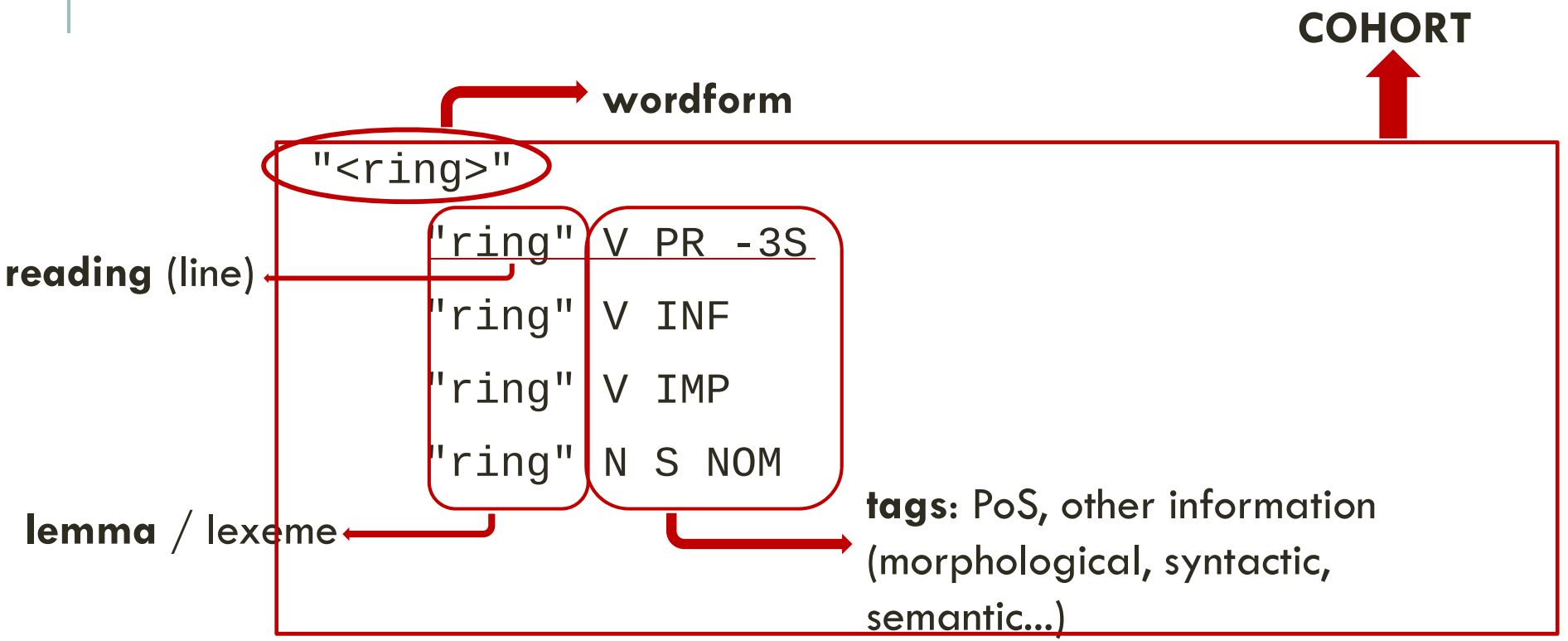
COMPUTATIONAL SYNTAX CONSTRAINT GRAMMAR (CG-3)

Ruben Urizar

CONSTRAINT GRAMMAR PARSER

- **Text-based** => It operates on a text stream, adding and removing information
- The information in the analysed texts is organized in:
  - ✓ **lines:**
    - — word form
    - — readings (analysis)
  - ✓ **cohorts:** presents all the analyses of the word form (one interpretation per line)
- **Contextual rules** => The information is modified using rules that refer to other lines and cohorts
- The **scope** of the rule is a sentence
- The contextual tests in the rules can be linked to each other (weak regular expressions)

CG FORMAT


COHORT
wordform
"<ring>"
"ring" V PR -3S
"ring" V INF
"ring" V IMP
"ring" N S NOM
reading (line)
lemma / lexeme
tags: PoS, other information (morphological, syntactic, semantic...)

He's adopted another blasted lame duck and has got him living here with him!

```
"<he_>"
       "he" <sam-> <*> <masc> PERS MASC 3S NOM
"<_s>"
       "be" <-sam> V PR 3S
       "we" <-sam> PERS 1P ACC
       "have" <-sam> V PR 3S
"<adopted>"
       "adopt" V PCP2 AKT
       "adopt" V PCP2 PAS
       "adopt" V PCP2 STA
       "adopt" V IMPF
"<another>"
       "another" INDP S NOM
       "another" DET S
"<blasted>"
       "blasted" ADJ POS
       "blast" <v.contact> V PCP2 AKT
       "blast" <v.contact> V PCP2 PAS
       "blast" <v.contact> V PCP2 STA
       "blast" <v.contact> V IMPF
"<lame>"
       "lame" V PR -3S
       "lame" V INF
       "lame" V IMP
       "lame" <jsick> ADJ POS
       "lame" <cc> N S NOM
"<duck>"
       "duck" V PR -3S
       "duck" V INF
       "duck" V IMP
       "duck" <Hprof> <comp2> N S NOM
"<and>"
       "and" KC
"<has>"
       "have" V PR 3S
"<got>"
       "get" <va+DIR> <vta+DIR> <vtk+ADJ> V IMPF
       "get" <va+DIR> <vta+DIR> <vtk+ADJ> V PCP2 AKT
       "get" <va+DIR> <vta+DIR> <vtk+ADJ> V PCP2 PAS
       "get" <va+DIR> <vta+DIR> <vtk+ADJ> V PCP2 STA
"<him>"
       "he" <masc> PERS MASC 3S ACC
"<living>"
       "living" <f-psych> <HH> <mon> <state> N S NOM
       "living" <jpert> <jppl> ADJ POS
       "live" <va+LOC> V PCP1
       "living" <pcp1> ADJ POS
       "living" <pcp1> N S NOM
"<here>"
       "here" <aloc> ADV
"<with>"
       "with" PRP
"<him>"
       "he" <masc> PERS MASC 3S ACC
"<$!>"
       "!" PU <<< @PU
```

CG was originally devised for disambiguation (choosing the correct reading/analysis for each word in a sentence).

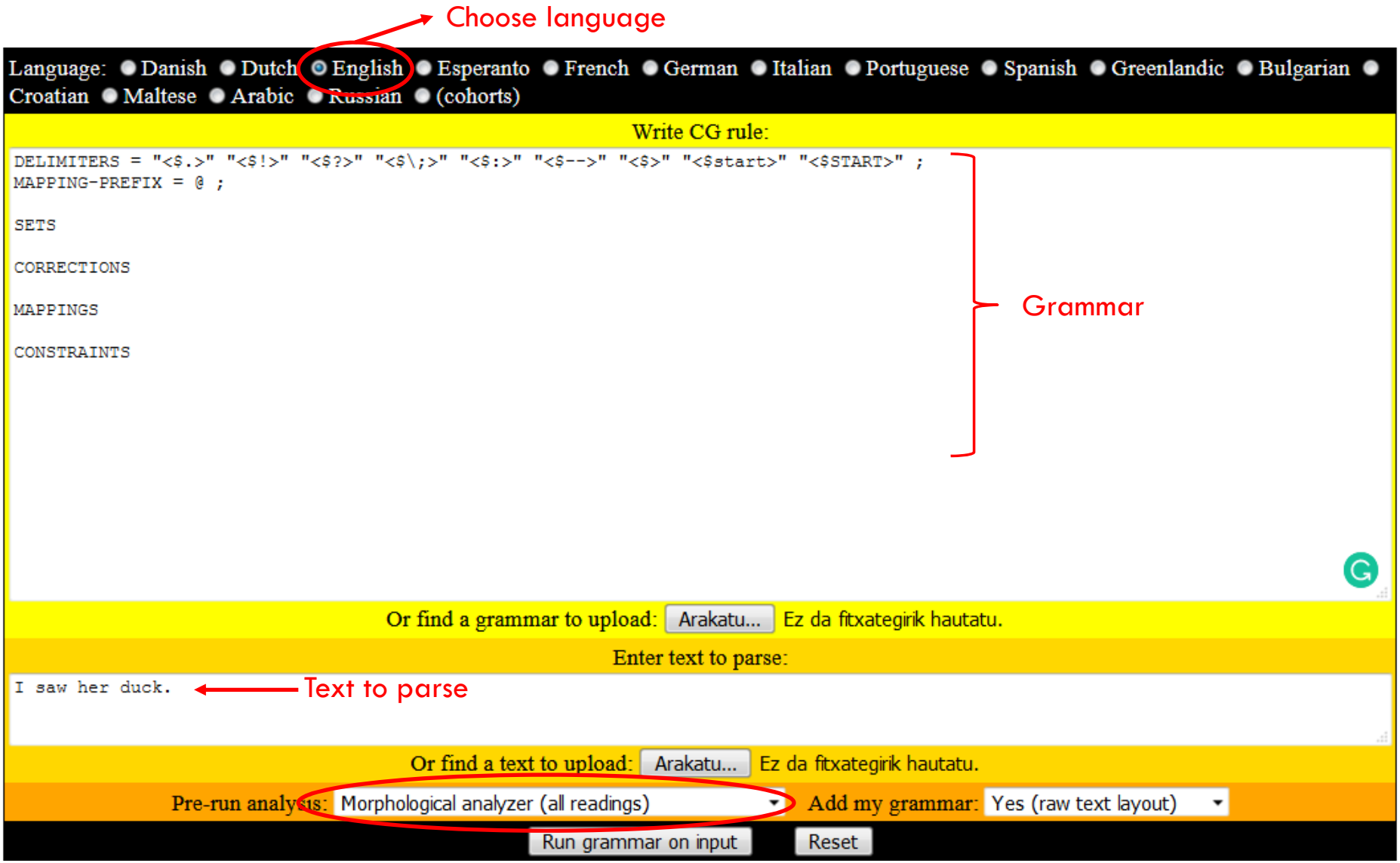
It aimed at discarding the wrong analyses and/or choosing the correct one in the specific context of the sentence.

He's adopted another blasted lame duck and has got him living here with him!

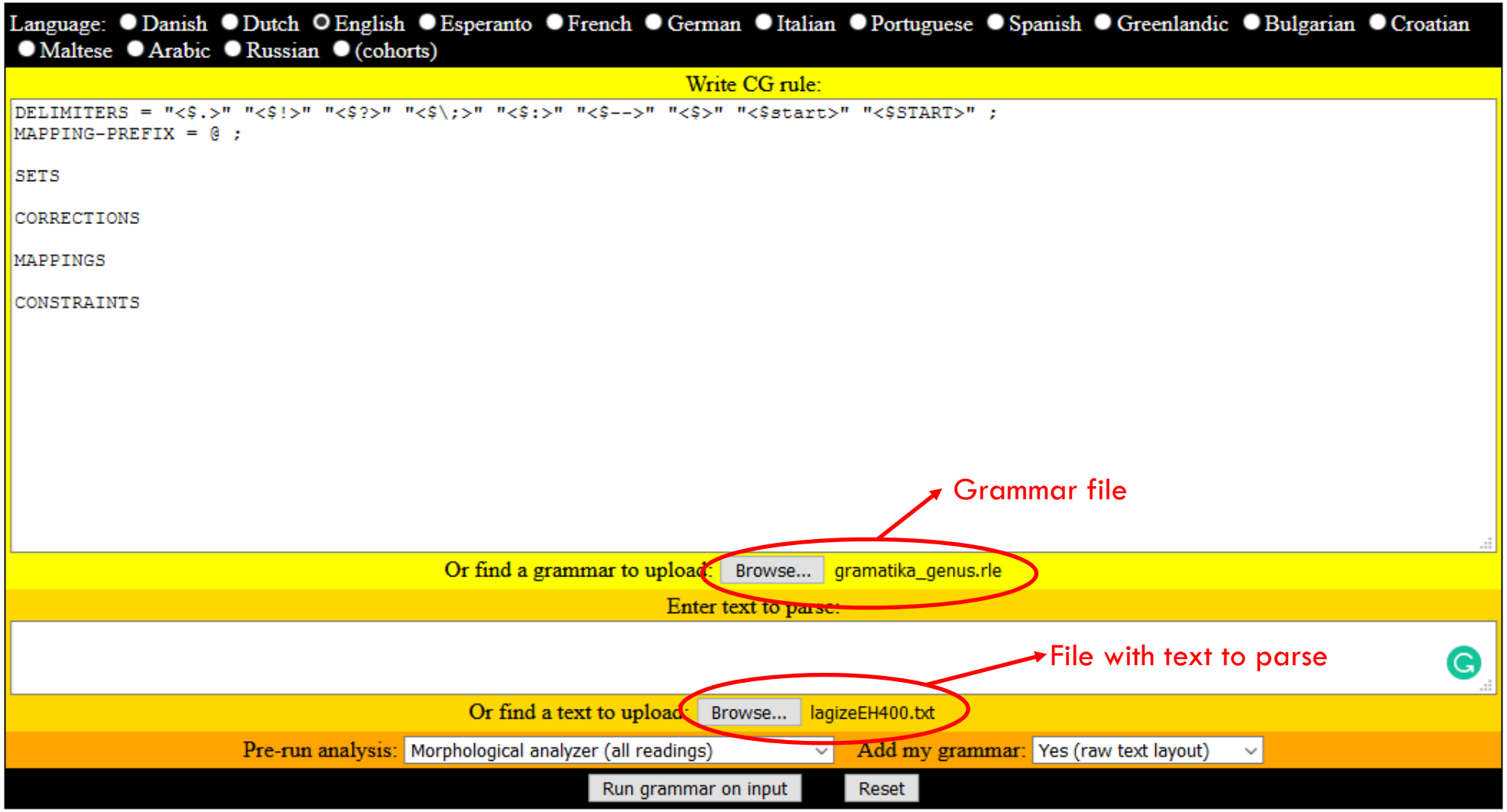
```
"<he_>"
       "he" <sam-> <*> <masc> PERS MASC 3S NOM
"<_s>"
       "be" <-sam> V PR 3S
       "we" <-sam> PERS 1P ACC
       "have" <-sam> V PR 3S
"<adopted>"
       "adopt" V PCP2 AKT
       "adopt" V PCP2 PAS
       "adopt" V PCP2 STA
       "adopt" V IMPF
"<another>"
       "another" INDP S NOM
       "another" DET S
"<blasted>"
       "blasted" ADJ POS
       "blast" <v.contact> V PCP2 AKT
       "blast" <v.contact> V PCP2 PAS
       "blast" <v.contact> V PCP2 STA
       "blast" <v.contact> V IMPF
"<lame>"
       "lame" V PR -3S
       "lame" V INF
       "lame" V IMP
       "lame" <jsick> ADJ POS
       "lame" <cc> N S NOM
"<duck>"
       "duck" V PR -3S
       "duck" V INF
       "duck" V IMP
       "duck" <Hprof> <comp2> N S NOM
"<and>"
       "and" KC
"<has>"
       "have" V PR 3S
"<got>"
       "get" <va+DIR> <vta+DIR> <vtk+ADJ> V IMPF
       "get" <va+DIR> <vta+DIR> <vtk+ADJ> V PCP2 AKT
       "get" <va+DIR> <vta+DIR> <vtk+ADJ> V PCP2 PAS
       "get" <va+DIR> <vta+DIR> <vtk+ADJ> V PCP2 STA
"<him>"
       "he" <masc> PERS MASC 3S ACC
"<living>"
       "living" <f-psych> <HH> <mon> <state> N S NOM
       "living" <jpert> <jppl> ADJ POS
       "live" <va+LOC> V PCP1
       "living" <pcp1> ADJ POS
       "living" <pcp1> N S NOM
"<here>"
       "here" <aloc> ADV
"<with>"
       "with" PRP
"<him>"
       "he" <masc> PERS MASC 3S ACC
"<$!>"
       "!" PU <<< @PU
```

He's adopted another blasted lame duck and has got him living here with him!

```
"<he_>"
       "he" <sam-> <*> <masc> PERS MASC 3S NOM
"<_s>"
;      "be" <-sam> V PR 3S
;      "we" <-sam> PERS 1P ACC
       "have" <-sam> V PR 3S
"<adopted>"
       "adopt" V PCP2 AKT
;      "adopt" V PCP2 PAS
;      "adopt" V PCP2 STA
;      "adopt" V IMPF
"<another>"
;      "another" INDP S NOM
       "another" DET S
"<blasted>"
       "blasted" ADJ POS
;      "blast" <v.contact> V PCP2 AKT
;      "blast" <v.contact> V PCP2 PAS
;      "blast" <v.contact> V PCP2 STA
;      "blast" <v.contact> V IMPF
"<lame>"
;      "lame" V PR -3S
;      "lame" V INF
;      "lame" V IMP
       "lame" <jsick> ADJ POS
;      "lame" <cc> N S NOM
"<duck>"
;      "duck" V PR -3S
;      "duck" V INF
;      "duck" V IMP
       "duck" <Hprof> <comp2> N S NOM
"<and>"
       "and" KC
"<has>"
       "have" V PR 3S
"<got>"
;      "get" <va+DIR> <vta+DIR> <vtk+ADJ> V IMPF
       "get" <va+DIR> <vta+DIR> <vtk+ADJ> V PCP2 AKT
;      "get" <va+DIR> <vta+DIR> <vtk+ADJ> V PCP2 PAS
;      "get" <va+DIR> <vta+DIR> <vtk+ADJ> V PCP2 STA
"<him>"
       "he" <masc> PERS MASC 3S ACC
"<living>"
;      "living" <f-psych> <HH> <mon> <state> N S NOM
;      "living" <jpert> <jppl> ADJ POS
       "live" <va+LOC> V PCP1
;      "living" <pcp1> ADJ POS
;      "living" <pcp1> N S NOM
"<here>"
       "here" <aloc> ADV
"<with>"
       "with" PRP
"<him>"
       "he" <masc> PERS MASC 3S ACC
"<$!>"
       "!" PU <<< @PU
```


Choose language
Language: Danish Dutch English Esperanto French German Italian Portuguese Spanish Greenlandic Bulgarian Croatian Maltese Arabic Russian (cohorts)
Write CG rule:
DELIMITERS = "<$.>" "<$!>" "<$?>" "<$\;>" "<$:>" "<$-->" "<$>" "<$start>" "<$START>" ;
MAPPING-PREFIX = @ ;
SETS
CORRECTIONS
MAPPINGS
CONSTRAINTS
Grammar
Or find a grammar to upload: Arakatu... Ez da fitxategirik hautatu.
Enter text to parse:
I saw her duck.
Text to parse
Or find a text to upload: Arakatu... Ez da fitxategirik hautatu.
Pre-run analysis: Morphological analyzer (all readings)
Add my grammar: Yes (raw text layout)
Run grammar on input
Reset

OR...


Language: Danish Dutch English Esperanto French German Italian Portuguese Spanish Greenlandic Bulgarian Croatian
Maltese Arabic Russian (cohorts)
Write CG rule:
DELIMITERS = "<$.>" "<$!>" "<$?>" "<$\;>" "<$:>" "<$-->" "<$>" "<$start>" "<$START>" ;
MAPPING-PREFIX = @ ;
SETS
CORRECTIONS
MAPPINGS
CONSTRAINTS
Grammar file
Or find a grammar to upload: Browse... gramatika_genus.rle
Enter text to parse:
File with text to parse
Or find a text to upload: Browse... lagizeEH400.txt
Pre-run analysis: Morphological analyzer (all readings)
Add my grammar: Yes (raw text layout)
Run grammar on input
Reset

He's adopted another blasted lame duck and has got him living here with him!

```
"<he_>"
  "he" <sam-> <*> <masc> PERS MASC 3S NOM
"<_s>"
  "be" <-sam> V PR 3S
  "we" <-sam> PERS 1P ACC
  "have" <-sam> V PR 3S
"<adopted>"
  "adopt" V PCP2 AKT
  "adopt" V PCP2 PAS
  "adopt" V PCP2 STA
  "adopt" V IMPF
"<another>"
  "another" INDP S NOM
  "another" DET S
[...]
```

Let's disambiguate this item!

```
"<_S>"
    "be" <-sam> V PR 3S
    "we" <-sam> PERS 1P ACC
    "have" <-sam> V PR 3S
```

```
DELIMITERS = "<$.>" "<$!>" "<$?>" "<$\;>" "<$:>" "<$-->" "<$>"
"<$start>" "<$START>" ;
MAPPING-PREFIX = @ ;

SETS

CORRECTIONS

MAPPINGS

CONSTRAINTS
REMOVE ("we") IF (0 ("<_s>")) (NOT -1 ("let")) ;
```

RULE

Operation — Target — Test/Condition 1 — Test/Condition 2

```
Enter text to parse: He's adopted another blasted lame duck and has
got him living here with him!
```

He's adopted another blasted lame duck and has got him living here with him!

OUTCOME

```
"<he_>"
  "he" <sam-> <*> <masc> PERS MASC 3S NOM
"<_s>"
  "be" <-sam> V PR 3S
; "we" <-sam> PERS 1P ACC REMOVE:5
  "have" <-sam> V PR 3S
"<adopted>"
  "adopt" V PCP2 AKT
  "adopt" V PCP2 PAS
  "adopt" V PCP2 STA
  "adopt" V IMPF
"<another>"
  "another" INDP S NOM
  "another" DET S
[...]
```

- the reading **removed** is marked with **;** at the beginning of the reading
- the **rule** applied (line) is added at the end of the reading

GRAMMAR FILE: SECTIONS

```
DELIMITERS = "<$.>" "<$!>" "<$?>" "<$\;>" "<$:>" "<$-->" "<$>" "<$start>" "<$START>" ;
MAPPING-PREFIX = @ ;

SETS

CORRECTIONS

MAPPINGS

CONSTRAINTS
```

DELIMITERS

- Obligatory first section.
- It lists the word forms that break the input into sentences.

SECTION: SETS (i)

It defines the sets that will be used in rules:

- LIST: it contains a list of tags (from the readings)
- SET: it contains combinations of previously defined SETS

Keyword set name list of tags end of list	
`LIST N = N ;`	The set N contains only the tag
`LIST A/N = A N ;`	The set contains tags A and N
`LIST SGPERSON = 1S 2S 3S ;`	The set contains tags 1S 2S and 3S
`LIST N-NOM-SG = (N NOM SG) ;`	The set contains one tag that is a sequence of three tags: N, NOM and SG (not necessarily contiguous)
`LIST AFTER = "after" "before" "since" ;`	The set contains three lemmas
`LIST AFTER-CS = ("after" CS) ("before" CS) ("since" CS) ;`	The set contains three tag sequences

SECTION: SETS (ii)

Keyword set name list of sets and/or tags end of list	
`SET AP = A/N OR PERSON ;`	The set is the union of existing sets **A/N** and **PERSON**
`SET AFTER-CS2 = AFTER + (CS) ;`	This set is equivalent to list **AFTER-C2**, constructed using the existing set **AFTER**

DISAMBIGUATION RULES: REMOVE & SELECT (i)

- REMOVE operation removes all readings containing the target tag if all the tests are satisfied. The last remaining reading, however, is not removed.
- SELECT operation removes all the readings not containing the target tag if all the tests are satisfied.

Operation target set optional test to be satisfied **REMOVE target IF (test1) (test2)… ;**	
REMOVE (N) IF (0 A) (-1C DET) (1C N) ; Non-declared tags ( ) Declared sets	This rule resolves the ambiguity between an adjective A and a noun N. The noun reading is removed if the given context is given.
SELECT (V SG2) IF (-1 PRONSG2) ;	This rule selects the reading having both tags V and SG2 if the condition is fulfilled. PRONSG2 must be previously declared.

DISAMBIGUATION RULES: REMOVE & SELECT (ii)

Disambiguating specific word forms

Word form Operation target set optional test to be satisfied	
`WORDFORM OPERATION target IF (test1)(test2)… ;`	
`"<voi>" SELECT (NEG) IF (-1 NEGV) ;`	For the running text word *voi*, select the reading with the tag (NEG) if the previous word has a tag belonging to the set NEGV.
`"<voi>" REMOVE (INS) ;`	This rule always removes the (INS) reading form the word form *voi*.

POSITIONS

```
0      =  the same cohort
1      =  the next cohort to the right
-1     =  the next cohort to the left
2      =  the second cohort to the right
*1     =  any cohort to the right
*-1    =  any cohort to the left
**1    =  any cohort to the right but the following LINKed test is evaluated
**-1   =  any cohort to the left but the following LINKed test is evaluated
```

Any position number can be followed by the letter C to mark *careful* checking:

```
1C   =  in the next cohort to the right, all the readings of the cohort have to belong to
        the given set
```

We can also refer to the absolute position of a cohort in the sentence.

```
@1     =  the first cohort to the right
@-1    =  the last cohort to the left
```

Operation with sets: UNION OF SETS

There should be only one contextual for every position, i.e. the tests `(1 A) (1 B)` are not legitimate in the same rule. For tests referring to the **same position**, we can UNION or CONCATENATIONS of sets, depending on the purpose.

UNION of SETS

`A OR N`	is a union of sets A and N
`A OR N OR ADV`	is a union of sets A, N and ADV
`A OR (N SG)`	is a set that contains the set A and the tag (N SG)
`(A SG) OR (N SG)`	is a set that contains two items: (A SG) and (N SG)

Example:

```
SETS

LIST Noun = N ;
LIST Adj = A ;

CONSTRAINTS

SELECT (PSS) IF
(1 Noun OR Adj)
(2 (N) OR Adj)
(3 (N) OR (A))
(4 Noun OR (A));
```

These four sets are equivalent

Operation with sets: CONCATENATION OF SETS

The concatenation operation makes a Cartesian product of two sets, i.e. it adds all possible combinations of the following tags to the preceding tags.

Example:

```
SETS

LIST A/N = A N ;
LIST CASE = INE ILL ELA ;

CONSTRAINTS
SELECT (PSS) IF
(1 A/N + CASE) ;
```

This set contains (A INE), (A ILL), (A ELA), (N INE), (N ILL) and (N ELA)

Operation with sets: DIFFERENCE OF SETS

Tags (or sets) can be removed from sets declared earlier.

Given the set HEAD:

```
LIST HEAD = @SUBJ @OBJ @COMP ;
```

The set `HEAD - (@SUBJ)` then contains the tags `@OBJ @COMP`

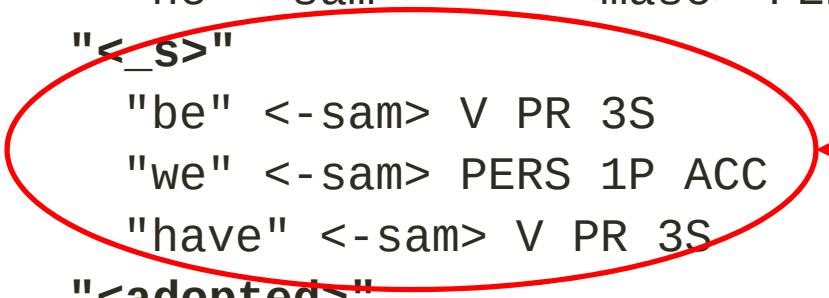
NEGATION OF TEST

Any contextual test can be negated. We can state that any reading of a cohort should NOT have the given reading

`(NOT -1 A)`	The previous cohort does not contain a reading which has a tag belonging to the set A.
`(NOT 1 A)`	The following cohort does not contain a reading which has a tag belonging to the set A.
`(NOT 0 A)`	The target cohort does not contain a reading which has a tag belonging to the set A.
`(NOT @1 AUX)`	The first cohort of the sentence does not contain a reading which has a tag belonging to the set AUX.
`(NOT 1 A OR B)`	This is equivalent to `(NOT 1 A) (NOT 1 B)`*. Neither of them of them should appear in the following cohort.
`(NOT 1 A + B)`	The following cohort does not contain a reading which has a tag belonging to the set A and in the same reding a tag belonging to the set B.
`(NOT 1 A) (1 B)`	Two tests may refer to the same position only if one of them is a NOT test. he following cohort does not contain a reading which has a tag belonging to the set A and in the same reding a tag belonging to the set B.

He's adopted another blasted lame duck and has got him living here with him!

```
"<he_>"
  "he" <sam-> <*> <masc> PERS MASC 3S NOM
"<_s>"
  "be" <-sam> V PR 3S
  "we" <-sam> PERS 1P ACC
  "have" <-sam> V PR 3S
"<adopted>"
  "adopt" V PCP2 AKT
  "adopt" V PCP2 PAS
  "adopt" V PCP2 STA
  "adopt" V IMPF
"<another>"
  "another" INDP S NOM
  "another" DET S
[...]
```



Let's disambiguate this item!

```
"<_S>"
    "be" <-sam> V PR 3S
    "we" <-sam> PERS 1P ACC
    "have" <-sam> V PR 3S
```

```
# 1st option. REMOVE “we” interp. If not “let’s”
REMOVE ("we") IF (0 ("<_s>")) (NOT -1 ("let")) ;

# 2nd option. REMOVE “we” interp. if not “let’s”
"<_s>" REMOVE ("we") IF (NOT -1 ("let")) ;

# 3rd option. Same as previous, declaring some sets.
LIST LET = "let" ;
LIST PERPRO = PERS ;

REMOVE PERPRO IF (0 ("<_s>")) (NOT -1 LET) ;

# 4th option. The previous operation refers to the lemma “let”. It should
refer to word form “<let>”.
LIST LET = “<let>" ;
LIST PERPRO = PERS ;

REMOVE PERPRO IF (0 ("<_s>")) (NOT -1 LET) ;

# [...]
```

He's adopted another blasted lame duck and has got him living here with him!

```
"<he_>"
  "he" <sam-> <*> <masc> PERS MASC 3S NOM
"<_s>"
  "be" <-sam> V PR 3S
; "we" <-sam> PERS 1P ACC REMOVE:5
  "have" <-sam> V PR 3S
"<adopted>"
  "adopt" V PCP2 AKT
  "adopt" V PCP2 PAS
  "adopt" V PCP2 STA
  "adopt" V IMPF
"<another>"
  "another" INDP S NOM
  "another" DET S
[...]
```

- the reading **removed** is marked with **;** at the beginning of the reading
- the **rule** applied (line) is added at the end of the reading

SCANNING

It is possible to scan for a tag when its exact position is not known.

`(*-1 A)`	There is a cohort to the left that contains a reading which has a tag belonging to the set A.
`(*1 A)`	There is a cohort to the right that contains a reading which has a tag belonging to the set A.
`(*2 A)`	Two or more cohorts to the right, there is a cohort that contains a reading which has a tag belonging to the set A.
`(NOT *-1 A)`	There is no cohort to the left that contains a reading which has a tag belonging to the set A.
`(NOT *1 A)`	There is no cohort to the right that contains a reading which has a tag belonging to the set A.
`(NOT *-3 A)`	There is no cohort three or more cohorts to the left containing a reading which has a tag belonging to the set A. Cohorts in position -1 and -2 may have such a tag.
`(*-1C A)`	There is a cohort to the left that contains a reading which has a tag belonging to the set A. The first such cohort must have a tag belonging to the set A in all its readings.

BARRIERS

BARRIERS are used to restrict scanning.

`(*-1 DET BARRIER VERB OR ADV)`	To the left, there is a cohort with a tag belonging to the set DET. The cohort with a tag belonging to the barrier set (here VERB OR ADV) is the last word that is scanned.
`(NOT *-1 VERB BARRIER DET)`	To the left, there is no tag belonging to the set VERB before (or in) the first cohort that has a tag belonging to the barrier set DET.

LINKING

LINKING establishes further tests on the cohort which is found by scanning.
The scanning stops at the first match of the given set.

`(*-1 DET LINK 1 N)`	The next cohort to the left which has a tag belonging to the set DET is immediately followed by a cohort with a tag belonging to the set N.
`(*-1 DET LINK 1 A LINK 1 N)`	The next cohort to the left which has a tag belonging to the set DET is followed by a cohort with a tag belonging to the set A. That is followed by a cohort with a ag belonging to the set N.
`(*-1 DET BARRIER VERB LINK 1 N)`	The next cohort to the left which has a tag belonging to the set DET is followed by a cohort with a tag belonging to the set N. There must not be a tag from the set VERB before the occurrence of the set DET.
`(*1 NTH LINK *1 NOR)`	To the right, there is cohort with a tag belonging to the set NTH and to the right from it a cohort with a tag belonging to the set NOR.
`(*-1 A LINK *1 B BARRIER C)`	The first A to the right is followed by B somewhere to the right. Thee is no C between A and B.
`(*@1 A LINK 1 B)`	The first occurrence of A in the sentence is followed by B.

CONTINUOUS SEARCH

Scanning stops at the first cohort having a tag belonging to the set being checked, and if the linked test fails, the whole contextual test fails. For linking, there is an special operator (**) that continuous the search if the linked tests fail.

`(**-1 DET LINK 1 N)`	To the left, there is a cohort with a tag belonging to the set DET is immediately followed by a cohort with a tag belonging to the set N.
`(NOT **-1 DET LINK 1 N)`	To the left, there is not a cohort with a tag belonging to the set DET is immediately followed by a cohort with a tag belonging to the set N. Here the negation is applied last.
`(**1C A LINK 1 B)`	In *careful* mode, scanning stops at the first occurrence of A where the linked tests holds, i.e. the next occurrence of A followed by B is unambiguously A.

Practical exercise 1