Probabilistic Parsing of Mathematics

Jiří Vyskočil Josef Urban

Czech Technical University in Prague,

Czech Republic

Cezary Kaliszyk

University of Innsbruck,

Austria

Outline

- Why and why not current formal proof assistants
- Aligned corpora as a resource for learning to formalize
- Overview of parsing methods
- Problems with PCFG and the CYK algorithm
- Experiments with Informalized Flyspeck
- Parsing and Typechecking over Flyspeck
- Future Work

Why (and why not) proof assistants?

+ Remarkable success
+ “...fully certified world...”
    + Towards Self-verification of HOL Light [Harrison 2006]
    + A Formally Verified Compiler Back-end [Leroy 2009]
    + and some more…
+ “...impressive mathematics...”
    + The Four Colour Theorem: Engineering of a Formal Proof [Gonthier 2007]
    + Engineering mathematics: the odd order theorem proof [Gonthier 2013]
    + A formal proof of the Kepler conjecture [Hales+ 2015]
- “…not for mathematicians…” [Wiedijk 2007]
- “...nontrivial to learn...”
- syntax, foundations, tactics
- “...work...”
- search, level of detail, automation

Why (and why not) proof assistants?

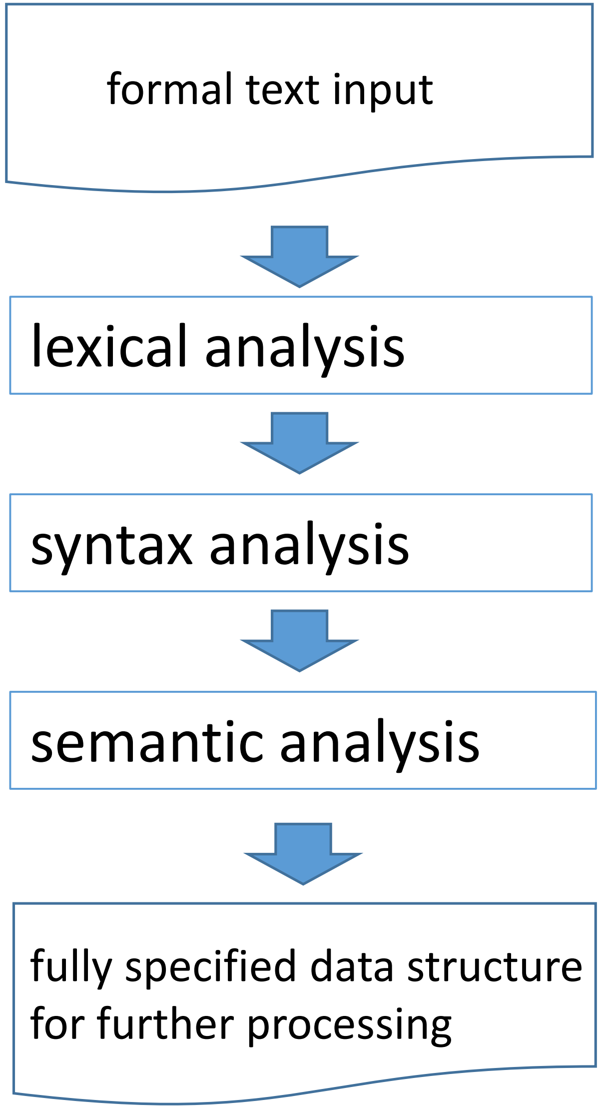
- But humans have learned how to do this “work”!
- Can someone do this for us?
- Can a computer do this for us?
- This is what we are trying in this project
- Try to automate the translation from informal to formal!
- In particular, try to learn such translation from aligned informal/formal corpora

Learn parsing on big corpora: which ones?

- Dense Sphere Packings: A Blueprint for Formal Proofs [Hales 2013]
  - 400 theorems and 200 concepts mapped
- IsaFoR [Sternagel, Thiemann 2014]
  - most of “Term Rewriting and All That” [Bader, Nipkow 1998]
- Compendium of Continuous Lattices (CCL) [Gierz at al. 1980]
  - 60% formalized in Mizar [Bancerek, Rudnicki 2002]
  - high-level concepts and theorems aligned
- Feit-Thompson theorem (two books)
  - formalized by Gonthier [Gonthier 2013] (two books)
- ProofWiki with detailed proofs and symbol linking
- General topology correspondence with Mizar
- Similar projects (PlanetMath, ...)

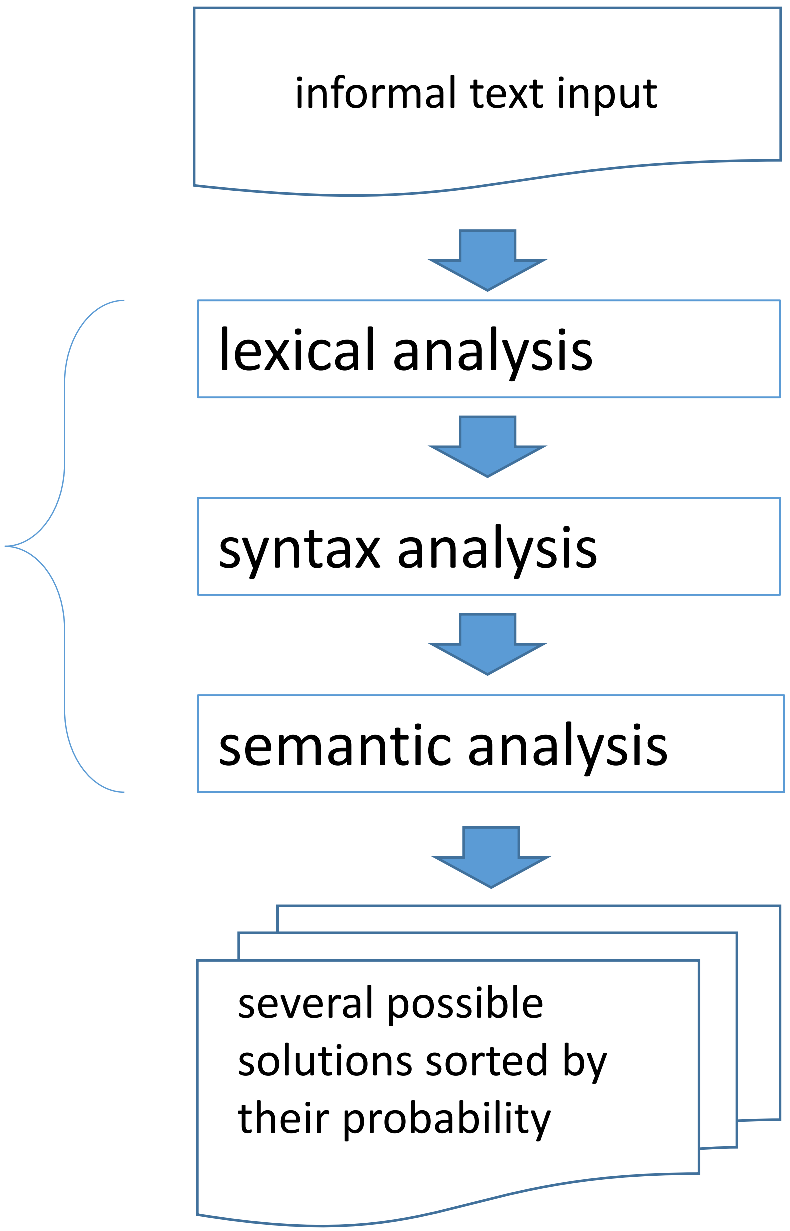
Traditional parsing approach:

- a language is designed manually in such a way that:
  - lexical tokens can be fully specified by some *regular language*
  - syntax analyzer can be fully specified by some *unambiguous context free grammar* (typically by deterministic CFG)
  - semantic analyzer typically resolves types of symbols and subtrees in a parsing tree, checks semantic correctness of binders, ….



Linguistic parsing approach:

- all of these phases (or at least some of them) can be learned (instead of encoding them manually) from examples by machine learning
- syntax (and mostly even semantic) analysis can be done by ambiguous CFG with probabilities (PCFG) and lexical analysis (in case of English) is often simple
- examples for learning have same (or similar) structure as parsing trees and they are called *treebanks* in this domain.
- rules and probabilities can be learned from treebanks
- CYK or Early parser can be used for parsing such PCFG



Comparison of
Traditional parsing X Linguistic parsing

Traditional parsing	Linguistic parsing
• have strong semantics	• does not have (or weak) semantics statistical methods are used instead
• it is fast due to deterministic algs	• It is relatively slow (cubic time)
• it can be hardly learn by machine	• can be learned by machine
• has only one correct solution	• has many possible solutions

CYK (CKY) algorithm for accepting sentence by CNF grammar

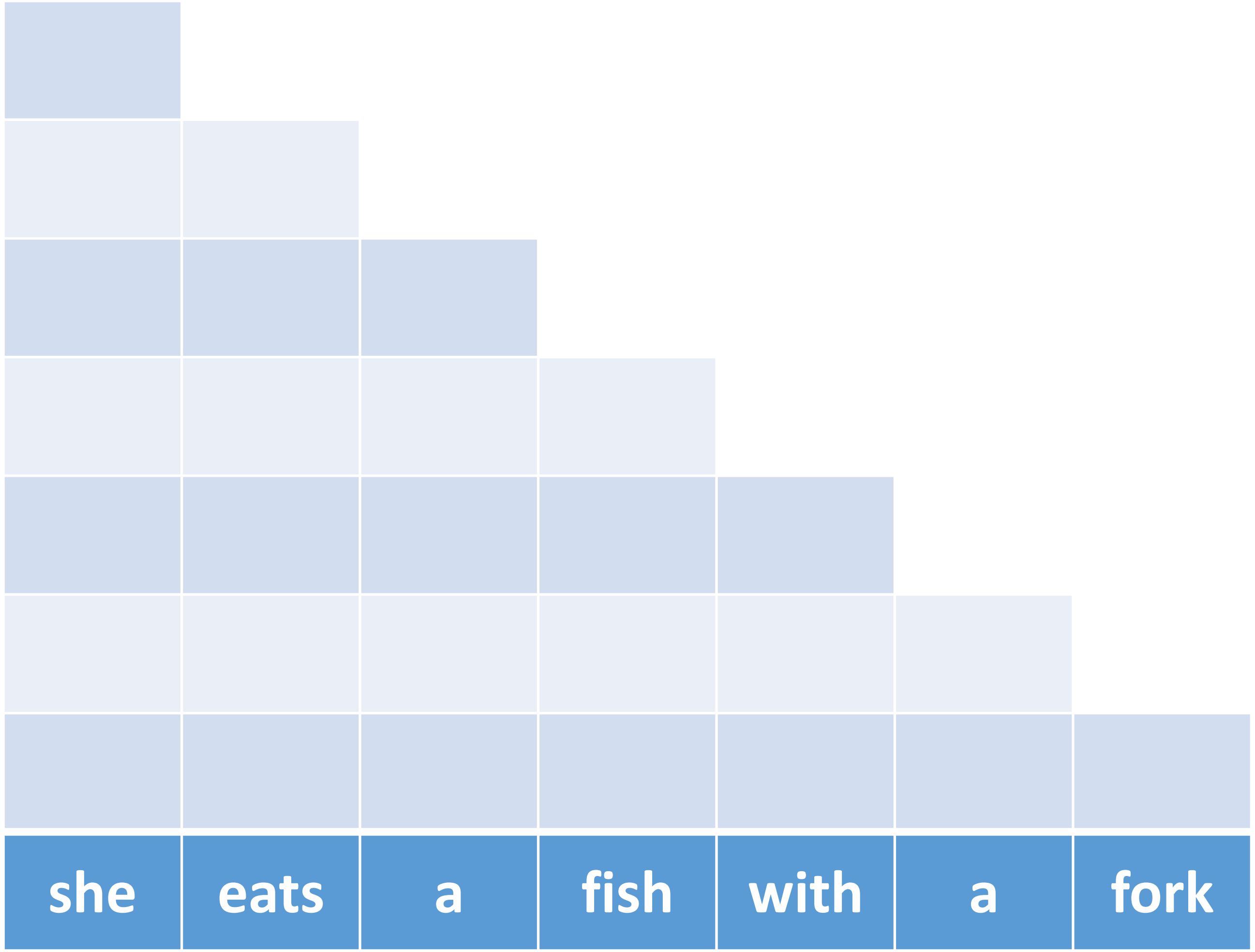
Example:

S -> NP VP
VP -> VP PP
VP -> V NP
VP -> eats
PP -> P NP
NP -> Det N
NP -> she
V -> eats
P -> with
N -> fish
N -> fork
Det -> a

CYK (CKY) algorithm for accepting sentence by CNF grammar

Example:

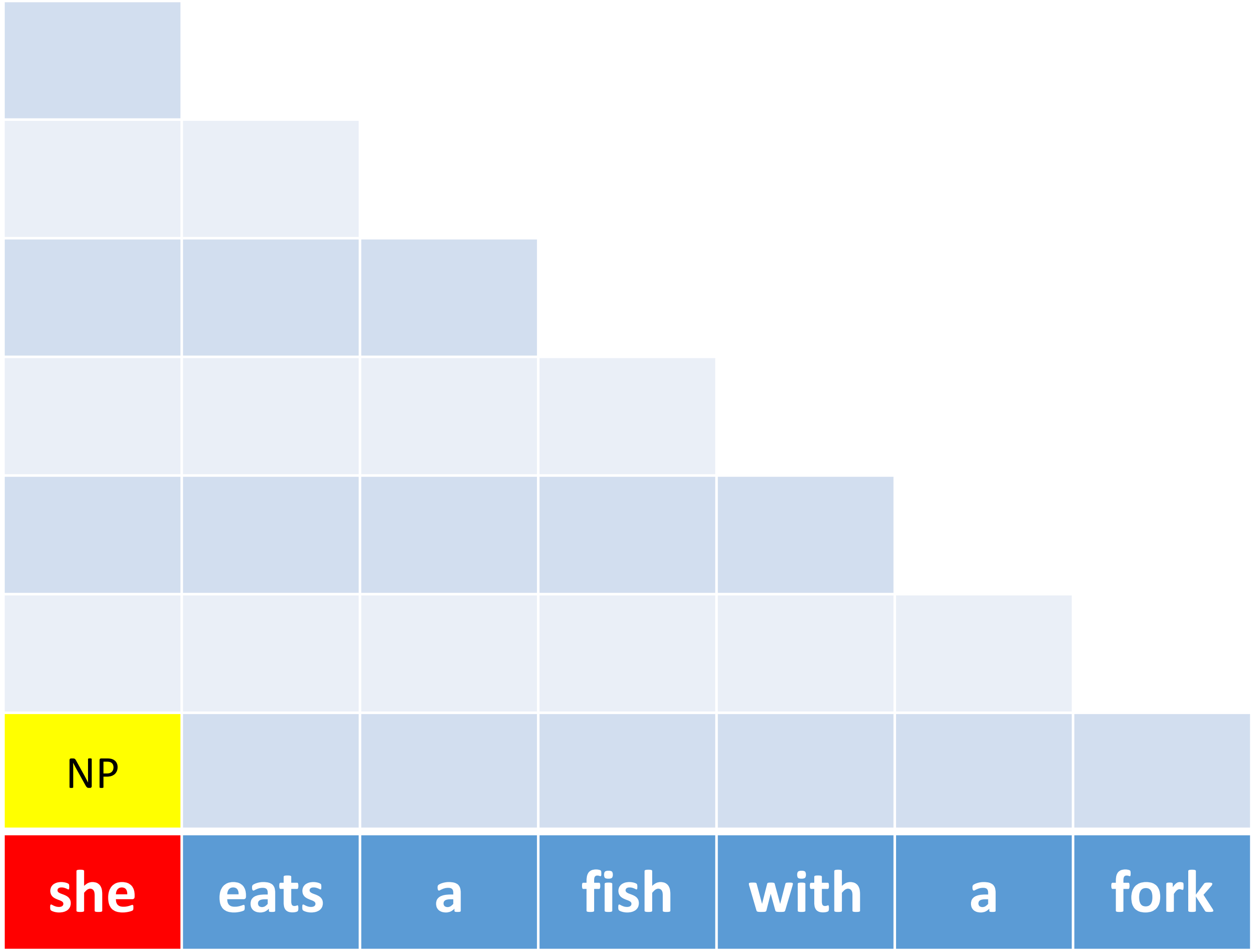
S -> NP VP
VP -> VP PP
VP -> V NP
VP -> eats
PP -> P NP
NP -> Det N
NP -> she
V -> eats
P -> with
N -> fish
N -> fork
Det -> a



CYK (CKY) algorithm for accepting sentence by CNF grammar

Example:

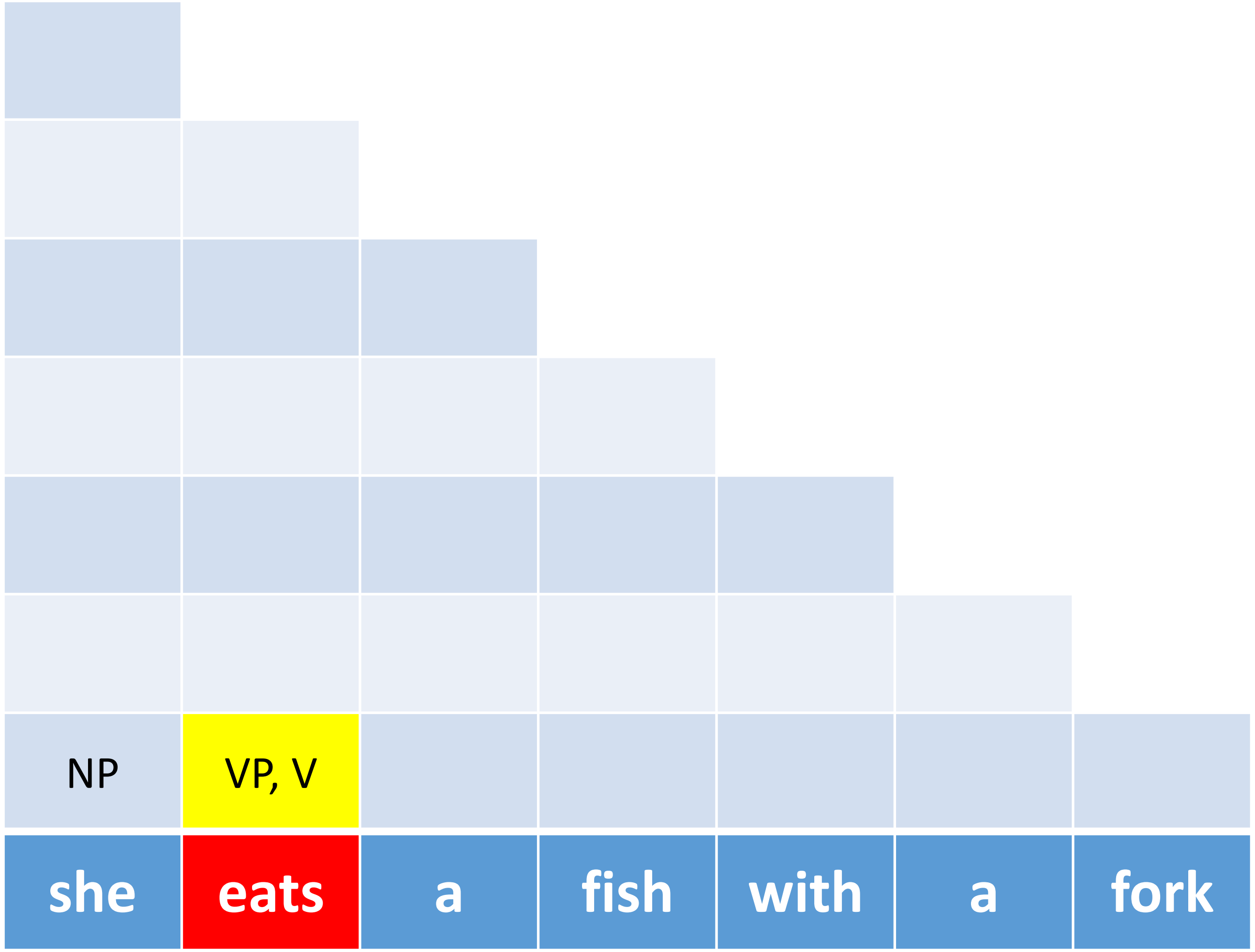
S -> NP VP
VP -> VP PP
VP -> V NP
VP -> eats
PP -> P NP
NP -> Det N
NP -> she
V -> eats
P -> with
N -> fish
N -> fork
Det -> a



CYK (CKY) algorithm for accepting sentence by CNF grammar

Example:

S -> NP VP
VP -> VP PP
VP -> V NP
VP -> eats
PP -> P NP
NP -> Det N
NP -> she
V -> eats
P -> with
N -> fish
N -> fork
Det -> a



CYK (CKY) algorithm for accepting sentence by CNF grammar

Example:

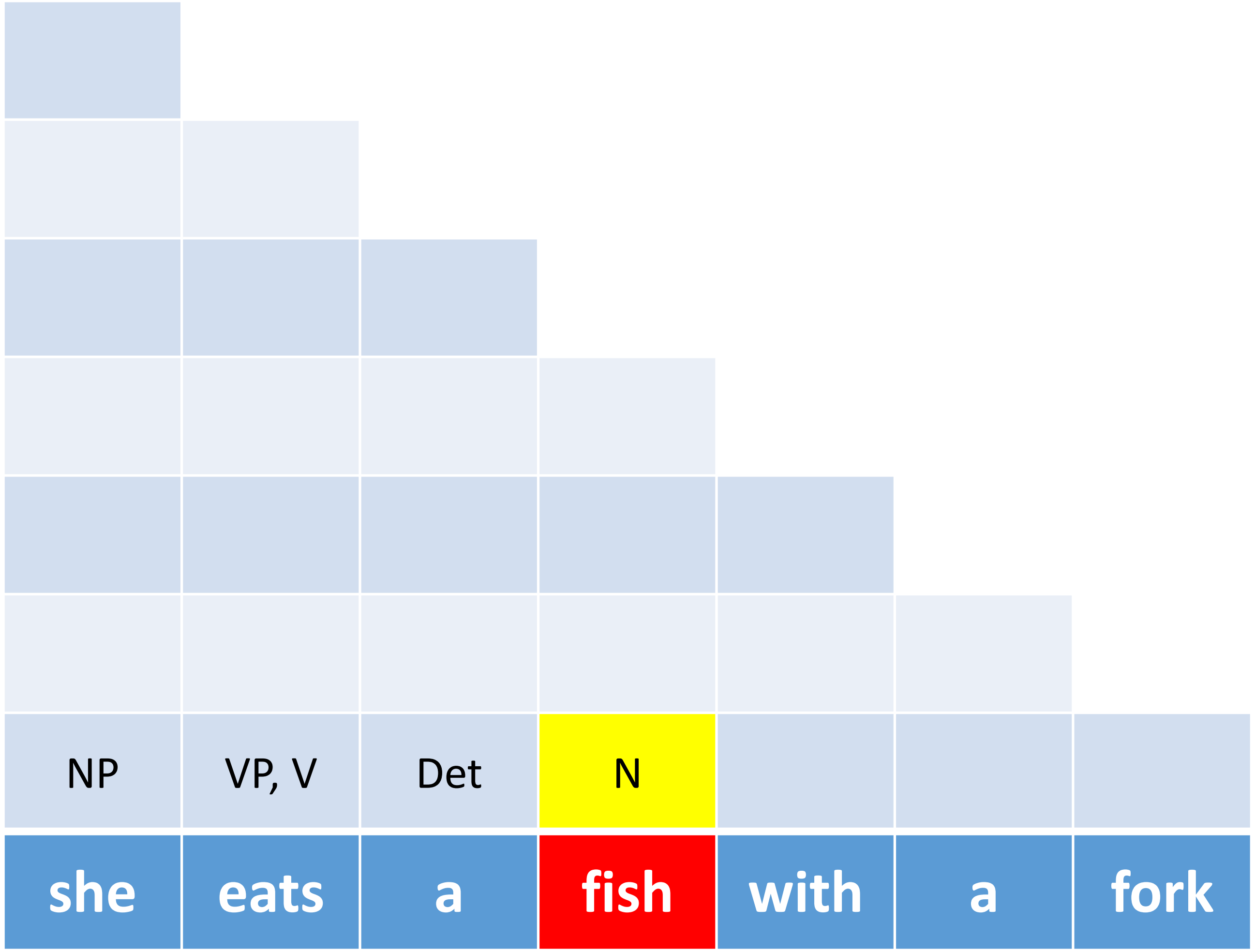
S -> NP VP
VP -> VP PP
VP -> V NP
VP -> eats
PP -> P NP
NP -> Det N
NP -> she
V -> eats
P -> with
N -> fish
N -> fork
Det -> a

NP	VP, V	Det				
she	**eats**	**a**	**fish**	**with**	**a**	**fork**

CYK (CKY) algorithm for accepting sentence by CNF grammar

Example:

S -> NP VP
VP -> VP PP
VP -> V NP
VP -> eats
PP -> P NP
NP -> Det N
NP -> she
V -> eats
P -> with
N -> fish
N -> fork
Det -> a



CYK (CKY) algorithm for accepting sentence by CNF grammar

Example:

S -> NP VP
VP -> VP PP
VP -> V NP
VP -> eats
PP -> P NP
NP -> Det N
NP -> she
V -> eats
P -> with
N -> fish
N -> fork
Det -> a

NP	VP, V	Det	N	P		
she	**eats**	**a**	**fish**	**with**	**a**	**fork**

CYK (CKY) algorithm for accepting sentence by CNF grammar

Example:

S -> NP VP
VP -> VP PP
VP -> V NP
VP -> eats
PP -> P NP
NP -> Det N
NP -> she
V -> eats
P -> with
N -> fish
N -> fork
Det -> a

NP	VP, V	Det	N	P	Det	
she	**eats**	**a**	**fish**	**with**	**a**	**fork**

CYK (CKY) algorithm for accepting sentence by CNF grammar

Example:

S -> NP VP
VP -> VP PP
VP -> V NP
VP -> eats
PP -> P NP
NP -> Det N
NP -> she
V -> eats
P -> with
N -> fish
N -> fork
Det -> a

NP	VP, V	Det	N	P	Det	N
she	**eats**	**a**	**fish**	**with**	**a**	**fork**

CYK (CKY) algorithm for accepting sentence by CNF grammar

Example:

S -> NP VP
VP -> VP PP
VP -> V NP
VP -> eats
PP -> P NP
NP -> Det N
NP -> she
V -> eats
P -> with
N -> fish
N -> fork
Det -> a

S						
NP	VP, V	Det	N	P	Det	N
she	**eats**	**a**	**fish**	**with**	**a**	**fork**

CYK (CKY) algorithm for accepting sentence by CNF grammar

Example:

S -> NP VP
VP -> VP PP
VP -> V NP
VP -> eats
PP -> P NP
NP -> Det N
NP -> she
V -> eats
P -> with
N -> fish
N -> fork
Det -> a

S						
NP	VP, V	Det	N	P	Det	N
she	**eats**	**a**	**fish**	**with**	**a**	**fork**

CYK (CKY) algorithm for accepting sentence by CNF grammar

Example:

S -> NP VP
VP -> VP PP
VP -> V NP
VP -> eats
PP -> P NP
NP -> Det N
NP -> she
V -> eats
P -> with
N -> fish
N -> fork
Det -> a

S		NP				
NP	VP, V	Det	N	P	Det	N
she	**eats**	**a**	**fish**	**with**	**a**	**fork**

CYK (CKY) algorithm for accepting sentence by CNF grammar

Example:

S -> NP VP
VP -> VP PP
VP -> V NP
VP -> eats
PP -> P NP
NP -> Det N
NP -> she
V -> eats
P -> with
N -> fish
N -> fork
Det -> a

S		NP				
NP	VP, V	Det	N	P	Det	N
she	**eats**	**a**	**fish**	**with**	**a**	**fork**

CYK (CKY) algorithm for accepting sentence by CNF grammar

Example:

S -> NP VP
VP -> VP PP
VP -> V NP
VP -> eats
PP -> P NP
NP -> Det N
NP -> she
V -> eats
P -> with
N -> fish
N -> fork
Det -> a

S		NP				
NP	VP, V	Det	N	P	Det	N
she	**eats**	**a**	**fish**	**with**	**a**	**fork**

CYK (CKY) algorithm for accepting sentence by CNF grammar

Example:

S -> NP VP
VP -> VP PP
VP -> V NP
VP -> eats
PP -> P NP
NP -> Det N
NP -> she
V -> eats
P -> with
N -> fish
N -> fork
Det -> a

S		NP			NP	
NP	VP, V	Det	N	P	Det	N
she	**eats**	**a**	**fish**	**with**	**a**	**fork**

CYK (CKY) algorithm for accepting sentence by CNF grammar

Example:

S -> NP VP
VP -> VP PP
VP -> V NP
VP -> eats
PP -> P NP
NP -> Det N
NP -> she
V -> eats
P -> with
N -> fish
N -> fork
Det -> a

S		NP			NP	
NP	VP, V	Det	N	P	Det	N
she	**eats**	**a**	**fish**	**with**	**a**	**fork**

CYK (CKY) algorithm for accepting sentence by CNF grammar

Example:

S -> NP VP
VP -> VP PP
VP -> V NP
VP -> eats
PP -> P NP
NP -> Det N
NP -> she
V -> eats
P -> with
N -> fish
N -> fork
Det -> a

S		NP			NP	
NP	VP, V	Det	N	P	Det	N
she	**eats**	**a**	**fish**	**with**	**a**	**fork**

CYK (CKY) algorithm for accepting sentence by CNF grammar

Example:

S -> NP VP
VP -> VP PP
VP -> V NP
VP -> eats
PP -> P NP
NP -> Det N
NP -> she
V -> eats
P -> with
N -> fish
N -> fork
Det -> a

	VP					
S		NP			NP	
NP	VP, V	Det	N	P	Det	N
she	**eats**	**a**	**fish**	**with**	**a**	**fork**

CYK (CKY) algorithm for accepting sentence by CNF grammar

Example:

S -> NP VP
VP -> VP PP
VP -> V NP
VP -> eats
PP -> P NP
NP -> Det N
NP -> she
V -> eats
P -> with
N -> fish
N -> fork
Det -> a

	VP					
S		NP			NP	
NP	VP, V	Det	N	P	Det	N
she	**eats**	**a**	**fish**	**with**	**a**	**fork**

CYK (CKY) algorithm for accepting sentence by CNF grammar

Example:

S -> NP VP
VP -> VP PP
VP -> V NP
VP -> eats
PP -> P NP
NP -> Det N
NP -> she
V -> eats
P -> with
N -> fish
N -> fork
Det -> a

	VP					
S		NP			NP	
NP	VP, V	Det	N	P	Det	N
she	**eats**	**a**	**fish**	**with**	**a**	**fork**

CYK (CKY) algorithm for accepting sentence by CNF grammar

Example:

S -> NP VP
VP -> VP PP
VP -> V NP
VP -> eats
PP -> P NP
NP -> Det N
NP -> she
V -> eats
P -> with
N -> fish
N -> fork
Det -> a

	VP					
S		NP			NP	
NP	VP, V	Det	N	P	Det	N
she	**eats**	**a**	**fish**	**with**	**a**	**fork**

CYK (CKY) algorithm for accepting sentence by CNF grammar

Example:

S -> NP VP
VP -> VP PP
VP -> V NP
VP -> eats
PP -> P NP
NP -> Det N
NP -> she
V -> eats
P -> with
N -> fish
N -> fork
Det -> a

	VP			PP		
S		NP			NP	
NP	VP, V	Det	N	P	Det	N
she	**eats**	**a**	**fish**	**with**	**a**	**fork**

CYK (CKY) algorithm for accepting sentence by CNF grammar

Example:

S -> NP VP
VP -> VP PP
VP -> V NP
VP -> eats
PP -> P NP
NP -> Det N
NP -> she
V -> eats
P -> with
N -> fish
N -> fork
Det -> a

	VP			PP		
S		NP			NP	
NP	VP, V	Det	N	P	Det	N
she	**eats**	**a**	**fish**	**with**	**a**	**fork**

CYK (CKY) algorithm for accepting sentence by CNF grammar

Example:

S -> NP VP
VP -> VP PP
VP -> V NP
VP -> eats
PP -> P NP
NP -> Det N
NP -> she
V -> eats
P -> with
N -> fish
N -> fork
Det -> a

S						
	VP			PP		
S		NP			NP	
NP	VP, V	Det	N	P	Det	N
she	**eats**	**a**	**fish**	**with**	**a**	**fork**

CYK (CKY) algorithm for accepting sentence by CNF grammar

Example:

S -> NP VP
VP -> VP PP
VP -> V NP
VP -> eats
PP -> P NP
NP -> Det N
NP -> she
V -> eats
P -> with
N -> fish
N -> fork
Det -> a

S						
	VP			PP		
S		NP			NP	
NP	VP, V	Det	N	P	Det	N
she	**eats**	**a**	**fish**	**with**	**a**	**fork**

CYK (CKY) algorithm for accepting sentence by CNF grammar

Example:

S -> NP VP
VP -> VP PP
VP -> V NP
VP -> eats
PP -> P NP
NP -> Det N
NP -> she
V -> eats
P -> with
N -> fish
N -> fork
Det -> a

S						
	VP			PP		
S		NP			NP	
NP	VP, V	Det	N	P	Det	N
she	**eats**	**a**	**fish**	**with**	**a**	**fork**

CYK (CKY) algorithm for accepting sentence by CNF grammar

Example:

S -> NP VP
VP -> VP PP
VP -> V NP
VP -> eats
PP -> P NP
NP -> Det N
NP -> she
V -> eats
P -> with
N -> fish
N -> fork
Det -> a

	VP					
S						
	VP			PP		
S		NP			NP	
NP	VP, V	Det	N	P	Det	N
she	**eats**	**a**	**fish**	**with**	**a**	**fork**

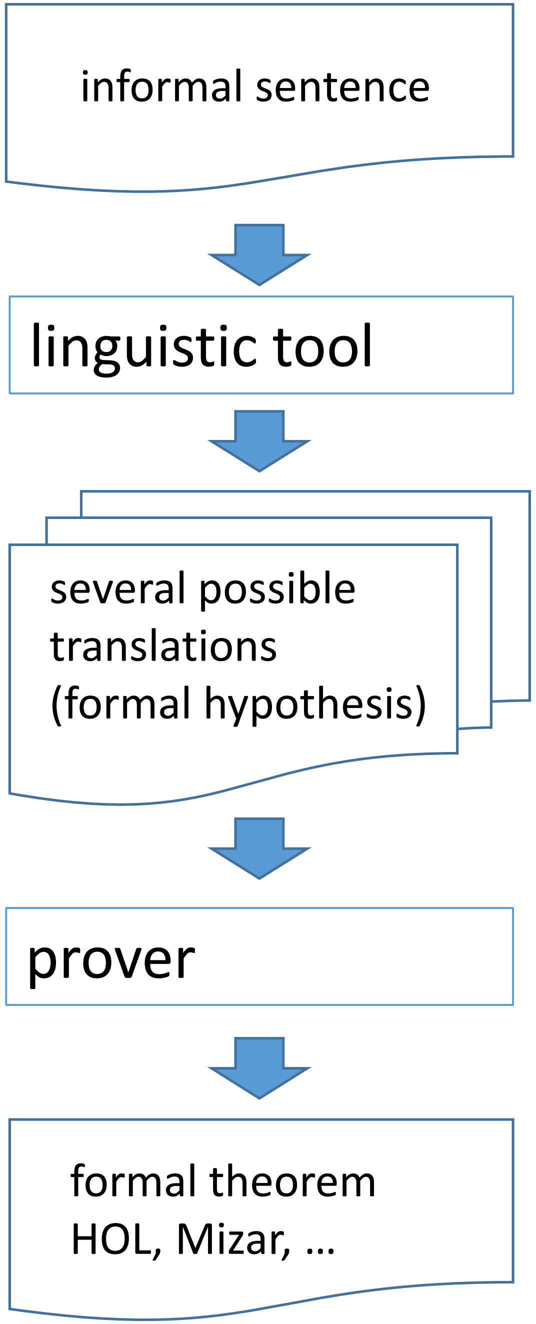
CYK (CKY) algorithm for accepting sentence by CNF grammar

Example:

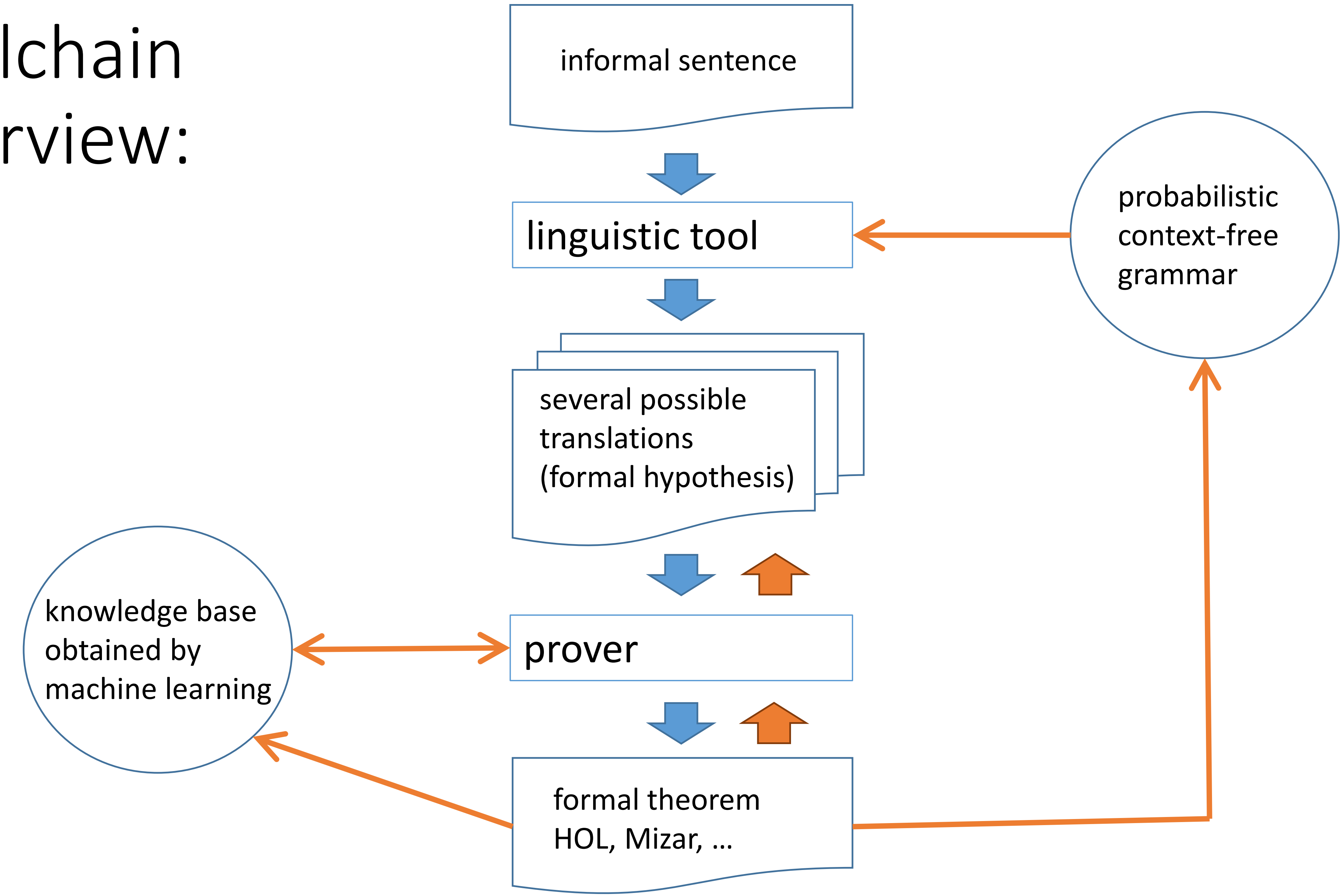
S -> NP VP
VP -> VP PP
VP -> V NP
VP -> eats
PP -> P NP
NP -> Det N
NP -> she
V -> eats
P -> with
N -> fish
N -> fork
Det -> a

S						
	VP					
S						
	VP			PP		
S		NP			NP	
NP	VP, V	Det	N	P	Det	N
she	**eats**	**a**	**fish**	**with**	**a**	**fork**

Toolchain overview:



Toolchain overview:



Experiments with Informalized Flyspeck

- Instead of using “real” informal mathematical text we obtain training parse trees from informalized theorem statements of Flyspeck project.
- 22000 Flyspeck theorem statements informalized:
  - 72 overloaded instances like “+” for `vector_add`
  - 108 infix operators
  - all “prefixes” are forgotten
    - `real_, int_, vector_, nadd_, hreal_, matrix_, complex_`
    - `ccos, cexp, clog, csin, ...`
    - `vsum, rpow, nsum, list_sum, ...`
  - all brackets, type annotations, and casting functors are deleted
    - `Cx` and `real_of_num` (which alone is used 17152 times)
  - online parsing/proving demo system:

http://colo12-c703.uibk.ac.at/hh/parse.html

Statistical Parsing of Informalized HOL

1) Training and testing examples are exported form Flyspeck formulas

Example:

REAL_NEGNEG: !x . -- -- x = x

Statistical Parsing of Informalized HOL

1) Training and testing examples are exported form Flyspeck formulae

Example:

REAL_NEGNEG: !x . -- -- x = x

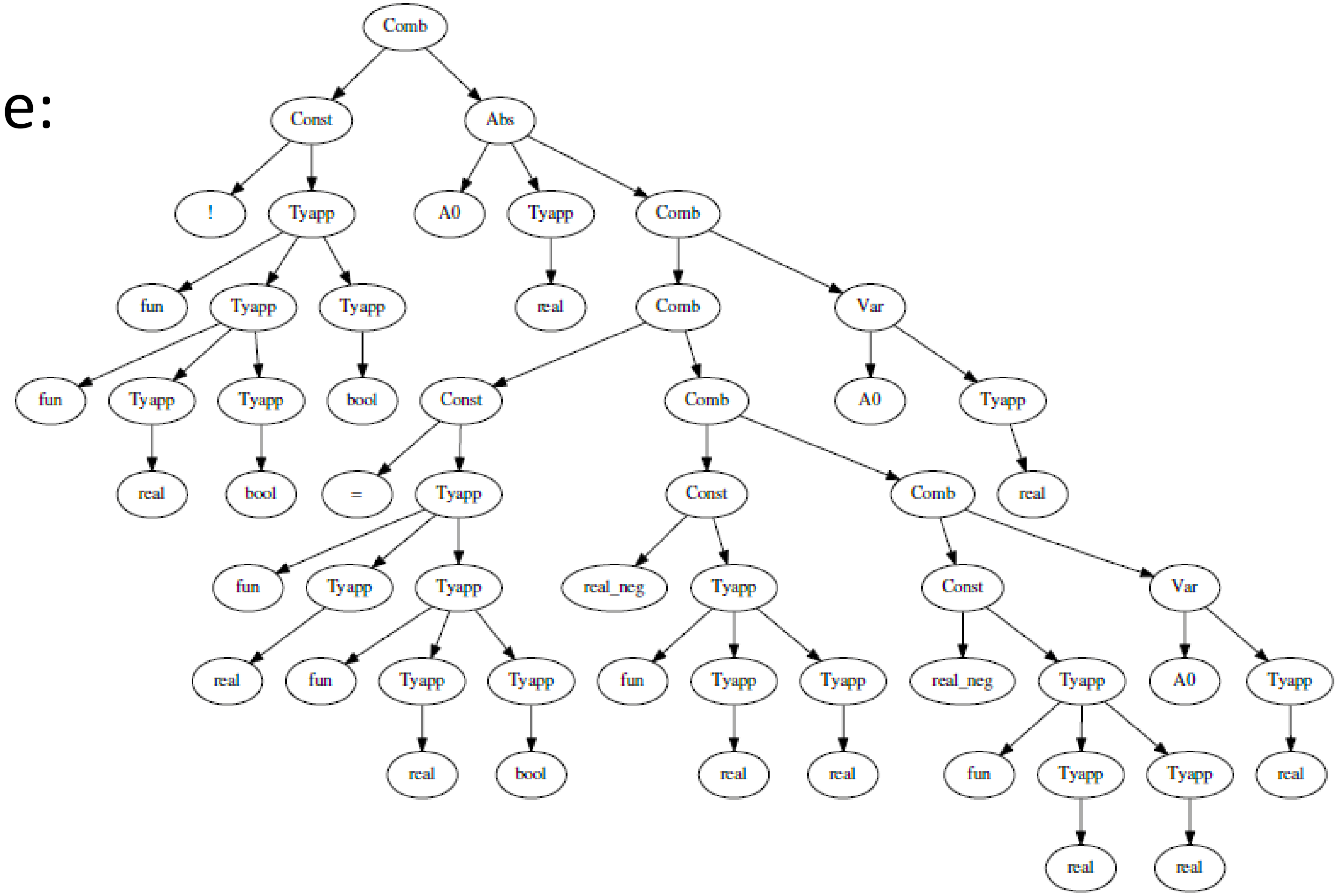
HOL Light lambda calculus internal term structure:

```
(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")))))
```

Statistical Parsing of Informalized HOL

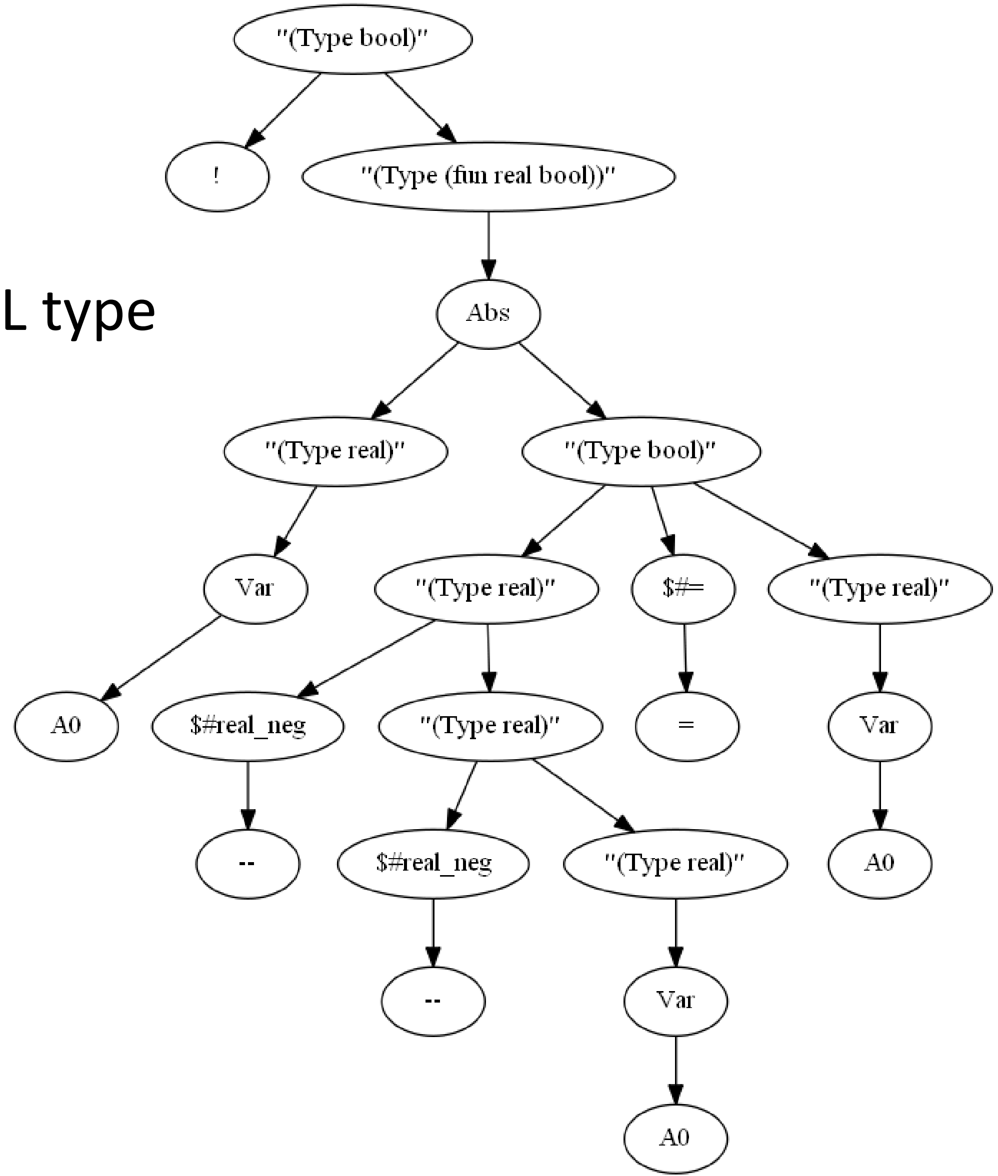
1) Training and testing examples are exported form Flyspeck formulae

Example:



Statistical Parsing of Informalized HOL

2) Conversion into a Grammar Tree
   - Terminals exactly compose textual form
   - Annotate each (nonterminal) symbol with its HOL type
   - Also “semantic (formal)” nonterminals annotate overloaded terminals



Example:

("(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))))))

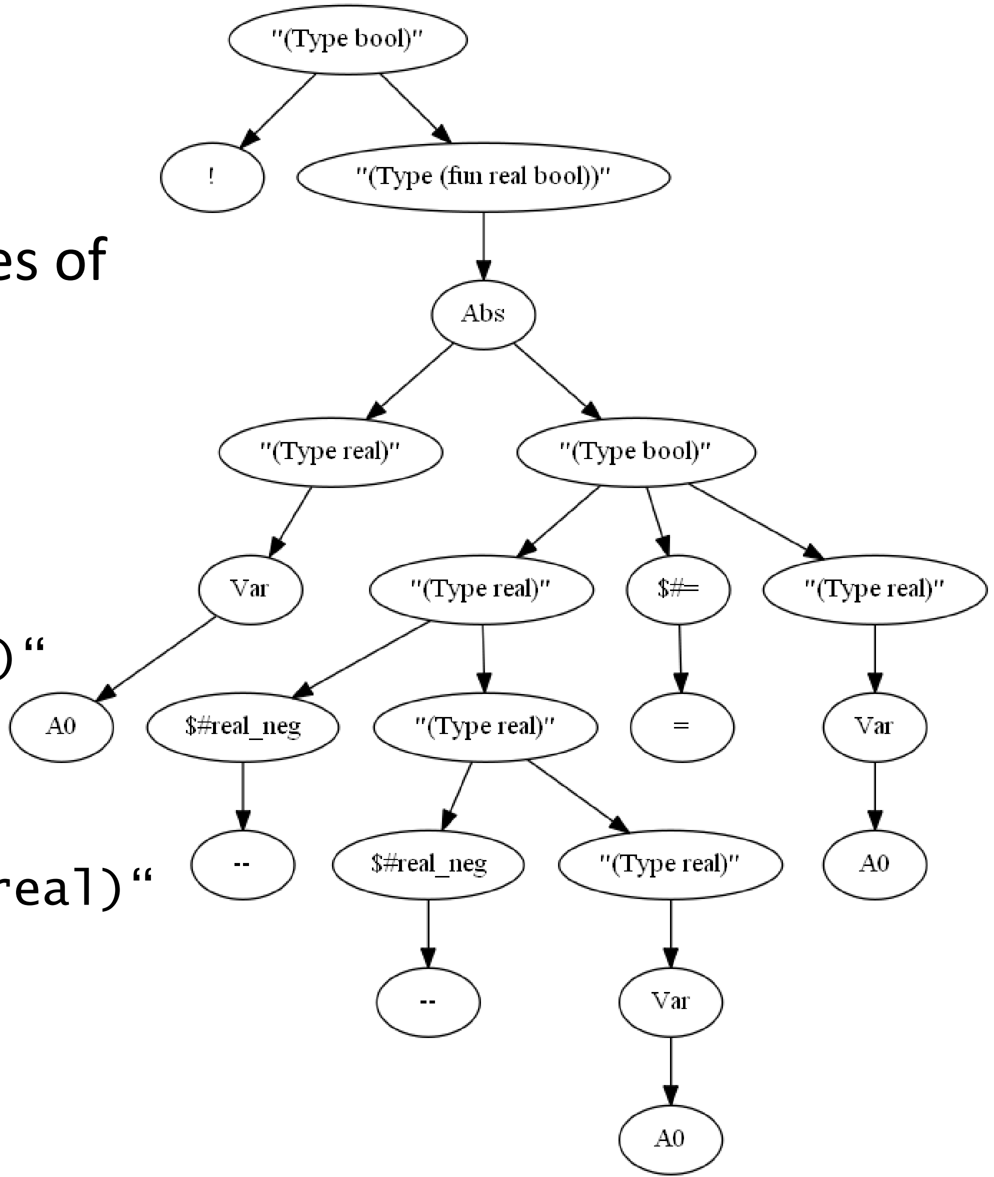
Corresponding textual form: `! A0 -- -- A0 = A0`

Statistical Parsing of Informalized HOL

3) Induce PCFG (Probabilistic Context-Free Grammar) from the trees
   - Grammar rules are obtained from the inner nodes of each grammar tree

Example:

```
          "(Type bool)" →  ! "(Type(fun real bool))“
"(Type(fun real bool))" →  Abs
                   Abs  →  "(Type real)“ "(Type bool)“
         "(Type real)“  →  Var
         "(Type real)“  →  $#real_neg "(Type real)“
                   Var  →  A0
         "(Type bool)“  →  "(Type real)“ $#= "(Type real)“
            $#real_neg →  --
                   $#= →  =
```



Statistical Parsing of Informalized HOL

3) Induce PCFG (Probabilistic Context-Free Grammar) from the trees
   - Grammar rules are obtained from the inner nodes of each grammar tree
   - Probabilities are computed from the frequencies

Example:

			freq:	prob:
"(Type bool)"	→	! "(Type(fun real bool))“	1	1/2
"(Type(fun real bool))"	→	Abs	1	1
Abs	→	"(Type real)“ "(Type bool)“	1	1
"(Type real)“	→	Var	3	3/5
"(Type real)“	→	$#real_neg "(Type real)“	2	2/5
Var	→	A0	3	1
"(Type bool)“	→	"(Type real)“ $#= "(Type real)“	1	1/2
$#real_neg	→	--	2	1
$#=	→	=	1	1

Statistical Parsing of Informalized HOL

3) Induce PCFG (Probabilistic Context-Free Grammar) from the trees
   - Grammar rules are obtained from the inner nodes of each grammar tree
   - Probabilities are computed from the frequencies
   - Grammar rules are binarized for efficient parsing (by CYK algorithm)

(around 20K grammar rules in Flyspeck case)

Example:

rule			freq:	prob:
"(Type bool)"	→	! "(Type(fun real bool))“	1	1/2
"(Type(fun real bool))"	→	Abs	1	1
Abs	→	"(Type real)“ "(Type bool)“	1	1
"(Type real)“	→	Var	3	3/5
"(Type real)“	→	$#real_neg "(Type real)“	2	2/5
Var	→	A0	3	1
"(Type bool)“	→	N1 "(Type real)“	1	1/2
N1	→	"(Type real)“ $#=	1	1
$#real_neg	→	--	2	1
$#=	→	=	1	1

Statistical Parsing of Informalized HOL

4) The learning part is done

- Rules probabilities can be further tuned for even better parsing results (Inside-Outside algorithm)

- Binarization should be designed with respect to possible reconstruction of original grammar trees

Statistical Parsing of Informalized HOL

4) Now, CYK dynamic-programming algorithm can be used for parsing ambiguous sentences

input:

- sentence – a sequence of words
- learned binarized PCFG

output:

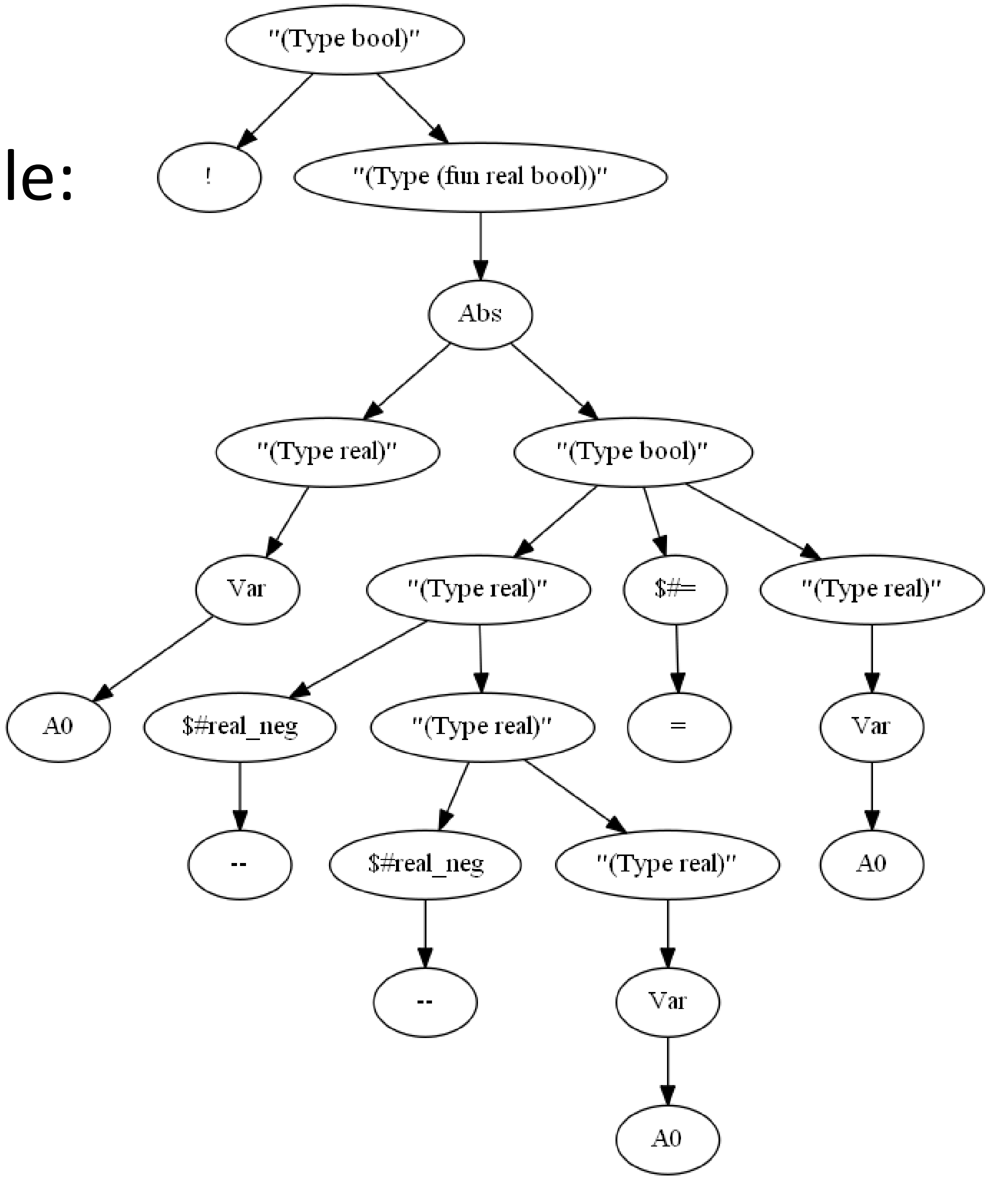
- N - most probable parse trees

where N is a parameter of CYK algorithm that can significantly affect the time complexity of parsing process

Problems with PCFG and CYK algorithm

- It is not possible to guarantee same type of same variables on different positions
- It is not possible to correctly handle types of lambda abstractions
- Above simple semantic pruning affects the parsing a lot!

Example:



Problems with PCFG and CYK algorithm

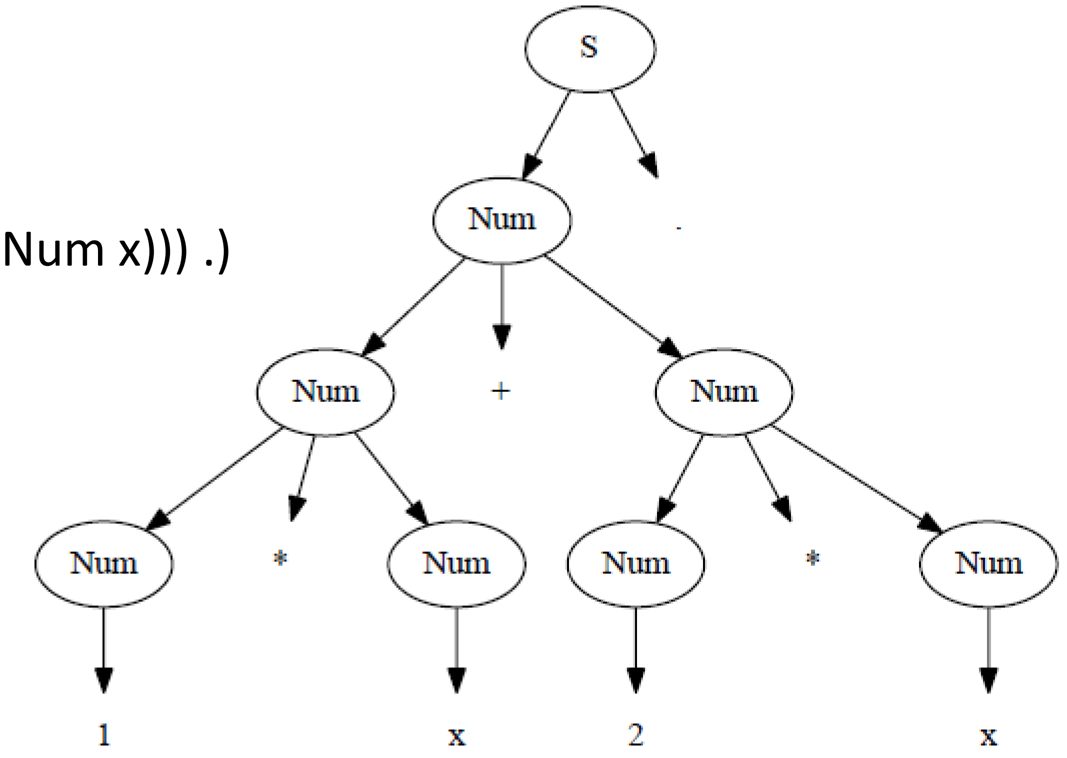
- Standard PCFG cannot handle any context of grammar rules.
  This effect can be seen on priorities of operators and type prediction of overloaded symbols.

Example:

input sentence: `1 * x + 2 * x.`

correct parsing tree:
(S (Num (Num (Num 1) * (Num x)) + (Num (Num 2) * (Num x))) .)

derived grammar rules:
S -> Num .
Num -> Num + Num
Num -> Num * Num
Num -> 1
Num -> 2
Num -> x



Problems with PCFG and CYK algorithm

- Standard PCFG cannot handle any context of grammar rules.
  This effect can be seen on priorities of operators and type prediction of overloaded symbols.

Example:

all possible parses according to the grammar:

```
1) (S (Num (Num 1) * (Num (Num (Num x) + (Num 2)) * (Num x))) .)
2) (S (Num (Num 1) * (Num (Num x) + (Num (Num 2) * (Num x)))) .)
3) (S (Num (Num (Num 1) * (Num (Num x) + (Num 2))) * (Num x)) .)
4) (S (Num (Num (Num (Num 1) * (Num x)) + (Num 2)) * (Num x)) .)
5) (S (Num (Num (Num 1) * (Num x)) + (Num (Num 2) * (Num x))) .)
```

probability of every parsed term is same =

p(`S -> Num .`) · p(`Num -> Num + Num`) · p(`Num -> Num * Num`) · p(`Num -> Num * Num`) · p(`Num -> 1`) · p(`Num -> 2`) · p(`Num -> x`) · p(`Num -> x`)

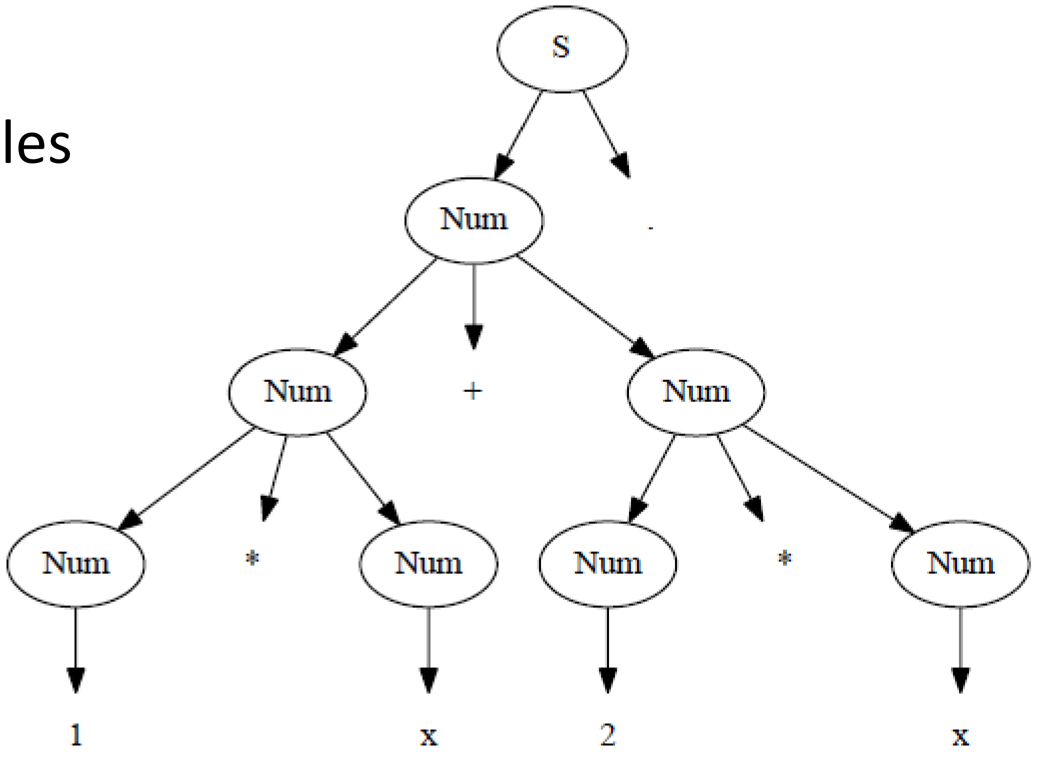
Problems with PCFG and CYK algorithm

- Standard PCFG cannot handle any context of grammar rules.
  This effect can be seen on priorities of operators and type prediction of overloaded symbols.

Example:

S -> Num .
Num -> Num + Num
Num -> Num * Num
Num -> 1
Num -> 2
Num -> x
S -> (Num Num + Num) .
Num -> (Num Num * Num) + (Num Num * Num)
Num -> (Num 1) * (Num x)
Num -> (Num 2) * (Num x)

subtree extension rules



Problems with PCFG and CYK algorithm

- Standard PCFG cannot handle any context of grammar rules.
  This effect can be seen on priorities of operators and type prediction of overloaded symbols.

Example:

The best (the most probable) parse according to the new grammar:

```
(S (Num (Num (Num 1) * (Num x)) + (Num (Num 2) * (Num x))) .)
```

Probability of the best parse =

p(`Num -> (Num 1) * (Num x)`) · p(`Num -> (Num 2) * (Num x)`)

· p(`Num -> (Num Num * Num) + (Num Num * Num)`)

· p(`S -> Num .`)

Parsing and Type-checking over Flyspeck (without subtrees PCFG extension)

- 698,549 of the parse trees typecheck (221,145 do not)
- 302,329 distinct (modulo alpha) HOL formulae
- For each HOL formula we try to prove it with a single AI-ATP method
- 70,957 (23%) can be automatically proved (but 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: 9.3

Parsing and Type-checking over Flyspeck (with subtrees PCFG extension)

- combination of subtrees with depths from 4 to 8
- ~~70,957 (23%)~~ **?** can be automatically proved
- In ~~39.4%~~ **75.7%** of the 22,000 Flyspeck sentences the correct (training) HOL parse tree is among the best 20 parses
- its average rank: ~~9.3~~ **1.9**

Future Work

- More corpora -> more alignments -> more knowledge -> …
- Smarter parsing methods
  different shapes of subtrees
  better matching patterns
  neural networks instead of subtrees (or instead of the whole parser)
- Tighter integration of probabilistic parsing with semantic pruning
- Incremental self-learning system:
  train on some data → parse → typecheck/prove the parses ...
  ... and thus get more data to train on → loop ...
- Implement backtracking into parsing process
  in case there is a point without any provable parse
- integrate into AI/ATP self-improving systems (MaLARea, BliStr, ...)