# On how to write rules in Constraint Grammar (CG-3)

**Eckhard Bick**

University of Southern Denmark
VISL Project, ISK
GrammarSoft / GramTrans

# Constraint Grammar – what is it?

- (1) a methodological paradigm for handling token-linked information in a contextual, rule-based fashion (Karlsson 1990, 1995)
- (2) a descriptive convention within the dependency camp, supporting a lexical approach with a clear form-function distinction
- reductionist, focus on disambiguation, robust, fast, “non-chomskyan” ..
- (A) a formal language to express context grammars
- (B) a number of specific compiler implementations to support different dialects of this formal language

# Token-based tags

| | | | | |
|---|---|---|---|---|
| O | <artd> | DET M S | @>N | #1->3 |
| último | | ADJ M S | @>N | #2->3 |
| diagnóstico | | N M S | @SUBJ> | #3->9 |
| elaborado | | V PCP2 M S | @ICL-N< | #4->3 |
| por | | PRP | @<PASS | #5->4 |
| a | <artd> | DET F S | @>N | #6->7 |
| Comissão=Nacional | | PROP F S | @P< | #7->5 |
| não | | ADV | @ADVL> | #8->9 |
| deixa | | V PR 3S | @FMV | #9->0 |
| dúvidas | | N F P | @<ACC | #10->9 |
| $. | | | | #11->0 |

# What is CG used for?

Lingsoft LANGUAGE SOLUTIONS

**Lingsoft Proofreader Finnish for Sisulizer 2008**
99,00 € + VAT 23 %
Add to cart >
More Sisulizer versions...

**Lingsoft Svefix 2.2 for Microsoft Office**
50,00 € + VAT 23 %
Evaluate Add to cart >

**Lingsoft Speech Controller 1.4 (Windows)**
220,49 € + VAT 23 %
Evaluate Add to cart >

VISL grammar games:

connexor Natural Knowledge

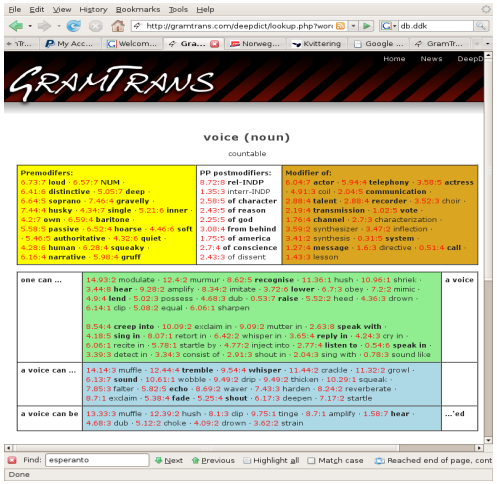
Machinese parsers
News feed and relevance filtering
Opinion mining in blogs
Science publication monitoring

GRAMTRANS

OrdRet

QA

Machine translation
Spell- and Grammar checking
Corpus annotation
Relational dictionaries:
DeepDict




NER

Annotated corpora:
CorpusEye

# CG input

- Preprocessing
  - Tokenizer:
    - **Word-splitting**: *punctuation vs. abbreviation?, won't, he's vs. Peter's*
    - **Word-fusion:** *Abdul=bin=Hamad, instead=of*
  - Sentence separation: <s>...</s> markup vs. CG delimiters
- Morphological Analyzer
  - outputs cohorts of morphological reading lines
  - needs a lexicon and/or morphological rules

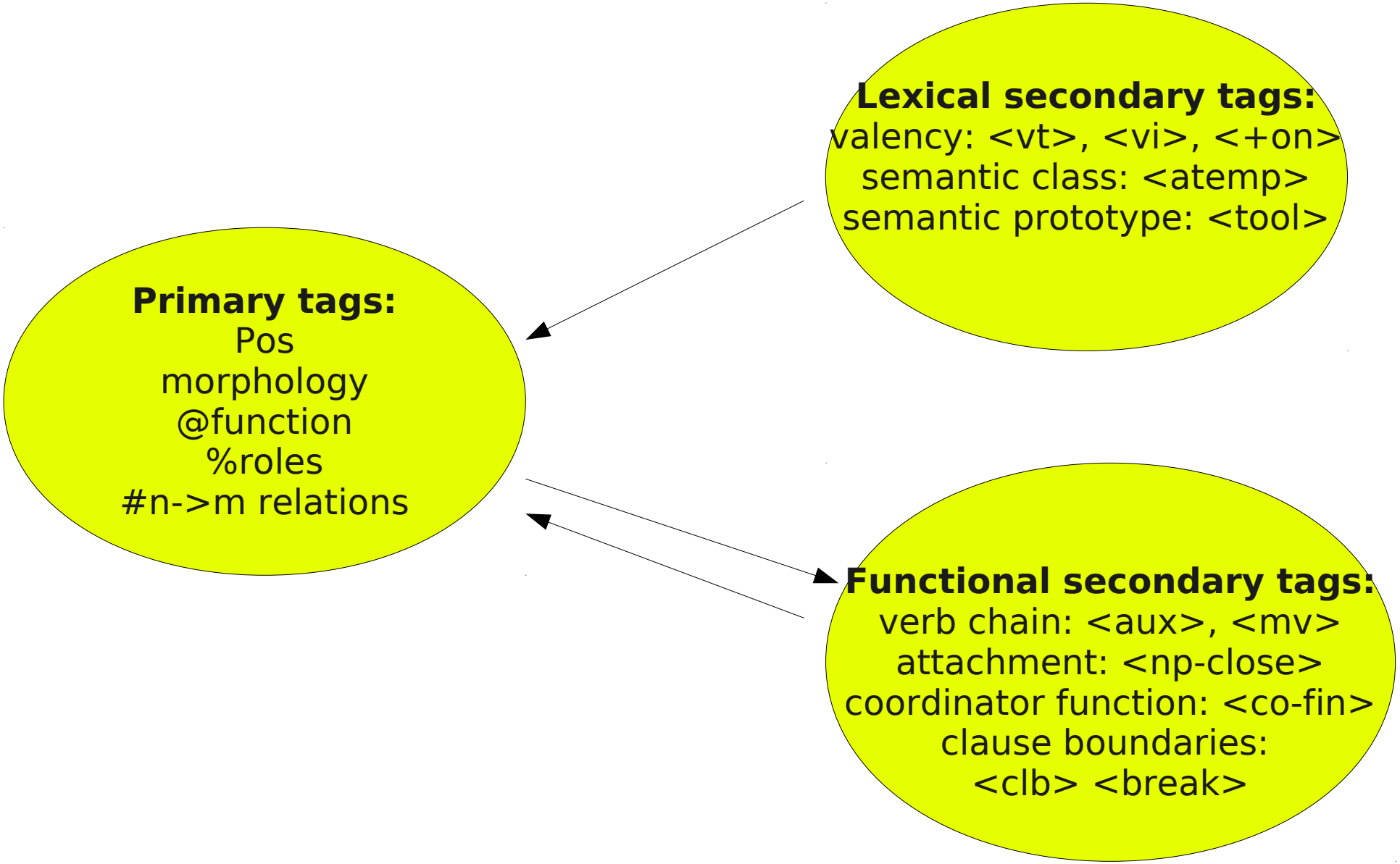
# CG rules

- rules add, remove or select morphological, syntactic, semantic or other readings
- rules use context conditions of arbitrary distance and complexity (i.e. other words and tags in the sentence)
- rules are applied in a deterministic and sequential way, so removed information can't be recovered (though I t can be traced). Robust because:
  - rules in batches, usually safe rules first
  - last remaining reading can't be removed
  - will assign readings even to very unconventional language input (“non-chomskyan”)

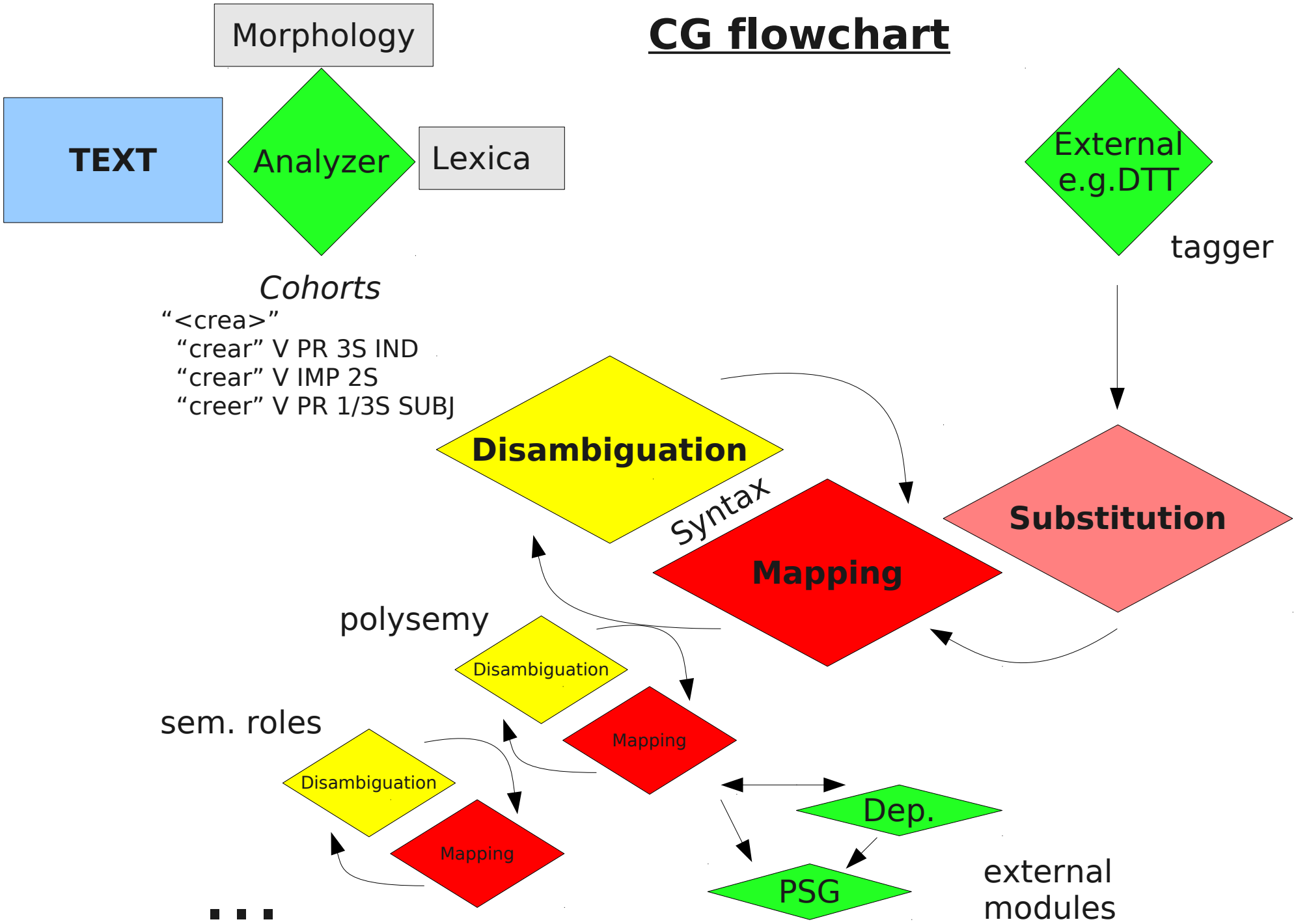
# some simple rule examples

- REMOVE VFIN
  IF (*-1C VFIN BARRIER CLB OR KC)
  *exploits the uniqueness principle: only one finite verb per clause*
- MAP (@SUBJ> @<SUBJ @<SC) TARGET (PROP)
  IF (NOT -1 PRP)
  *syntactic potential of proper nouns*
- SELECT (@SUBJ>)
  IF (*-1 >>> OR KS BARRIER NON-PRE-N/ADV)
  (*1 VFIN BARRIER NON-ATTR)
  *clause-initial np's, followed by a finite verb, are likely to be subjects*

# primary vs. secondary tags



# CG flowchart



# CG languages (VISL/GS)

| | Parser | Lexicon | Analyzer | Grammar | Levels |
|---|---|---|---|---|---|
| da | DanGram | 100.000 val/sem, 40.000 names | Full | 8.000 rules | morph., syntax, dep., psgMT roles, frames, gramarchecker |
| pt | PALAVRAS | 70.000 val/sem, 15.000 names | Full | 7.500 rules | morph., syntax, dep., psg, roles, anaphora |
| es | HISPAL | 73.000 lexemes | Full | 4.500 rules | morph., syntax, dep., psg |
| en | EngCG | 160.000 sem | Full | 4.400 rules | morph., syntax, dep., psg roles, frames, chunks,MT |
| fr | FrAG | 57.000 lexemes | Full | 2.000 rules | morph., syntax, dep., psg |
| de | GerGram | 25.000 val/sem | Full | 2.100 rules | morph, syn, dep, psg,chunk,MT |
| eo | EspGram | 30.000 val/sem | Full | 2.600 rules | morph, syntax, dep.gr.check |
| it | ItaGram | 30.600 lexemes | Full | 3.300 rules | morph., syntax, dep. |
| se | SveGram | 63.000 val/sem | Full | 8.400 | morph., syntax, dep., MT |
| no | NorGram | 77.000 val/sem | Full | adapt. da/OBT | morph., syntax, dep., MT |
| nl | NedGram | 58.000 lexemes | Full | 1.960 | morph.,syntax,dep. |

# VISL languages (others)

- Basque
- Catalan
- English ENGCG (CG-1, CG-2, FDG)
- Estonian (local)
- Finnish (CG-1?)
- Irish (Vislcg)
- Norwegian (CG-1)
- Sami (CG-3)
- Swedish (CG1, CG-2?)
- Swahili (Vislcg)

# Apertium “incubator” CGs

(https://apertium.svn.sourceforge.net/svnroot/apertium/...)

- Turkish
  - .../incubator/apertium-tr-az/apertium-tr-az.tr-az.rlx
- Serbo-Croatian
  - .../incubator/apertium-sh-mk/apertium-sh-mk.sh-mk.rlx
- Icelandic
  - .../trunk/apertium-is-en/apertium-is-en.is-en.rlx
- Breton
  - .../trunk/apertium-br-fr/apertium-br-fr.br-fr.rlx
- Welsh
  - .../trunk/apertium-cy-en/apertium-cy-en.cy-en.rlx
- Macedonian
  - .../trunk/apertium-mk-bg/apertium-mk-bg.mk-bg.rlx
- Russian
  - .../incubator/apertium-kv-ru/apertium-kv-ru.ru-kv.rlx

# Performance and uses

- Published performance for system-internal evaluations is astonishingly high across languages, with F-scores for mature systems around
- 99% for POS
- 95% for syntactic function (shallow dependency)
- Relative performance in open joint evaluation:
  - e.g. HAREM (Portuguese NER & classification)
- Supports a wide variety of applications
  - Grammar checking (Norwegian, Swedish, Danish ...), e.g. OrdRet (better at weighting suggestions than Word)
  - Corpus annotation (e.g. treebanks) and teaching
  - IR, NER and QA
  - MT and other semantic stuff
  - Anaphora resolution

# Some history and comparisons: CG “dialects”

- Common to all CG systems:
  - the context-dependent manipulation of tag-encoded linguistic information at the token level (formally, akin to regular expression substitutions)
  - implemented as REMOVE, SELECT, MAP, ADD, REPLACE, SUBSTITUTE ...
- Differences at the implementational level:
  - programming language: Lisp, C/C++, finite state
  - speed, e.g. cg2 (Tapanainen 1996) = 6 x vislcg (Martin Carlsen)
  - proprietory (cg1, fdg/conexor), academic (cg2), project-bound (Müürisep 2005), commercial (FDG conexor.com), open source (vislcg, cg3)
  - cross compiler compatibility?
    [cg1] <-> [cg2 > vislcg > cg3]

# Differences at the Grammar level

- Differences in expressive power
  - scope: global context (standard, most systems) vs. local context (Lager's templates, Padró's local rules, Freeling ...)
  - templates, implicit vs. explicit barriers, sets in targets or not, replace (cg2: reading lines) vs. substitute (vislcg: individual tags)
  - topological vs. relational
- Differences of applicational focus
  - focus on disambiguation: classical morphological CG
  - focus on selection: e.g valency instantiation
  - focus on mapping: e.g. grammar checkers, dependency relations
  - focus on substitutions: e.g. morphological feature propagation, correction of probabilistic modules

# The CG3 project

- 6+ year project (University of Southern Denmark & GrammarSoft)
- some external or indirect funding (Nordic Council of Ministries, ESF) or external contributions (e.g. Apertium)
- programmer: Tino Didriksen
- design: Eckhard Bick (+ user wish list, PaNoLa, ...)
- open source, but can compile "non-open", commercial binary grammars (e.g. OrdRet)
- goals: implement a wishlist of features accumulated over the years, and do so in an open source environment
- enabling hybridisation of methodologies: CG, dependency grammar, probabilistic methods, ...
- support for specific tasks: MT, spell checking, anaphora ...

# The CG3 project -2

- working version downloadable at http://beta.visl.sdu.dk
- compiles on linux, windows, mac
- speed: equals vislcg in spite of the new complex features, faster for mapping rules, but still considerably slower than Tapanainen's cg2 (working on it).
- documentation available online
- sandbox for designing small grammars on top of existing parsers: cg lab and IDE

# A rules file 1 (definitions)

**DELIMITERS** = "<.> "<!>" "<?>" ; # sentence window

**SETS**

LIST NOMINAL = N PROP ADJ PCP ; # nominals, i.e. potentieal nominal heads
LIST PRE-N = DET ADJ PCP ; # prenominals
LIST P = P S/P ; # plural
SET PRE-N-P = PRE-N + P ; # plural prenominals, equivalent to (DET P) (DET S/P) (ADJ P) (ADJ S/P) (PCP P) (PCP S/P)
LIST CLB = "<,>" KS (ADV <rel>) (ADV <interr>) ; # clause boundaries
LIST ALL = N PROP ADJ DET PERS SPEC ADV V PRP KS KC IN ; # all word classes
LIST V-SPEAK = "dizer" "falar" "propor" ; # speech verbs
LIST @MV = @FMV @IMV ; # main verbs

# A rules file 2 (morphological disambiguation)

**CONSTRAINTS**

REMOVE (N S) IF (-1C PRE-N-P) ; # remove a singular noun reading if there is a safe plural prenominal directly to the left.

REMOVE NOMINAL IF (NOT 0 P) (-1C (DET) + P) ; # remove a nominal if it isn't plural but preceded by a safe plural determiner.

REMOVE (VFIN) IF (*1 VFIN BARRIER CLB OR (KC) LINK *1 VFIN BARRIER CLB OR (KC)) ; # remove a finite verb reading if there are to more finite verbs to the right none of them barred by a clause boundary (CLB) and coordinating conjunction (KC).

# A rules file 3
# (syntactic disambiguation)

**MAPPINGS**

MAP (@SUBJ> @ACC>) TARGET (PROP) IF (*1C VFIN BARRIER ALL -(ADV)) (NOT -1 PROP OR PRP) (NOT *-1 VFIN) ; # a proper noun can be either forward subject or forward direct object, if there follows a finite verb to the right with nothing but adverbs in between, provided there is no proper noun or preposition directly to the left, and a finite verb anywhere to the left.

**CONSTRAINTS**

REMOVE (@SUBJ>) IF (*1 @MV BARRIER CLB LINK *1C @<SUBJ BARRIER @MV) ; # remove a forward subject if there is a safe backward subject to the right with only one main verb in between

# CG Contexts

- **Context conditon**: word form “<...>”, base form “....”, tag A-Z, <[a-z]> @[A-Z], combinations ...
- direction: + (right), - (left)
- Position marker:
  - 0 self
  - local right: 1, 2, 3 ..., local left: -1, -2, -3, ...
- Globality
  - * continue until match is found
  - ** continue also across context match to fulfil further (linked) conditions
  - 0* nearest neighbour: search in both directions
- Careful: C, e.g. *1C (only unambiguous readings)

# CG contexts 2

- **NOT**: conditions can be negated
  - (NOT *1 VFIN)
- contexts can be **LINK**ed
  - (*1C xxx LINK 0 yyy LINK *1 zzz)
- searches can have a **BARRIER** or **CBARRIER**
  - (*1 N BARRIER VFIN)
- contexts can be ANDed
  - IF (0 xxx) (*1 yyy) (NOT *-1 zzz)
- **NEGATE**: for negating entire context chains
  - (NEGATE *1 ART LINK 1 ADJ LINK 1 N)
- **NONE**: for negating dependencies or relations
  - (NONE c @ACC) (NONE r:referent HUM)

# Mapping (MAP, ADD)

MAP (@SUBJ) TARGET (N) IF (NOT *-1 NON-PRE-N)
MAP (@SUBJ) (N) (NOT *-1 NON-PRE-N)

- Usually as a special section (MAPPING or BEFORE-SECTIONS), but in cg3 allowed anywhere
- Strictly ordered
- Both MAP and ADD can be used to add tags, but:
  - MAP "closes" a line for further mapping (but not SUBSTITUTE!) even if the mapped tag(s) does not contain the flagged prefix (default @)
  - ADD maps, but allows further mapping
- MAPed tags can be "seen" by later mapping rules, even in the same section

# Substitutions

SUBSTITUTE (X) (Y) TARGET (...) IF (...)

- Replaces a tag or tag chain with another, useful for:
  - correcting input from other modules, e.g. probabilistic taggers
    SUBSTITUTE (KS) (<rel> INDP) TARGET ("that") (*1C VFIN BARRIER NON-ADV) (*-1C N BARRIER NON-ADV)
  - inserting, changing and removing secondary tags
    SUBSTITUTE (N) (<def> N) TARGET N IF (c ART-DEF OR DET-DEF)
  - correcting lower level CG once higher lever information is available
  - spell or grammar checkers
    SUBSTITUTE (UTR) (NEU) TARGET (@<SC) IF (*-1C @SUBJ> + NEU)
- Usually as a special section (CORRECTIONS or BEFORE-SECTIONS), but in cg3 allowed anywhere
- Strictly ordered
- SUBSTITUTE does not "close" a line for mapping
- SUBSTITUTEd tags can be "seen" by later SUBSTITUTE or Mapping rules, even in the same section

# Regular expressions

- CG3 allows ordinary regular expressions in strings
- reg.ex. can be used in strings adding /r, "/ or >r
  - can be used for any tag
    <sem.*>r, /\@<?ADVL?>?$/r, ".+ize"r
  - /i means case-insensitive: "<.+ist>"i
  - use \ if a meta character doesn't work: \\1, <on\\^.*>r
- on the fly sets
  - ".*i[zs]e"r --- transitive verbs candidates in English
  - <[HA].*>r --- semantic prototype tag for *animates,*
    *i.e. humans* (e.g. <*Hprof*>) and *animals* (e.g. <*Aorn*>)

REMOVE @<ACC (0 @<SUBJ LINK 0 (<H.*>r) OR (".*ist"r)
-> discard object in favor of subjects if the token is +HUM

LIST <place> = <L.*>r ”(North|South|West|East).*”r ;
--> defines place-category on the fly

# Variables

- variables can occur in strings marked /v or prefixed VSTR:
  - $1 ... $10 sequentially match parentheses in /r strings,
  - can be used in "<wordforms>", "baseforms" and <secondary_tags>, but not in MORPH or @SYN tags
  - variables can be upper- or lower-cased on a first-letter or whole-string bases by prefixing %u, %U, %l, %L "%L$1"v, VSTR:"%l$1ly"v
  - variables can contain unified $$ or && sets in {} brackets

```
MAP KEEPORDER (VSTR:§$1) TARGET @SUBJ
    (*p V LINK -1 (*) LINK *1 (<r:.*>r) BARRIER <mv>
    LINK 0 PAS LINK 0 (<r:ACC:\(.*\)>r)) ; # raising function-conditioned
    semantic role information from framnet tags on main verbs
```

# APPEND: CG-input on the fly

- APPEND rules
  - **closed word classes:**
    APPEND ("$1"v <safe> <atemp> ADV)
    TARGET ("<(always|ever|never|now|today|tomorrow)>"r)
  - **open word classes**
    APPEND ("$1"v <safe> ADJ)
    TARGET ("<(.*(ic|oid|ous))>"r)
  - **inflexion**
    APPEND ("$1y"v <heur> N P NOM) TARGET ("<(.*)ies>"r) APPEND ("$1"v <safe> V PAST) TARGET ("<(.*?(.))\\2ed>"r)
  - **default**
    APPEND ("$1"v <heur> <default> N S NOM)
    TARGET ("<([a-z].*)>"r) (NOT 0 <lex> OR (N)) ;

# Creating Dependencies 1

SETPARENT (@>N) (0 (ART DET))  TO (*1 (N)) ;
SETPARENT (@P<) TO (*-1 (PRP)) ; #will attach several (coordinated) @P< arguments to the same preposition
SETCHILD (PRP) TO (*1 @P< OR @ICL-P<) ; #will attach only one (the first) possible argument to the preposition
SETPARENT (@FS-N<) TO (*-1 N LINK NOT p _TARGET_)

- used to create dependencies on the fly
- used to change existing dependencies
- either for full trees (treebanks) or for subtrees (e.g. np) to support other tasks (such as grammar checking or feature propagation)
- circularity
  - a rule won't be applied if it introduces circularity
  - however, if there IS circularity further up in the ancestor chain from a previous module, then it will be accepted

# Creating Dependencies 2

- two separate context fields: context can apply either to the SETfrom token (before TO) or the SETto token (at end of rule
- default: last-context attachment, otherwise: **A**
  - SETPARENT @ICL-FUNC>
    (**NONE p** (*))
    TO (****1A** <mv> LINK **NEGATE** *-1 KC BARRIER NON-V/ADV)
    (**NONE pS** @FS-N< OR @FS-P< OR @ICL-N< OR @ICL-P<) ;

# Using Dependencies

SELECT (%hum) (0 @SUBJ) (p <Vcog>)

-> assign +HUM to subjects of cognitive verbs

SELECT (@ACC) (NOT s @ACC)

-> uniqueness principle

(*-1 N LINK c DEF)

-> definite np recognized through dependent

ADD (§AG) TARGET @SUBJ (p V-HUM LINK c @ACC LINK 0 N-NON-HUM) ;

- accepts input from other programs in cg-format: ... #n->m
- in a rule, dep-relations (letters) replace positions (numbers), */** behaves “correspondingly”
  - Parent/Mother (p), Child/Daughter (c), Sibling/Sister (s)
  - Self as relation: S, Self as context: _TARGET_
- NOT/C refer to the context, use NONE/ALL for the relations

# Labelled arcs for other purposes

- instead of the default dependency arcs, other relations can be defined:
- SETRELATION (identity) TARGET (<rel>) TO (*-1 N) ;
  (Set a ***"identity"*** *relaton from a relative pronoun to a noun occurring earlier in the sentence.)*
- results in: ID:n R:identity:m
  - n: arc base (here pronoun) word number
  - identity: relation name introduced by R
  - m: arc head (here the referent noun) word number
- REMRELATION – removes one direction of a relation
  - REMRELATION (name) targetset () TO ()
- SETRELATIONS and REMRELATIONS simultaneously handle 2 names for the two directions of a relation

# Chunking: Grouping dependents

- Ordinary dependency trees ignore token order
- Some tasks, such as chunking, field grammar or syntactic movements, need token order
- Making dependencies (and relations) order-sensitive:
  - l (left of head), r (right of head): e.g. (lc ADJ) vs. (rc ADJ)
  - ll (leftmost dependent), rr (rightmost dependent)
    e.g. (llcS @>N) or (rrcS @N<) for np chunking
  - use cc (descendents) instead of *c to address descendents as a set rather than successively

**ADDRELATIONS** (np-head-l) (np-start) TARGET (*)
(c @>N OR @N<&)
TO (**llScc** (*)) ; # left edge of np

**ADDRELATIONS** (np-head-r) (np-stop) TARGET (*)
(c @>N OR @N<&) (r:np-head-l (*))
TO (**rrScc** (*)) ; # right edge of np

# Spanning Window Boundaries

(*1> ("http.*")) ; # Recognizing a reference section: Find urls
(*-1< UTR + @SUBJ BARRIER CLB) ; # Anaphora: Pronoun gender resolution
(*0W (<Vground>) ; # Domain: Text about cars

- Span Left (<): allows to span left boundaries
- Span Right (>): allows to span right boundaries
- Span Both (W): allows to span boundaries right and left
- Default ± 2 windows, otherwise
  - command line flag: --num-windows 5
- Always allowing **all** spans to cross boundaries
  - command line flag: --always-span

# Probabilistic / statistical tags

REMOVE (<Conf<5>)

-> confidence threshold 5 (%)

REMOVE (<Noun<=10>) (NOT -1 PRE-N)

-> context dependent frequency threshold 10%

SELECT (<W=MAX>), REMOVE (<W=MIN>)

-> select the highest value for W, or remove the lowest

- expects input tags with colon-separated numerical values:
  - <Conf:80> (confidence values, e.g. for suggestions of a spell checker
  - <Verb:70> (e.g. monogram PoS-likelihod for a given token)
- all positive integer values are possible, a cohort sum of 100% for confidence is an optional convention, as is the use of relative frequencies

# List Unification

- LIST labels can be defined and unified as variables by prefixing $$ (SET'ed lists will be OR'ed into a joined list)
- in a CG rule, all occurrences of the $$ set will be unified to mean the same set member
  - LIST ROLE = %AG %PAT %TH %LOC ;
  - SELECT $$ROLE (-1 KC) (-2C $$ROLE) ; # (4-in-1 rule)
- the $$ occurrence in the target position is the primary one (i.e. the one the others unify with)
- if $$ only is used in contexts, add KEEPORDER to force a safe interpretation of the first occurrence as the primary one:
  - REMOVE KEEPORDER (ADJ @N<)
    (NEGATE 0 $$CASE LINK -1 N + $$CASE)

# Set Unification

- SET labels can be defined and unified as variables by prefixing &&.
- Unlike list unification, set unification does not unify list members ("terminal" set members). Instead, it unifies subsets belonging to a superset. Two contexts will set-unify if they have tags sharing the same subset.

LIST N-SEM = <sem> <sem-l> <sem-r> <sem-w> <sem-c> <sem-s> <sem-e> <coll-sem> <sem-nons> <system> <system-h> ;
(***not:*** LIST N-SEM = <sem.*>r <system.*>r ;)

SET N-SEMS = N-HUM OR N-LOC ... OR N-SEM ... OR N-SUBSTANCE ;

REMOVE @SUBJ>
(0 $$@<ARG LINK 0 &&N-SEMS)
(*-1 KC BARRIER NON-PRE-N/ADV
LINK *-1C $$@<ARG BARRIER CLB-ORD OR &MV OR @ARG/ADVL>
LINK 0 &&N-SEMS) ; *# offered the reader detailed notes and* ***instructions*** *on most of the prayers*

# Templates

- Labels for complex contexts conditions, which – once defined – can then be used by many different rules, or even in other templates.

- (a) Templates can be in the form of **generative constituent templates**, with a dummy 0 or ? position

  - TEMPLATE np = (? ART LINK 1 N) OR (? ART LINK 1 ADJ LINK 1 N) referenced as **(*1 VFIN LINK *1 T:np)** or **(-1 T:np)**

  - note that the final instantiated position from a template is the one "seen" from the outside, so **(-1 T:np)** needs an N left of the target, and LINK will proceed from N

  - Templates can recurse within one definition, but not across definitions

    TEMPLATE **pp** = 0 PRP LINK *1 N/PROP BARRIER NON-PRE-N ;
    ADD (@<SUBJ) TARGET ART-IDF (-1X **T:pp** LINK *-1x ("be") OR ("appear") OR <ve> BARRIER NON-ADV LINK -1 there) ; # there appeared **among them** a prophet that ...

- (b) Templates can be **context shorthand** for CG code

  TEMPLATE v-hum = (c @SUBJ + HUM) OR (*1 (”that” KS) BARRIER V)
  ADD (§TH) TARGET @ACC (p V LINK T:v-hum) ;

# Runtime options

- --grammar, -g ... the grammar file to use for the run
- --vislcg-compat, -p ... compatible with older VISLCG
- --trace ... adds debug output
- --prefix ... sets mapping prefix, default @
- --sections ... number of sections to run, default all
- --single-run ... only runs each section once.
- --no-mappings ... disables MAP, ADD and REPLACE rules.
- --no-corrections ... disables SUBSTITUTE and APPEND
- --num-windows ... window buffer span, default ±2
- --always-span ... always scan across window boundaries.
- --soft-limit ... token limit for SOFT-DELIMITERS (def. 300)
- --hard-limit ... token limit for hard window breaks (500)

# CG input

- (1) Specific analytical analyzer program (e.g. most VISL grammars)
  - needs rules for inflexion *(close - closed - closes - closing)*, compounding, affixation *(-ify, -ation, over-),* gemination *(put - putting)*
  - needs a lexicon of acceptable base forms for these processes (lexemes og lemmata)
  - needs a list of full form exceptions - unless the language is completely regular (Esperanto)
  - advantages:
    - can achieve very good coverage
    - very malleable: easy to integrate exeptions or external lists
  - disadvantages:
    - needs a linguist, some creativity and lexical resources (time/labour/money-expensive)

# CG input

- (2) Finite State Transducer (e.g. Kimmo Koskenniemi, Xerox, ...)
  - chains continuation lexica (for roots, prefixes, suffixes, inflexion endings)
  - combinatorial rules (before/after lexicon type conditions)
  - possibly edge letter rules or two-level rules for orthographical variation (gemination, umlaut etc)
  - advantages:
    - very fast
    - can be used for both analysis and generation
  - disadvantages:
    - needs a (quite specialized) linguist, and lexical resources (time/labour/money-expensive)
    - no natural way of doing heuristics

- (3) Full form lists
  - Look-up method: outputs one or more analysis strings for each governed token
  - can be built from (1) given a good wordlist, or from (2) given a good lemma list
  - can be built from any annotated corpus, will have perfect coverage for that corpus, but not necessarily for the language as such
  - advantages:
    - fast: database lookup, simple: no algorithm involved
    - cheap and easy to make: does not need a linguist, exploits (where available) linguistic corpus annotation labour
  - disadvantages:
    - depending on the language, bad coverage (heavily inflecting, agglutinating, compounding, polysynthetic languages ... i.e. most languages but English and Chinese)
    - once made, it is difficult to adapt in systematic ways

# CG input

- (4) Statistical tagger: Brill-tagger, DTT, ...
  - cheap solution, IF the language in question has training data
  - depends heavily on the training data
    - lexical gaps?
    - missing morphology? (i.e. just PoS)
    - black box with only one reading surviving, can't be improved, amended or repaired
  - solution: post-processing with CG rules
    - SUBSTITUTE (PRON) (CONJ) TARGET (”that”) IF (*-1 <Vcog> OR <speak> BARRIER VV)
    - if necessary, add more morphology (SUBSTITUTE, ADD)
    - if necessary, add new reading lines (APPEND) for CG disambiguation

# What now?

- check the details, get ideas: ***visl.sdu.dk***
  - CG analyses for 11 languages, tools, applications, cg-lab, links ....
- remember CG is modular - if you lack ressources, consider using other people's - even if a ressource is closed, it could be used as a ”black box”, through an API or through output samples for development
- if you decide to use CG in a project, let us know what you are working on, join the network:
  - ***constraint-grammar@googlegroups.com***
- visit us in Odense or Tromsø, we can always improvise a workshop ...