP in parsing and proving

Why (and why not) proof assistants?

- Remarkable success
  - "...fully certified world..." [*Harrison06*] [*Leroy09,Asperti+12,Kumar+14*] [*Klein+14*]
    but who writes certified scripts?
  - "...impressive mathematics..." [*Gonthier07,Gonthier13,Hales+15*]
    we know them all
- Not for mathematicians [*Wiedijk07*]
  - "...nontrivial to learn..."
    syntax, foundations, tactics
  - "...work..."
    search, level of detail, automation

Why (and why not) proof assistants?

- Remarkable success
  - "...fully certified world..." [*Harrison06*] [*Leroy09,Asperti+12,Kumar+14*] [*Klein+14*]
    but who writes certified scripts?
  - "...impressive mathematics..." [*Gonthier07,Gonthier13,Hales+15*]
    we know them all
- Not for mathematicians [*Wiedijk07*]
  - "...nontrivial to learn..."
    syntax, foundations, tactics
  - "...work..."
    search, level of detail, automation
- But we have learned how to do this!
  - Can someone do this for me?
  - Can a computer do this for me?

QED+20 Workshop Discussion

- “...a proof assistant gets in the way, rather than helps...”
  - A spell-checker for LaTeX does not get in the way
  - A CAS does not get in the way
- Why does a proof assistant need to get in the way?
  - Syntax
    much worse than LaTeX
  - Knowledge
    a formal step relies on other steps being formal
  - Understanding
    what is obvious
  - Foundation issues
    is a type dependent? does this need reflection?

QED+20 Workshop Discussion

- "...a proof assistant gets in the way, rather than helps..."
  - A spell-checker for LaTeX does not get in the way
  - A CAS does not get in the way
- Why does a proof assistant need to get in the way?
  - Syntax
    much worse than LaTeX
  - Knowledge
    a formal step relies on other steps being formal
  - Understanding
    what is obvious
  - Foundation issues
    is a type dependent? does this need reflection?
- Why not allow LaTeX input for PAs?
  - Science Fiction?

Interaction:

- What you wrote
- What you wanted to write
- Is it actually true*

Interaction:

- What you wrote
- What you wanted to write
- Is it actually true*

Demo

Components of a “Science Fiction” Proof Assistant

- Understand LaTeX formulas, as well as some text
- Translate it to logic (of the proof assistant)
- Report on the success

Questions:

- Can we (a computer) learn formalization?
  - First: to state the lemmas formally? (this talk)
- Can we learn to prove?

Learn parsing on big corpora: which ones?

- Dense Sphere Packings: A Blueprint for Formal Proofs
  - 400 theorems and 200 concepts mapped [*Hales13*]
  - simple wiki
- IsaFoR [*SternagelThiemann14*]
  - most of “Term Rewriting and All That” [*BaderNipkow*]
- Compendium of Continuous Lattices (CCL)
  - 60% formalized in Mizar [*BancerekRudnicki02*]
  - high-level concepts and theorems aligned
- Feit-Thompson theorem by Gonthier [*Gonthier13*]
  - Two graduate books
- ProofWiki with detailed proofs and symbol linking
  - General topology corresponence with Mizar
  - Similar projects (PlanetMath, ...)

Aligned Formal and Informal Math - Flyspeck [*CICM13, ITP'13*]

Informal Formal

Definition of [fan, blade] DSKAGVP (fan) [fan ↔ FAN]

Let (V, E) be a pair consisting of a set $V \subset \mathbb{R}^3$ and a set E of unordered pairs of distinct elements of V. The pair is said to be a *fan* if the following properties hold.

1. (CARDINALITY) V is finite and nonempty. [cardinality ↔ fan1]
2. (ORIGIN) $\mathbf{0} \notin V$. [origin ↔ fan2]
3. (NONPARALLEL) If $\{\mathbf{v}, \mathbf{w}\} \in E$, then $\mathbf{v}$ and $\mathbf{w}$ are not parallel. [nonparallel ↔ fan6]
4. (INTERSECTION) For all $\varepsilon, \varepsilon' \in E \cup \{\{\mathbf{v}\} \ : \ \mathbf{v} \in V\}$, [intersection ↔ fan7]

$$C(\varepsilon) \cap C(\varepsilon') = C(\varepsilon \cap \varepsilon').$$

When $\varepsilon \in E$, call $C^0(\varepsilon)$ or $C(\varepsilon)$ a *blade* of the fan.

Informal Formal

```
#DSKAGVP?
let FAN=new_definition`FAN(x,V,E) <=> ((UNIONS E) SUBSET V) /\ graph(E) /\ fan1(x,V,E) /\ fan2(x,V,
fan6(x,V,E)/\ fan7(x,V,E)`;;
```

basic properties

The rest of the chapter develops the properties of fans. We begin with a completely trivial consequence of the definition.

Informal Formal

Lemma [] CTVTAQA (subset-fan)

If (V, E) is a fan, then for every $E' \subset E$, (V, E') is also a fan.

Proof

This proof is elementary.

Informal Formal

```
let CTVTAQA=prove(`!(x:real^3) (V:real^3->bool) (E:(real^3->bool)->bool) (E1:(real^3->bool)->bool).
FAN(x,V,E) /\ E1 SUBSET E
==>
FAN(x,V,E1)`,

REPEAT GEN_TAC
THEN REWRITE_TAC[FAN;fan1;fan2;fan6;fan7;graph]
THEN ASM_SET_TAC[]);;
```

Informal Formal

Lemma [fan cyclic] XOHLED

[$E(v) \leftrightarrow$ set_of_edge] Let (V, E) be a fan. For each $\mathbf{v} \in V$, the set

$$E(\mathbf{v}) = \{\mathbf{w} \in V \ : \ \{\mathbf{v}, \mathbf{w}\} \in E\}$$

is cyclic with respect to $(\mathbf{0}, \mathbf{v})$.

Proof

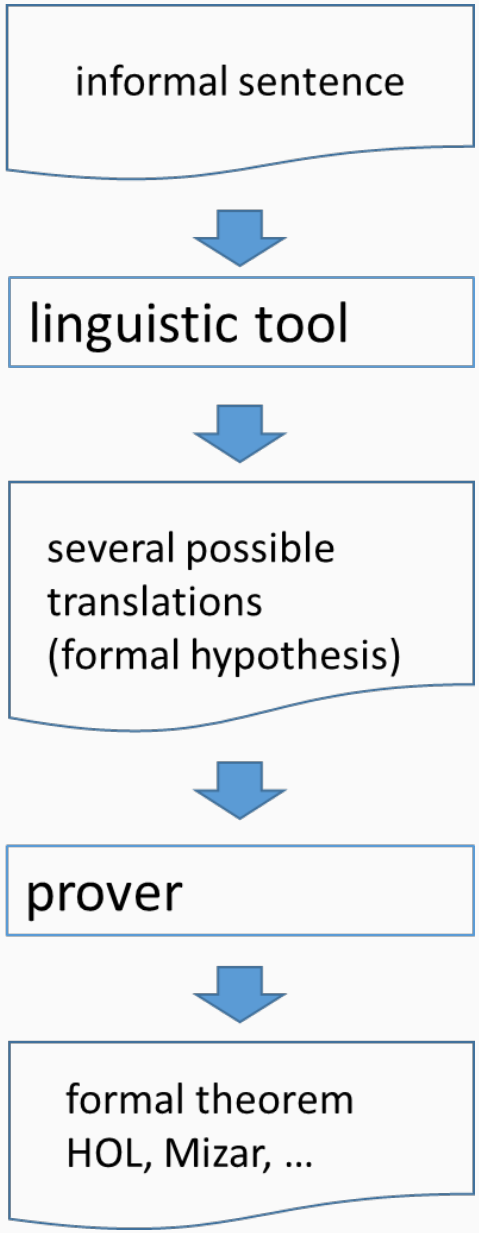
If $\mathbf{w} \in E(\mathbf{v})$, then $\mathbf{v}$ and $\mathbf{w}$ are not parallel. Also, if $\mathbf{w} \neq \mathbf{w}' \in E(\mathbf{v})$, then

Informal Formal

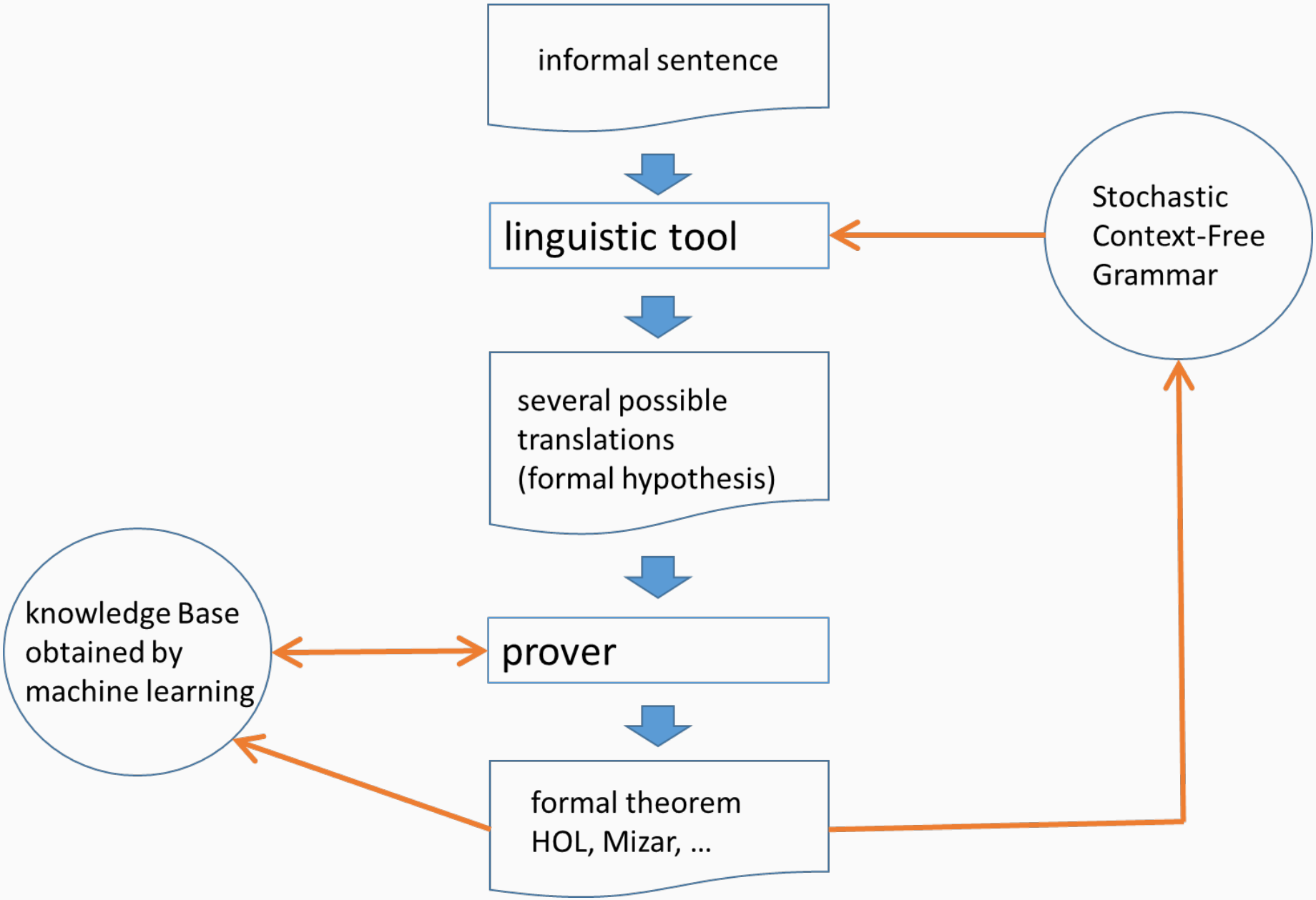
```
let XOHLED=prove(`!(x:real^3) (V:real^3->bool) (E:(real^3->bool)->bool) (v:real^3).
FAN(x,V,E) /\ v IN V
==> cyclic_set (set_of_edge v V E) x v`,

MESON_TAC[CYCLIC_SET_EDGE_FAN]);;
```

Parsing Scheme



Parsing Scheme with Learning



Statistical Parsing of Informalized HOL

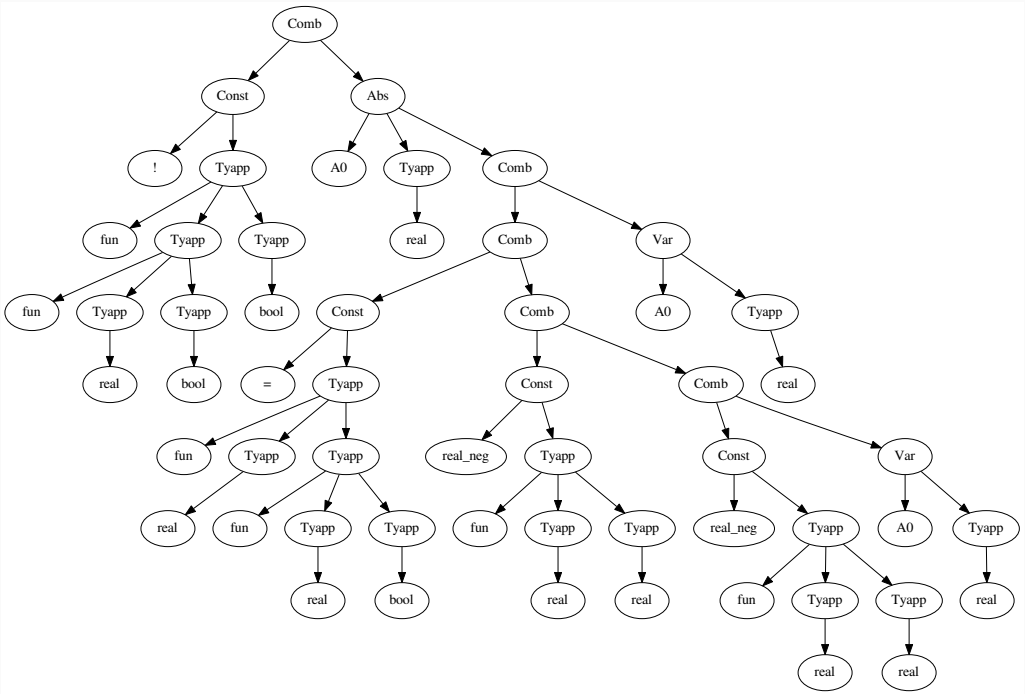
- Experiments with Standford parser and CYK chart parser
- Training and testing examples exported form Flyspeck formulas
  - Along with their informalized versions
- Grammar parse trees
  - Annotate each (nonterminal) symbol with its HOL type
  - Also “semantic (formal)” nonterminals annotate overloaded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

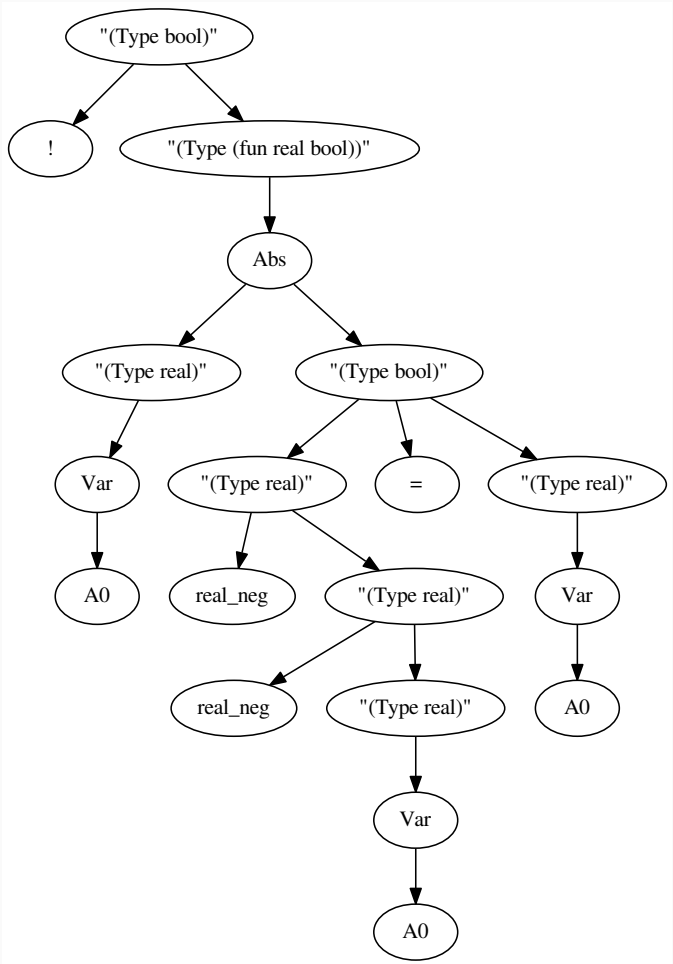
```
(Comb (Const "!" (Tyapp "fun" (Tyapp "fun" (Tyapp "real") (Tyapp "bool"))
(Tyapp "bool"))) (Abs "A0" (Tyapp "real") (Comb (Comb (Const "=" (Tyapp "fun"
(Tyapp "real") (Tyapp "fun" (Tyapp "real") (Tyapp "bool")))) (Comb (Const
"real_neg" (Tyapp "fun" (Tyapp "real") (Tyapp "real"))) (Comb (Const
"real_neg"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



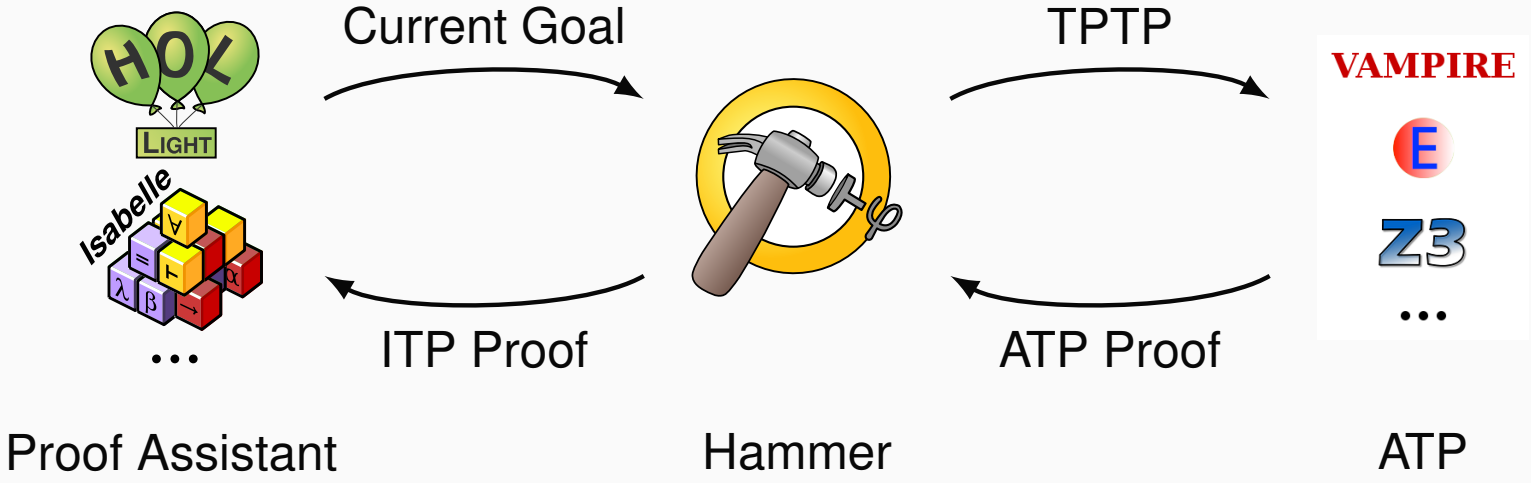


CYK Learning and Parsing

- 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 substructure/subtree preferences because CFG cannot handle many situations well because of its context free property
- 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")?



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).
- online parsing/proving demo system
- 100-fold cross-validation

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
```

Online parsing system

1. Extract n-grams from the target sentence
2. Use k-nearest neighbor to pre-select 1024 closest Flyspeck sentences
3. Train probabilistic grammar on their correct HOL parse trees
4. Use that grammar to get 16 best parses of the target sentence
5. Filter the 16 parse trees by typechecking in HOL
6. Try to prove them by 14 AI/ATP methods (using the whole Flyspeck)
7. Only the last phase is slow - with 200 CPUs it would be real-time too

100-fold cross-validation on the whole Flyspeck

- Split Flyspeck randomly into 100 chunks of 220 statements
- For each chunk C, build a probabilistic grammar on the union of remaining chunks (3–5 seconds)
- For each sentence in C get 20 best parse trees
- Takes about 4 seconds for each sentence – about 25 CPU hours in total

Typechecking and proving over Flyspeck (end of 2014)

- 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 parenthesation
- 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

Typechecking and proving over Flyspeck (now)

- ~~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 parenthesation
- In ~~39.4%~~60% of the 22,000 Flyspeck sentences the correct (training) HOL parse tree is among the best 20 parses
- its average rank: ~~9.34~~2.73

Conjecturing effect

- Many of the proved formulas are new and interesting
- 43 (0.2%) are same as an existing theorem, but a different one
- It seems that we have also produced a probabilistic conjecture-maker!
- All conjecture-makers we know about use exhaustive (non-probabilistic) generative methods
- Conjecture-making is a key problem-solving method in math
- State-of-theart ATPs don't have this ability yet – badly needed
- Evolve the system also towards wilder non-exhaustive conjecturing!

Future Work

- More corpora $\rightarrow$ more alignments $\rightarrow$ more knowledge $\rightarrow$ ...
- Smarter parsing methods
- Tighter integration of probabilistic parsing with semantic pruning
- 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, BliStr, ...)