Learning Domain-Specific Grammars from Examples

Herbert Lange
Computer Science and Engineering
University of Gothenburg and Chalmers University of Technology

CLASP Seminar
April 22rd, 2020

Use Case: Language Learning

Use Case: Language Learning

- We have a grammar-based language learning application!
- The application uses a restricted grammar to automatically generate exercises to teach a specific language construction
- We have large, wide-coverage (Resource) Grammars!
- How can we get the restricted exercise grammar?

Infer the grammar from the Resource Grammar and example sentences

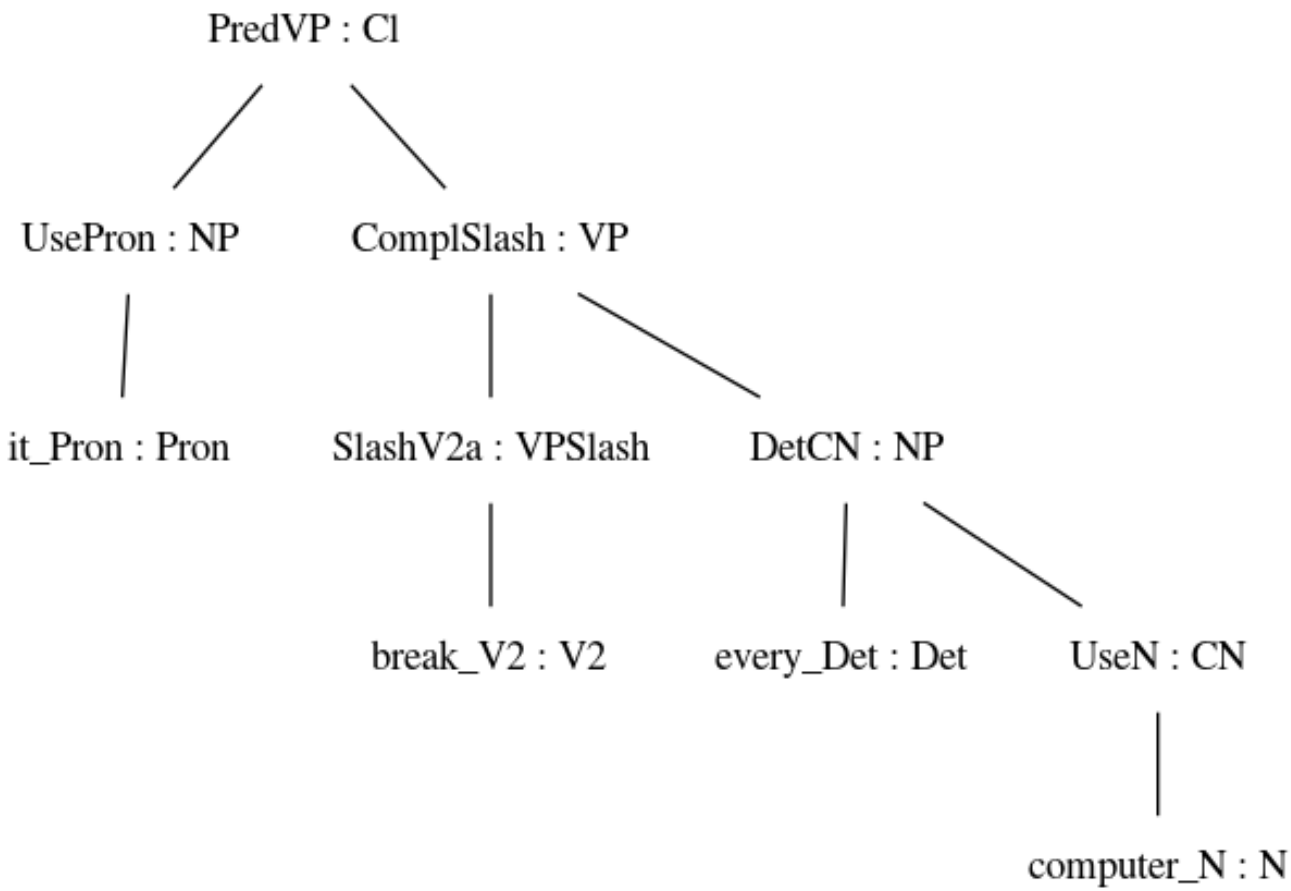
Example Grammar

```
-- Syntactic rules
UseN        : N     -> CN                ;
UsePron     : Pron  -> NP                ;
DetCN       : Det   -> CN   -> NP        ;
ComplSlash  : VPSlash -> NP -> VP        ;
SlashV2a    : V2 -> VPSlash              ;
PredVP      : NP -> VP -> Cl             ;

-- Lexical items
many_Det, every_Det, few_Det : Det ;
boy_N, girl_N : N ;
friend_N, king_N, house_N, book_N, computer_N : N ;
he_Pron, she_Pron, it_Pron, they_Pron : Pron ;
close_V2, break_V2, love_V2, read_V2, hit_V2 : V2 ;
```

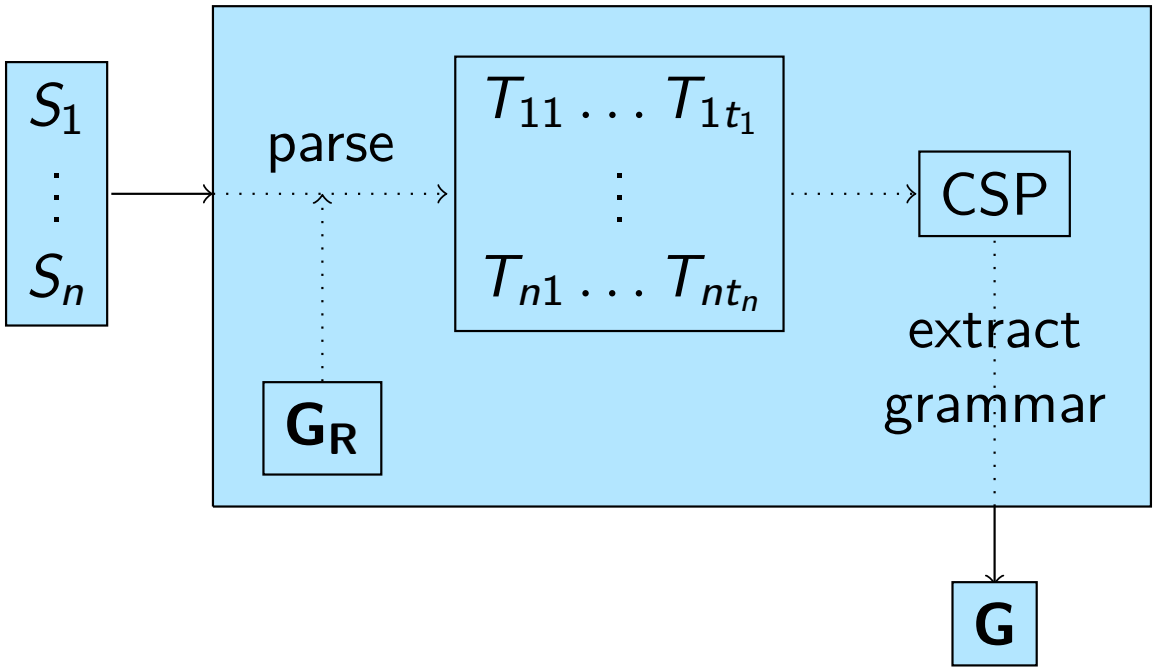
Example Sentence

It breaks every computer

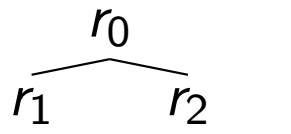
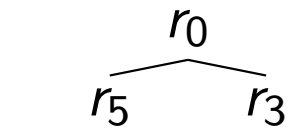
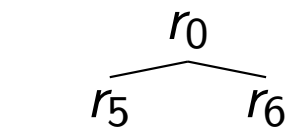


Grammar Learning Chapter 1: Simple Subgrammars

Inferring a grammar



Constraint Satisfaction Problem: Logic Variables

			Flattened:
	Tree T_{11}		$\{r_0, r_1, r_2, r_3, r_4\}$
Sentence S_1			
	Tree T_{12}		$\{r_0, r_5, r_3, r_1, r_4\}$
Sentence S_2	Tree T_{21}		$\{r_0, r_5, r_6, r_7, r_8\}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Constraint Satisfaction Problem: Logic Constraints

- All sentences have to be covered $S_1 \wedge S_2 \wedge \ldots$
- At least one tree per sentence has to be covered:
  $S_1 \rightarrow T_{11} \vee T_{12}$
  $S_2 \rightarrow T_{21}$
  $\ldots$
- All rules in a tree have to be covered:
  $T_{11} \rightarrow r_0 \wedge r_1 \wedge r_2 \wedge r_3 \wedge r_4$
  $T_{12} \rightarrow r_0 \wedge r_5 \wedge r_3 \wedge r_1 \wedge r_4$
  $T_{21} \rightarrow r_0 \wedge r_5 \wedge r_6 \wedge r_7 \wedge r_8$
  $\ldots$

Constraint Satisfaction Problem: Logic Constraints

- All sentences have to be covered $n \leq S_1 + S_2 + \ldots S_n$
- At least one tree per sentence has to be covered:
  $S_1 \leq T_{11} + T_{12}$
  $S_2 \leq T_{21}$
  $\ldots$
- All rules in a tree have to be covered:
  $5 * T_{11} \leq r_0 + r_1 + r_2 + r_3 + r_4$
  $5 * T_{12} \leq r_0 + r_5 + r_3 + r_1 + r_4$
  $5 * T_{21} \leq r_0 + r_5 + r_6 + r_7 + r_8$
  $\ldots$

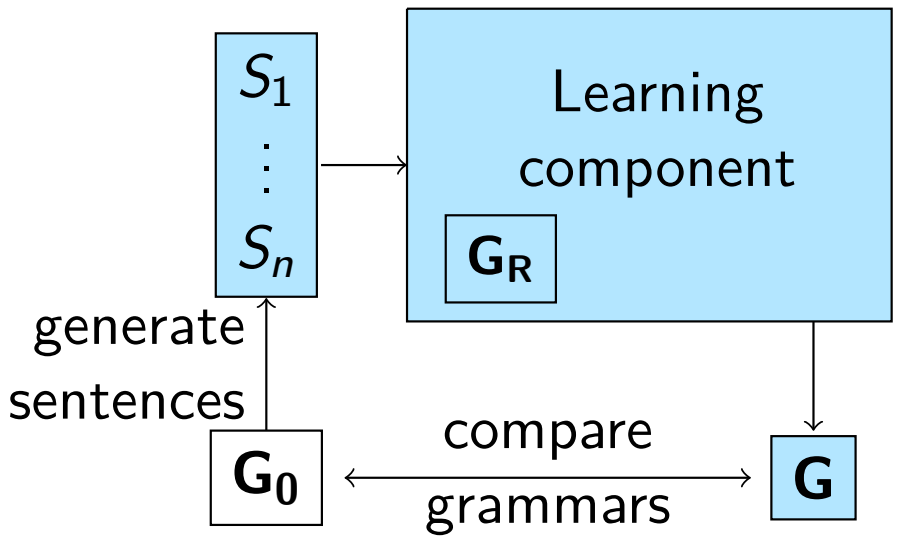
Constraint Optimization Problem: Objective Function

Minimize the variable assignment satisfying the constraints according to:

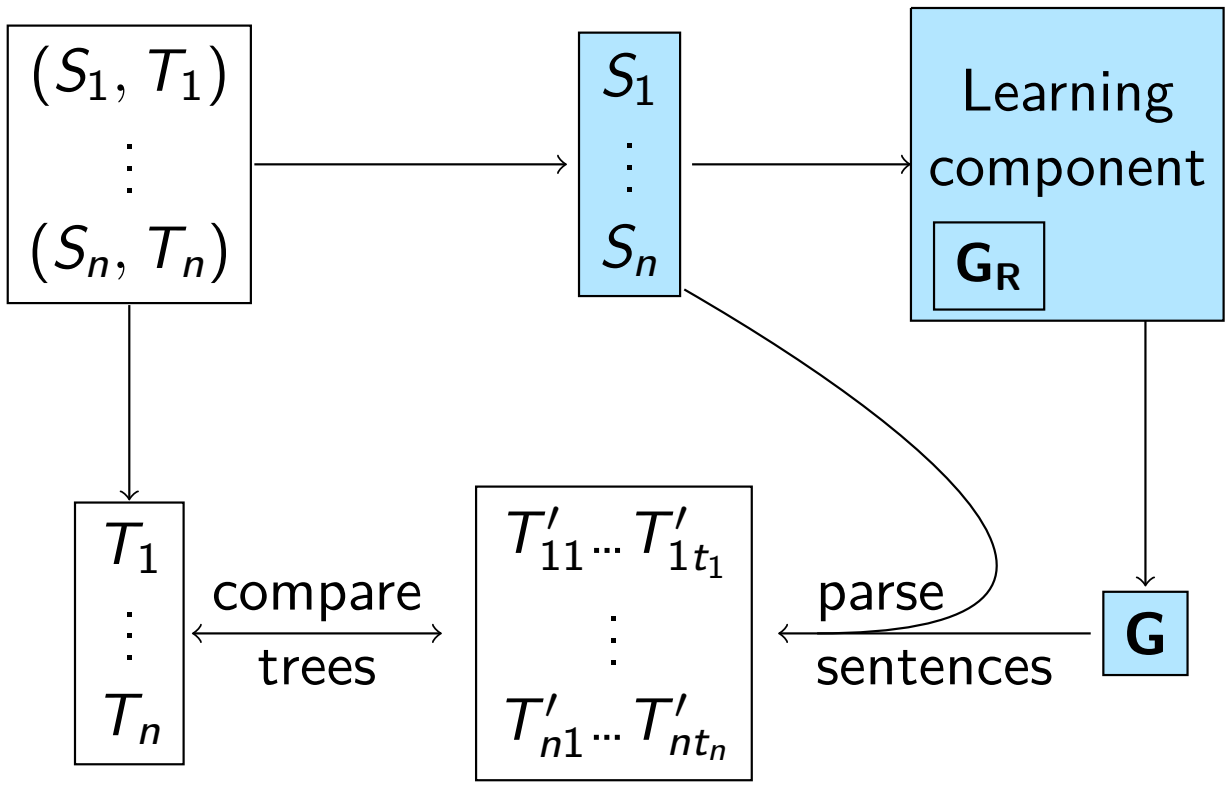
- **Rules:** Number of rules in the resulting grammar (i.e., Reducing the grammar size)
- **Trees:** Number of all initial parse trees T_{ki} that are, intended or not, valid in the resulting grammar (i.e., Reducing the ambiguity)
- **Rules+Trees:** Sum of **Rules** and **Trees**
- **Weighted:** Modification of **Rules+Trees** where each rule is weighted by the number of occurrences, prefering more common rules

Evaluation

Experiment 1: Rebuilding a Known Grammar



Experiment 2: Comparing to a Treebank



Evaluation

- Rebuilding a Known Grammar

$$\textit{Precision} = \frac{|\mathbf{R_0} \cap \mathbf{R}|}{|\mathbf{R}|} \quad \textit{Recall} = \frac{|\mathbf{R_0} \cap \mathbf{R}|}{|\mathbf{R_0}|}$$

  Where $\mathbf{R_0}$ the rules of the original grammar and $\mathbf{R}$ the rules of the inferred grammar

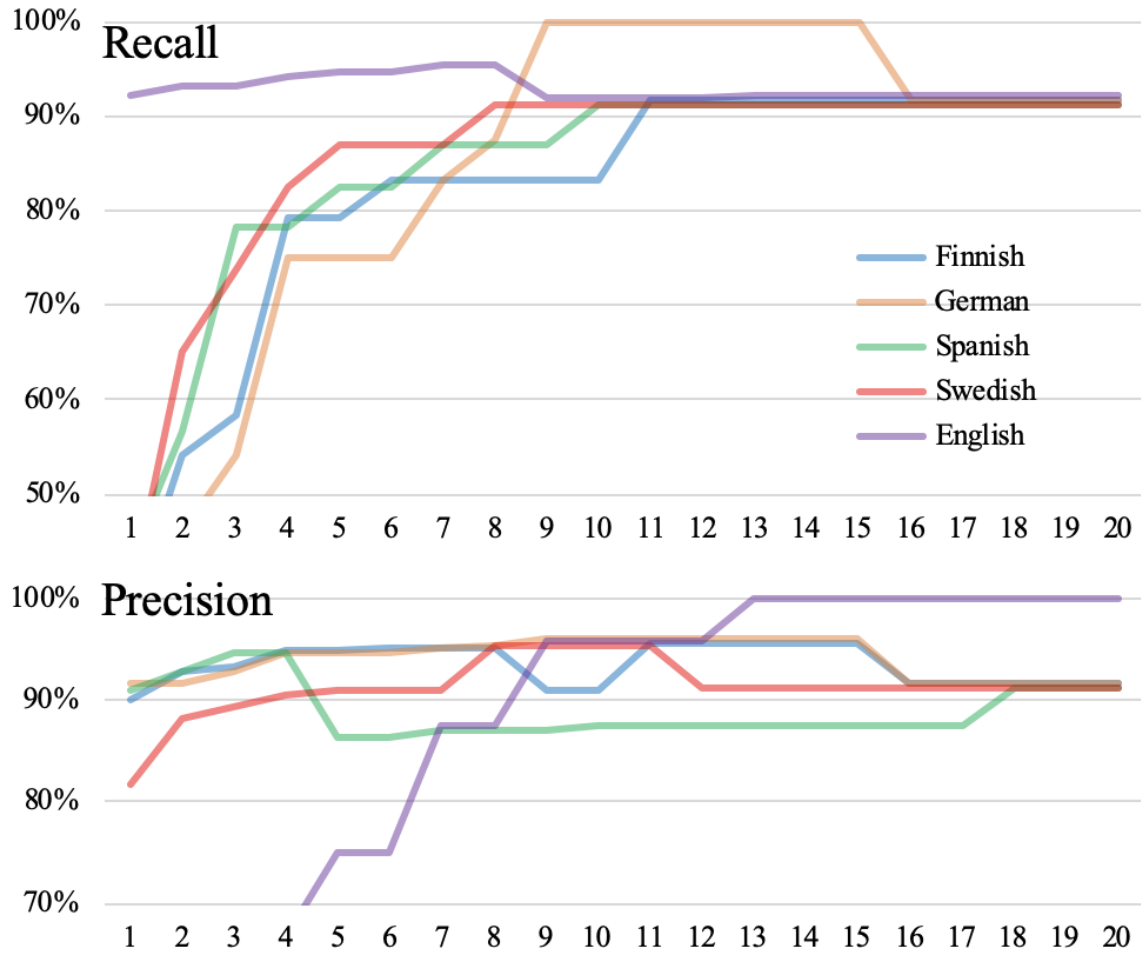
- Comparing to a Treebank

  Accuracy percentage of sentences where the correct tree is found

  Ambiguity average number of parse trees per sentence

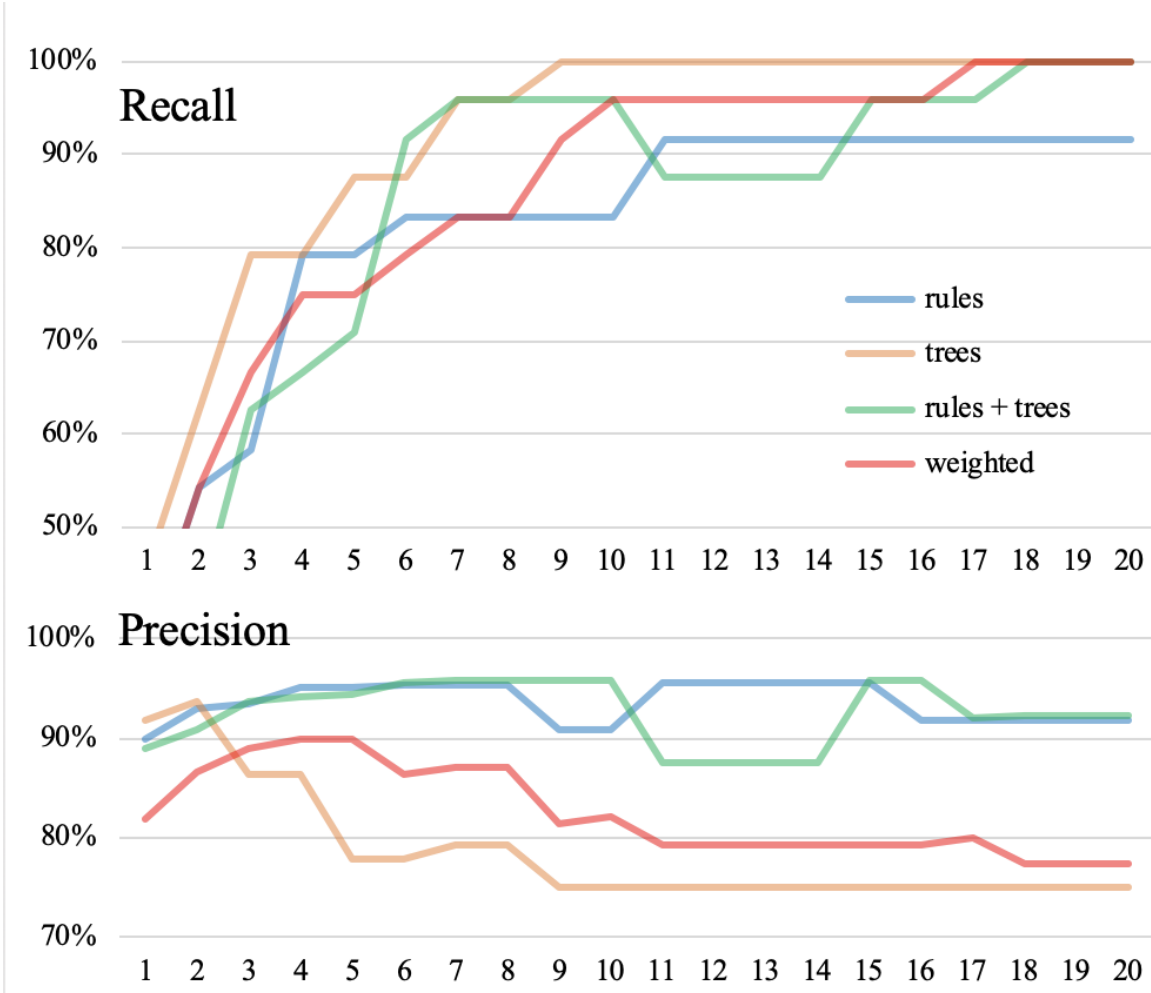
Results

Results: Objective Function **rules** and Various Languages



Number of examples used for training

Results: Finnish and Various Objective Functions



Number of examples used for training

Results: Comparing to a Treebank

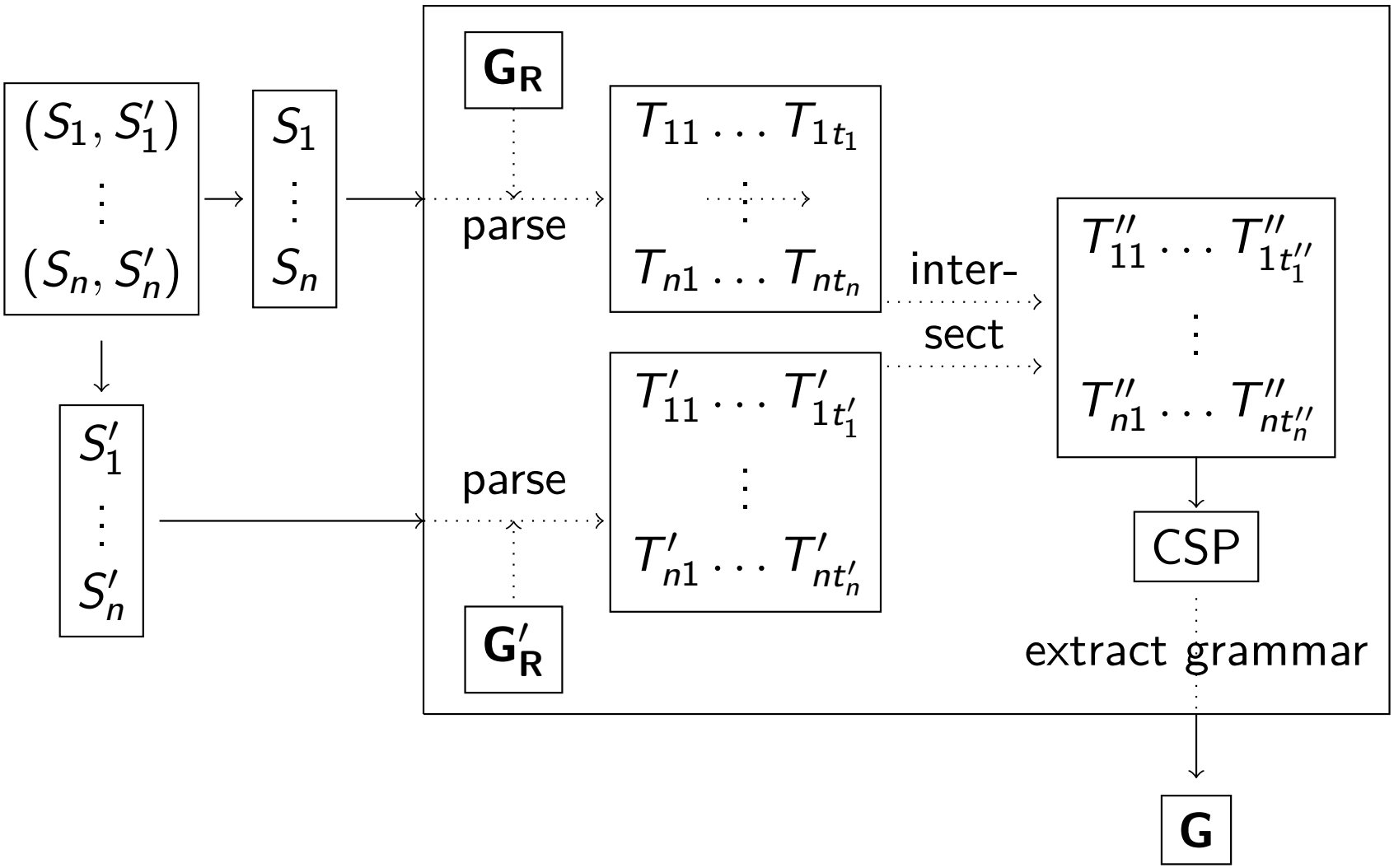
		Monolingual			
		Rules+Trees		**Weighted**	
	Size	Accuracy	Ambiguity	Accuracy	Ambiguity
Finnish	22	5%	1.0	91%	115
German	16	75%	1.1	100%	2.0
Swedish	10	100%	1.1	100%	2.8
Spanish	13	100%	1.2	92%	3.7

Finnish Treebank

```
laula laulu
sing a song
PhrUtt NoPConj (UttImpSg PPos (ImpVP (ComplSlash (SlashV2a sing_V2)
  (DetCN (DetQuant IndefArt NumSg) (UseN song_N))))) NoVoc

laulakaa laulu
sing a song
PhrUtt NoPConj (UttImpPl PPos (ImpVP (ComplSlash (SlashV2a sing_V2)
  (DetCN (DetQuant IndefArt NumSg) (UseN song_N))))) NoVoc
...
minä haluan laulaa laulun suihkussa
I want to sing a song in the shower
PhrUtt NoPConj (UttS (UseCl (TTAnt TPres ASimul) PPos (PredVP
  (UsePron i_Pron) (ComplVV want_2_VV (AdvVP (ComplSlash
  (SlashV2a sing_V2) (DetCN (DetQuant IndefArt NumSg) (UseN song_N)))
  (PrepNP in_Prep (DetCN (DetQuant DefArt NumSg) (UseN shower_N))))))))
```

Bilingual Learning



Results: Bilingual Treebank

		Bilingual			
		Rules+Trees		**Weighted**	
	Size	Accuracy	Ambiguity	Accuracy	Ambiguity
Finnish	22	86%	4.9	96%	8.7
German	16	94%	1.1	100%	1.5
Swedish	10	100%	1.1	100%	1.2
Spanish	13	100%	1.2	100%	2.3

Using English as a second language

Conclusion so far:

- We can learn relevant sub-grammars from very few sentences
- Using language pairs boosts the process

Future Work

~~Atomic unit:~~ ~~Scale up to larger subtrees~~

~~Negative examples:~~ ~~Include examples that should **not** be covered by the grammar~~

Multilingual learning: Explore influence of bi-/multilingual learning

Problem Size: Move from trees to parse chart

Other Grammar Formalisms: Try e.g. with TAG or HPSG

Grammar Learning Chapter 2: Beyond Simple Subgrammars

Negative Examples

Idea:
"Include A, B and C but not X, Y and Z"
To exclude a tree, not all rules can be included:

$$\neg(r_0 \wedge r_1 \wedge \cdots \wedge r_n) \equiv \neg r_0 \vee \neg r_0 \vee \cdots \vee \neg r_0$$

As linear constraint:

$$r_0 + r_1 + \cdots + r_n \leq n$$

Example: Dyck Language

```
concrete Dyck of DyckAbs {
  lincat S = Str ;
  lin
    -- empty, leftp, rightp, lefts, rights : S ;
    empty = "" ;
    leftp = "(" ;
    rightp = ")" ;
    lefts = "[" ;
    rights = "]" ;
    -- bothp, boths : S -> S ;
    bothp s = "(" ++ s ++ ")" ;
    boths s = "[" ++ s ++ "]" ;
    -- combine : S -> S -> S ;
    combine s1 s2 = s1 ++ s2 ;
}
```

```
DyckAbs> p "[ ( ) ]"
boths (bothp empty)
boths (combine leftp rightp)
combine lefts (combine leftp (combine rightp rights))
combine lefts (combine (bothp empty) rights)
combine lefts (combine (combine leftp rightp) rights)
combine (combine lefts leftp) (combine rightp rights)
combine (combine lefts (bothp empty)) rights
combine (combine lefts (combine leftp rightp)) rights
combine (combine (combine lefts leftp) rightp) rights
```

Positive examples:

```
[ ]
( )
( ) ( )
```

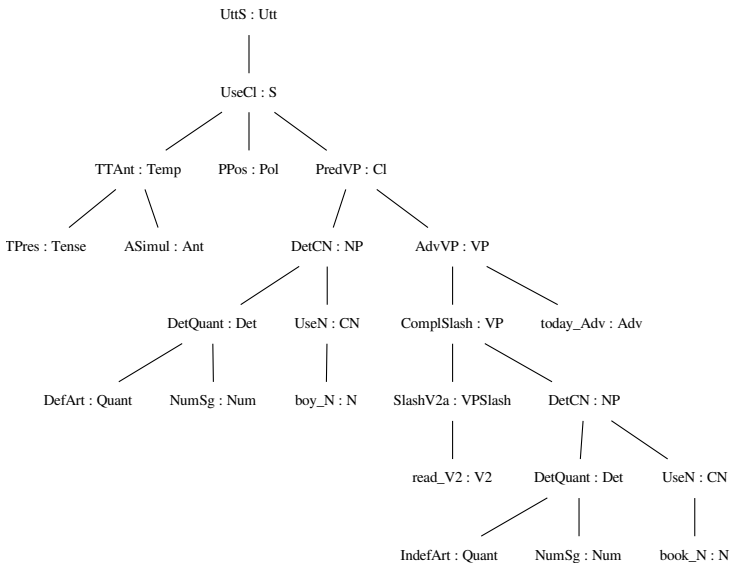
Negative examples:

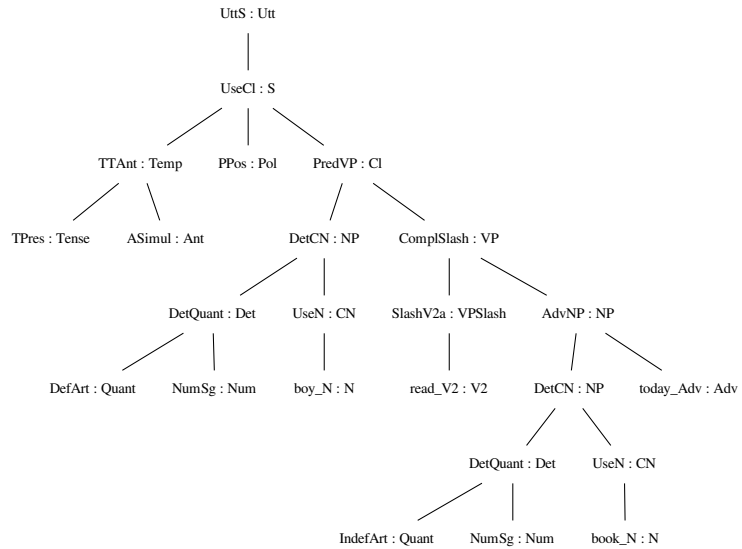
```
( ]
[ )
(
```

Demo

Example: Adverbials

```
AdvNP : NP -> Adv -> NP ;     -- e.g. Paris today
AdvVP : VP -> Adv -> VP ;     -- e.g. sleep here
```



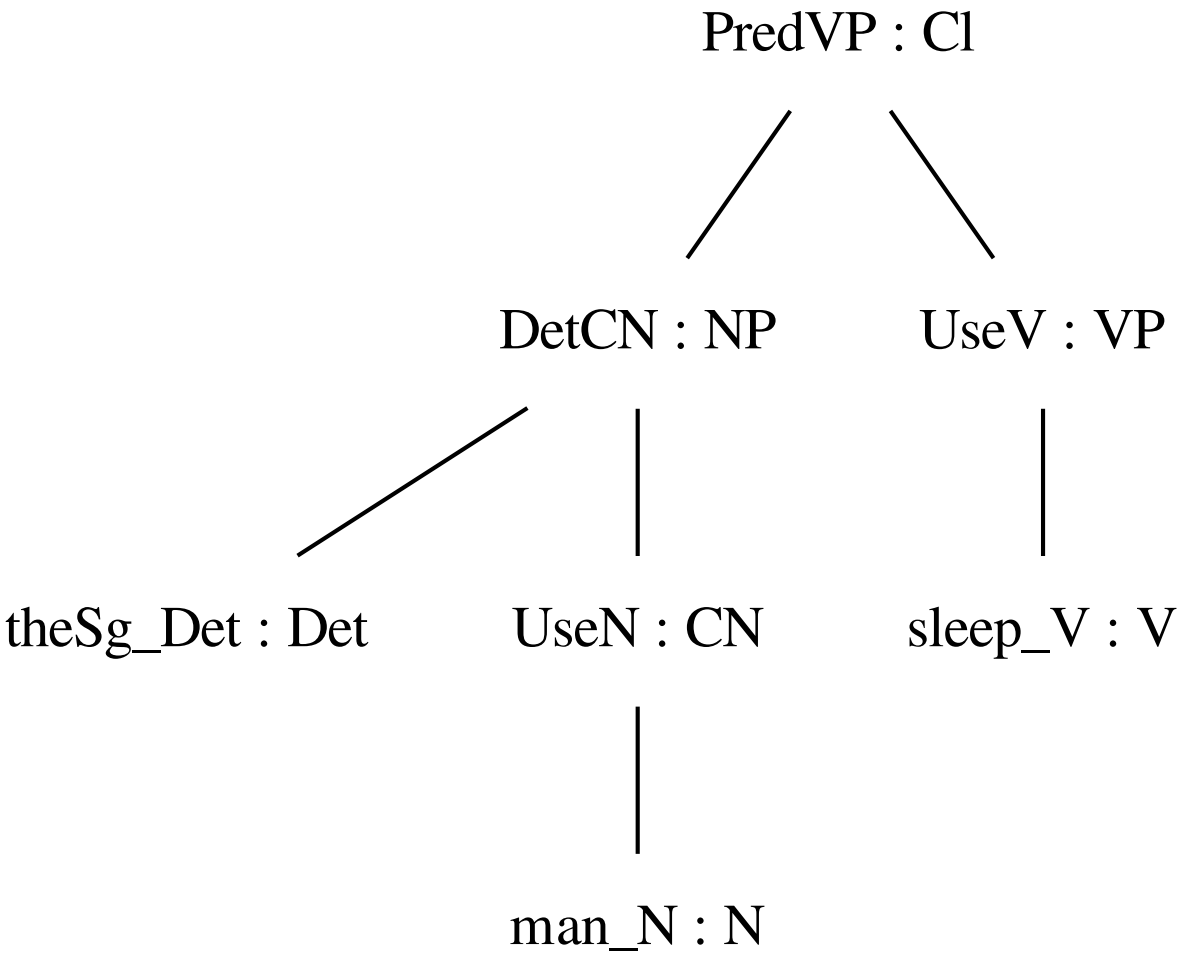


the boy reads a book today vs. **a book today comes*

Iterative Grammar Learning

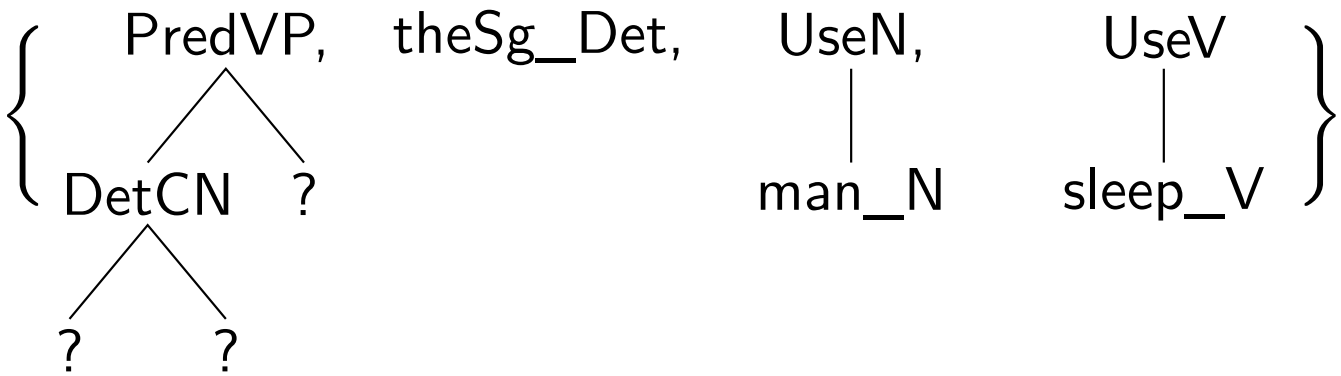
1. The starts with a set of positive examples, same as previously
2. Repeat until satisfied:
   - 2a. The system infers a grammar from the example sentences
   - 2b. The system randomly generates new example sentences
   - 2c. The user can mark sentences as acceptable or not and also add additional sentences

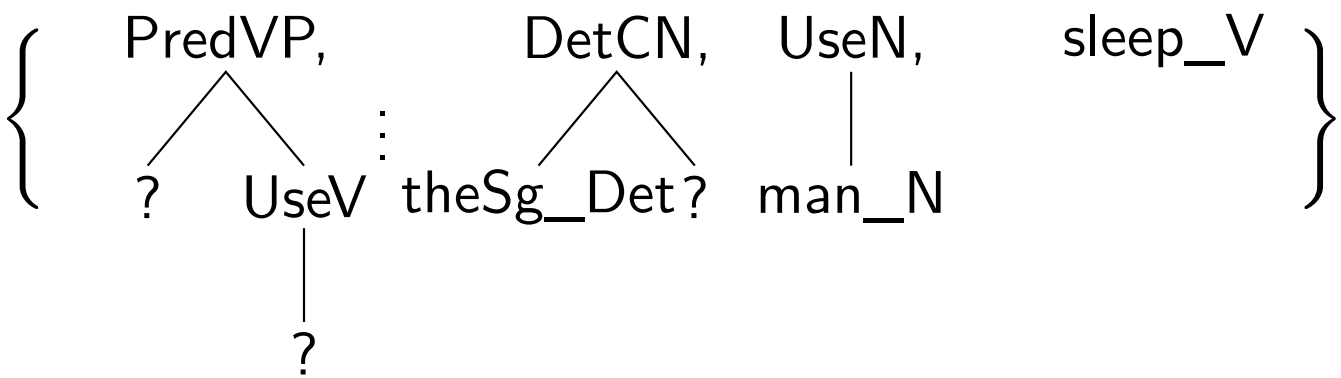
Subtrees


PredVP : Cl
DetCN : NP
UseV : VP
theSg_Det : Det
UseN : CN
sleep_V : V
man_N : N

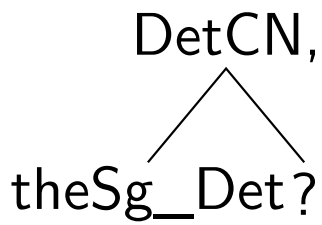
Split into subtrees (of maximum size 2)

{ PredVP, DetCN, UseV, UseN, theSg_Det, man_N,sleep_V }





Merging rules




```
DetCN : Det -> CN -> NP ;
theSg_Det : Det ;
-- Compose to new rule:
DetCN_theSg_Det : CN -> NP ;
```

Example: Dyck Language (again)

```
concrete Dyck2 of Dyck2Abs = {
  lincat Dyck, Open, Close = Str ;
  empty = "" ; -- empty : Dyck
  -- wrap : Open -> Dyck -> Close -> Dyck
  wrap o d c = o ++ d ++ c ;
  -- combine : Dyck -> Dyck -> Dyck
  combine d1 d2 = d1 ++ d2 ;
  -- leftp, lefts : Open
  leftp = "(" ;
  lefts = "[" ;
  -- rightp, rights : Close
  rightp = "]" ;
  rights = ")" ;
}
```

Example: Dyck Language (again)

Positive examples: *[( )]* and *[ ] ( )*
maximum subtree size 3 and at most 2 subtrees in each split

```
Empty : Dyck
Wrap#LeftP#?#RightP : Dyck -> Dyck
Wrap#LeftS#?#RightS : Dyck -> Dyck
Combine : Dyck -> Dyck -> Dyck
```

Problems

- Combinatorial Explosion: For a tree of size 7

Maximum subtree size	No. splits	No. subtrees
2	45	306
3	128	756
...	...	...

- Computational Effort: NP-completeness

Solutions

- Limit the number of subtrees per split
- Explore CSP instead of COP

Results

- We can learn both formal and natural language fragments using positive examples
- We can learn more challenging fragments using subtrees and by merging rules

Conclusion

- We can learn precise grammars using very few positive and negative examples
- We can create a human-centric, iterative learning process
- Merging rules allows us to create even more specific grammars

Future Work

~~Atomic unit:~~ ~~Scale up to larger subtrees~~

~~Negative examples:~~ ~~Include examples that should **not** be covered by the grammar~~

Multilingual learning: Explore influence of bi-/multilingual learning

Problem Size: Move from trees to parse chart

Other Grammar Formalisms: Try e.g. with TAG or HPSG

Other CSP methods: Explore use of SAT instead if 0/1 integer programming

Iterative Process: Implement and test the iterative learning process

Language Learning: Include grammar learning in the language learning application

Bonus

Bonus: Grammar Statistics

	Resource Grammar	Given Grammar
Syntactic Rules	284	24
Lexical Rules	591	47

Bonus: Treebank Statistics

	Sentences	Min Words	Max Words	Average Words
Finnish	22	2	6	3.9
German	16	2	5	4.4
Swedish	10	3	8	4.5
Spanish	13	3	6	4.5

Bonus: Adverbials (again)

The positive examples we use for training are:

- *I eat pizza with pineapple*
- *pizza with pineapple is delicious*
- *I run today*
- *I sleep now*
- *I run*

And the only negative example is:

- * *I eat pizza with scissors*

```
AdvNP#PrepNP : NP -> Prep -> NP -> NP
AdvVP#?#now_Adv : VP -> VP
AdvVP#?#today_Adv : VP -> VP
ComplSlash#SlashV2a : V2 -> NP -> VP
MassNP : CN -> NP
PositA#delicious_A : AP
PredVP : NP -> VP -> Cl
UseComp#CompAP : AP -> VP
UseN : N -> CN
UsePron#I_Pron : NP
UseV#run_V, UseV#sleep_V : VP
eat_V2 : V2
pineapple_N, pizza_N : N
with_Prep : Prep
```

Maximum subtree size	No. splits	No. subtrees
2	45	306
3	128	756
...	...	...

For subtrees with maximum size 2:

Maximum number subtrees	No. splits	No. subtrees
1	9	73
2	28	206
3	43	299
4	45	306
...	...	...