Constraint Grammar Manual

3rd version of the CG formalism variant

Mr. Tino Didriksen, GrammarSoft ApS `<tino@didriksen.cc>`

Constraint Grammar Manual: 3rd version of the CG formalism variant

by Mr. Tino Didriksen

Table of Contents

List of Tables

Chapter 1. Intro

Caveat Emptor

This manual should be regarded as a guideline. Some of the features or functionality described is either not implemented yet, or does not work exactly as advertised. A good place to see what is and is not implemented is to run the regression test suite as the test names are indicative of features. The individual tests are also good starting points to find examples on how a specific feature works.

What this is...

This document describes the design, features, implementation, usability, etc of an evolution in the constraint grammar formalism.

Naming

I have called this version of CG "VISL CG-3" since it implements the functionality of the VISL CG-2 variant of constraint grammar and is a general update of the grammar format and parser features that warrants a version bump. VISL CG-3 is feature-wise backwards compatible with CG-2 and VISLCG, and contains switches to enforce their respective behavior in certain areas.

Unicode

Even before any other designing and development had started, the desire to include Unicode support was a strong requirement. By default, the codepage for the grammar file, input stream, and output stream is detected from the environment. Internally and through switches there is full support for Unicode and any other encoding known by the Unicode library. I have chosen the International Components for Unicode library as it provides strong cross-platform Unicode support with built-in regular expressions.

Dicussion, Mailing Lists, Bug Reports, etc...

All public discussion, feature requests, bug reports, and talk of related technologies happen in the Constraint Grammar via Google Groups.

Chapter 2. License

GNU General Public License

Open Source Exception

If you plan on embedding VISL CG-3 into an open source project released under an OSI approved license that is not compatible with the GPL, please contact GrammarSoft ApS so we can grant you an explicit exception for the license or project.

Commercial/Proprietary License

If you wish to embed VISL CG-3 into a proprietary or closed source project, please contact GrammarSoft ApS to negotiate a commercial license.

Chapter 3. Installation & Updating

These guidelines are primarily for Linux, although I develop on Windows using Microsoft Visual C++ 2010 Express.

CMake Notes

A few have wondered how one installs into a different folder than the default /usr/local when there is no --prefix option to CMake. Pass `-DCMAKE_INSTALL_PREFIX=/your/folder/here` to `./cmake.sh` or run `cmake -DCMAKE_INSTALL_PREFIX=/your/folder/here .` if you have already run cmake.sh.

Also, you do not need to run cmake.sh at every build; make will detect stale files and re-run CMake as needed, just like automake. The cmake.sh script is to force a completely clean rebuild.

Ubuntu / Debian

Steps tested on a clean install of Ubuntu 10.10, but should work on any version. It is assumed you have a working network connection.

Launch a Terminal from Applications -> Accessories, and in that do

```
sudo apt-get install g++ libicu-dev subversion cmake libboost-dev
# For TCMalloc
sudo apt-get install libgoogle-perftools0
cd /tmp/
svn co http://beta.visl.sdu.dk/svn/visl/tools/vislcg3/trunk vislcg3
cd vislcg3/
./cmake.sh
make -j3
./test/runall.pl
sudo make install
```

Rest of this page can be skipped.

Fedora / Red Hat

Steps tested on a clean install of Fedora 14, but should work on any version. It is assumed you have a working network connection.

Launch a Terminal from Applications -> System Tools, and in that do

```
su -
yum install gcc-c++ libicu-devel subversion cmake boost-devel
# For TCMalloc
yum install google-perftools-devel
cd /tmp/
svn co http://beta.visl.sdu.dk/svn/visl/tools/vislcg3/trunk vislcg3
cd vislcg3/
./cmake.sh
make -j3
./test/runall.pl
make install
```

Rest of this page can be skipped.

Mac OS X

Homebrew

Launch a Terminal and in that do

```
brew install cmake
brew install boost
brew install icu4c
brew link icu4c
cd /tmp
svn co http://beta.visl.sdu.dk/svn/visl/tools/vislcg3/trunk vislcg3
cd vislcg3/
./cmake.sh
make -j3
./test/runall.pl
make install
```

Rest of this page can be skipped.

MacPorts

Launch a Terminal and in that do

```
sudo port install cmake
sudo port install boost
sudo port install icu
cd /tmp
svn co http://beta.visl.sdu.dk/svn/visl/tools/vislcg3/trunk vislcg3
cd vislcg3/
./cmake.sh -DCMAKE_INSTALL_PREFIX=/opt/local
make -j3
./test/runall.pl
sudo make install
```

Rest of this page can be skipped.

Other

Installing from source is very similar to Linux, but since the developer tools for OS X are so large, we provide binaries from the download folder. Look for files named *-osx.tar.gz. The archive contains the vislcg3, cg-comp, cg-proc, and cg-conv tools, the ICU library binaries, and wrapper scripts to enable running CG-3 from a self-contained folder.

Windows

Installing from source is rather complicated due to lack of standard search paths for libraries, so we provide binaries from the download folder. Look for files named *-win32.zip. The archive contains the vislcg3, cg-comp, cg-proc, and cg-conv tools and the ICU library DLLs. May also require installing the VC++ 2010 redist or VC++ 2008 redist.

Installing ICU

International Components for Unicode are required to compile and run VISL CG-3. Only need to do this once, however it must be done unless you already have a recent (newer or equal to ICU 3.6) version installed. ICU 3.4 may also work, but is considerably slower. Latest info on ICU can be found at http://icu-project.org/. I always develop and test with the latest version available.

Newer distributions may have a usable version of ICU available for install via the usual yum or apt managers. Just make sure to get the developer package alongside the runtime libraries.

If you do not have ICU from your distribution, manual install is as follows. These steps have been tested on all Red Hat based distributions from RH8 to Fedora 8. Similar steps have been tested on Ubuntu 7.10, and Mac OS X 10.3 to 10.5 both PPC and Intel. They may vary for your distribution.

As root:

```
cd /tmp
wget -c \
  'http://download.icu-project.org/files/icu4c/4.8/icu4c-4_8-src.tgz'
tar -zxvf icu4c-4_8-src.tgz
cd icu/source/
./runConfigureICU Linux
make
make install
echo "/usr/local/lib" >> /etc/ld.so.conf
ldconfig
```

Getting & Compiling VISL CG-3

This step requires you have Subversion installed. Subversion binaries are very likely available for your distribution already; try using **yum install subversion** or **apt-get install subversion** or whichever package manager your distribution uses.

As any user in any folder where you can find it again:

```
svn co \
  http://beta.visl.sdu.dk/svn/visl/tools/vislcg3/trunk vislcg3
cd vislcg3/
./cmake.sh
make
./test/runall.pl
... and if all tests succeed ...
make install
```

Updating VISL CG-3

In the same folder you chose above, as the same user:

```
$ svn up
$ make
$ ./test/runall.pl
... and if all tests succeed ...
$ make install
```

Regression Testing

After successfully compiling the binary, you can run the regression test suite with the command:

```
./test/runall.pl
```

This will run a series of tests that should all exit with "Success Success". If a test on your end exits with "Fail", please tar up that tests' folder and send it to me alongside any ideas you may have as to why it failed.

Cygwin

While Cygwin can compile and run VISL CG-3 via the cmake toolchain, it cannot be recommended as it is very slow (even when using GCC 4.3). Instead, compile with Microsoft Visual C++ or use the latest precompiled binary.

Chapter 4. Contributing & Subversion Access

The anonymous user does not have commit rights, and also cannot access any other part of the repository. If you want commit rights to VISL CG-3, just tell me...

Chapter 5. Compatibility and Incompatibilities

Things to be aware of.

Gotcha's

Magic Readings

In CG-3 all cohorts have at least one reading. If none are given in the input, one is generated from the wordform. These magic readings can be the target of rules, which may not always be intended.

For example, given the input

```
"<word>"
  "word" N NOM SG
"<$.>"
```

a magic reading is made so the cohorts internally looks like

```
"<word>"
  "word" N NOM SG
"<$.>"
  "<$.>" <<<
```

The above input combined with a rule a'la

```
MAP (@X) (*) ;
```

will give the output

```
"<word>"
  "word" N NOM SG @X
"<$.>"
  "<$.>" <<< @X
```

because MAP promoted the last magic reading to a real reading.

If you do not want these magic readings to be the possible target of rules, you can use the cmdline option --no-magic-readings. Internally they will still be generated and contextual tests can still reference them, but rules cannot touch or modify them directly. *SETCHILD is an exception.*

NOT and NEGATE

In CG-2 and VISLCG the keyword NOT behaved differently depending on whether it was in front of the first test or in front of a linked test. In the case of

```
(NOT 1 LSet LINK 1 KSet LINK 1 JSet)
```

the NOT would apply last, meaning it would invert the result of the entire chain, but in the case of

```
(1 LSet LINK NOT 1 KSet LINK 1 JSet)
```

it only inverts the result of the immediately following test.

CG-3 implements the NEGATE keyword to make the distinction clearer. This means that if you are converting grammars to CG-3 you must replace starting NOTs with NEGATEs to get the same functionality. So the first test should instead be

```
(NEGATE 1 LSet LINK 1 KSet LINK 1 JSet)
```

Alternatively you can use the --vislcg-compat (short form -2) to work with older grammars that you do not wish to permanently update to use NEGATE.

PREFERRED-TARGETS

PREFERRED-TARGETS is currently ignored in CG-3. See PREFERRED-TARGETS for details.

Default Codepage / Encoding

CG-3 will auto-detect the codepage from the environment, which in some cases is not what you want. It is not uncommon to work with UTF-8 data but have your environment set to US-ASCII which would produce some unfortunate errors. You can use the runtime option -C to override the default codepage, and you should always enforce it if you plan on distributing packages that depend on a certain codepage.

Set Operator -

In CG-2 the - operator means set difference; in VISLCG it means set fail-fast; in CG-3 it means set difference again, but the new operator ^ takes place of VISLCG's behavior.

Scanning Past Point of Origin

In CG-1 and some versions of CG-2, scanning tests could not pass the point of origin, but in CG-3 they can by default do so. The cmdline flag --no-pass-origin can set the default behavior to that of CG-1. See Scanning Past Point of Origin for more details.

>>> and <<<

In VISLCG the magic tags >>> and <<<, denoting sentence start and end respectively, could sometimes wind up in the output. In CG-3 they are never part of the output.

Rule Application Order

In CG-2 the order in which rules are applied on cohorts cannot be reliably predicted.

In VISLCG rules can be forced to be applied in the order they occur in the grammar, but VISLCG will try to run all rules on the current cohort before trying next cohort:

```
ForEach (Window)
  ForEach (Cohort)
```

```
    ForEach (Rule)
     ApplyRule
```

CG-3 always applies rules in the order they occur in the grammar, and will try the current rule on all cohorts in the window before moving on to the next rule. This yields a far more predictable result and cuts down on the need for many sections in the grammar.

```
ForEach (Window)
  ForEach (Rule)
   ForEach (Cohort)
    ApplyRule
```

Endless Loops

Since any rule can be in any section, it is possible to write endless loops.

For example, this grammar will potentially loop forever:

```
SECTION
ADD (@not-noun) (N) (0 (V)) ;
ADD (@noun) (N) ;

SECTION
REMOVE (@noun) IF (0 (V)) ;
```

Since ADD is in a SECTION it will be run again after REMOVE, and since ADD does not block from further appending of mapping tags it can re-add @noun each time, leading to REMOVE finding it and removing, ad nauseum.

In order to prevent this, the REMOVE rule can in most cases be rewritten to:

```
REMOVE (N) IF (0 (@noun) + (N)) (0 (V)) ;
```

That is, the target of the REMOVE rule should be a non-mapping tag with the mapping tag as 0 context. This will either remove the entire reading or nothing, as opposed to a single mapping tag, and will not cause the grammar to rewind.

Similarly, it is possible to make loops with APPEND and SELECT/REMOVE/IFF combinations, and probably many other to-be-discovered mixtures of rules. Something to be aware of.

Visibility of Mapped Tags

In CG-1, CG-2, and VISLCG it was not always clear when you could refer to a previously mapped in the same grammar. In VISL CG-3 all changes to readings become visible to the rest of the grammar immediately.

Incompatibilites

Mappings

The CG-2 spec says that readings in the format

```
"word" tag @MAP @MUP ntag @MIP
```

should be equivalent to

```
"word" tag @MAP
"word" tag @MUP
"word" tag ntag @MIP
```

Since the tag order does not matter in CG-3, this is instead equivalent to

```
"word" tag ntag @MAP
"word" tag ntag @MUP
"word" tag ntag @MIP
```

Baseforms & Mixed Input

The CG-2 spec says that the first tag of a reading is the baseform, whether it looks like [baseform] or "baseform". This is not true for CG-3; only "baseform" is valid.

The reason for this is that CG-3 has to ignore all meta text such as XML, and the only way I can be sure what is a reading and what is meta text is to declare that a reading is only valid in the forms of

```
"baseform" tag tags moretags
"base form" tag tags moretags
```

and not in the forms of

```
[baseform] tag tags moretags
baseform tag tags moretags
```

Chapter 6. Command Line Reference

A list of binaries available and their usage information.

vislcg3

vislcg3 is the primary binary. It can run rules, compile grammars, and so on.

```
Usage: vislcg3 [OPTIONS]

Options:
 -h, --help                shows this help
 -?, --?                   shows this help
 -V, --version             prints copyright and version information
 -g, --grammar             specifies the grammar file to use for disambiguation
     --grammar-out         writes the compiled grammar in textual form to a file
     --grammar-bin         writes the compiled grammar in binary form to a file
     --grammar-only        only compiles the grammar; implies --verbose
 -u, --unsafe              allows the removal of all readings in a cohort, even the las
 -s, --sections            number of sections to run; defaults to all sections
 -v, --verbose             increases verbosity
 -2, --vislcg-compat       enables compatibility mode for older CG-2 and vislcg gramma
 -I, --stdin               file to print output to instead of stdout
 -O, --stdout              file to read input from instead of stdin
 -E, --stderr              file to print errors to instead of stderr
 -C, --codepage-all        codepage to use for grammar, input, and output streams; def
     --codepage-grammar    codepage to use for grammar; overrides --codepage-all
     --codepage-input      codepage to use for input; overrides --codepage-all
     --codepage-output     codepage to use for output and errors; overrides --codepage-
 -L, --locale-all          locale to use for grammar, input, and output streams; defau
     --locale-grammar      locale to use for grammar; overrides --locale-all
     --locale-input        locale to use for input; overrides --locale-all
     --locale-output       locale to use for output and errors; overrides --locale-all
     --no-mappings         disables all MAP, ADD, and REPLACE rules
     --no-corrections      disables all SUBSTITUTE and APPEND rules
     --no-before-sections  disables all rules in BEFORE-SECTIONS parts
     --no-sections         disables all rules in SECTION parts
     --no-after-sections   disables all rules in AFTER-SECTIONS parts
 -t, --trace               prints debug output alongside with normal output
     --trace-name-only     if a rule is named, omit the line number; implies --trace
     --trace-no-removed    does not print removed readings; implies --trace
     --single-run          runs each section only once
 -S, --statistics          gathers profiling statistics while applying grammar
 -Z, --optimize-unsafe     destructively optimize the profiled grammar to be faster
 -z, --optimize-safe       conservatively optimize the profiled grammar to be faster
 -p, --prefix              sets the mapping prefix; defaults to @
     --num-windows         number of windows to keep in before/ahead buffers; defaults
     --always-span         forces scanning tests to always span across window boundari
     --soft-limit          number of cohorts after which the SOFT-DELIMITERS kick in; 
     --hard-limit          number of cohorts after which the window is forcefully cut;
     --dep-humanize        forces enumeration of dependencies to human-readable; will 
     --dep-original        outputs the original input dependency tag even if it is no 
     --dep-allow-loops     allows the creation of circular dependencies
     --dep-no-crossing     prevents the creation of dependencies that would result in 
     --no-magic-readings   prevents running rules on magic readings
 -o, --no-pass-origin      prevents scanning tests from passing the point of origin
     --show-unused-sets    prints a list of unused sets and their line numbers; implie
```

cg-conv

cg-conv converts between the Apertium and VISL CG stream formats. By default it converts from VISL CG to Apertium, but option -a inverses that. Currently only meant for use in a pipe.

```
USAGE: cg-conv [-a]

Options:
 -a, --a2v:    convert from Apertium to VISL format
```

cg-comp

cg-comp is a lighter tool that only compiles grammars to their binary form. It requires grammars to be in Unicode (UTF-8) encoding. Made for the Apertium toolchain.

```
USAGE: cg-comp grammar_file output_file
```

cg-proc

cg-proc is a grammar applicator which can handle the Apertium stream format. It works with binary grammars only, hence the need for cg-comp. It requires the input stream to be in Unicode (UTF-8) encoding. Made for the Apertium toolchain.

```
USAGE: cg-proc [-t] [-s] [-d] grammar_file [input_file [output_file]]

Options:
 -d, --disambiguation:    morphological disambiguation
 -s, --sections=NUM:      specify number of sections to process
 -f, --stream-format=NUM: set the format of the I/O stream to NUM,
                            where `0' is VISL format and `1' is
                            Apertium format (default: 1)
 -t, --trace:             print debug output on stderr
 -v, --version:           version
 -h, --help:              show this help
```

cg3-autobin.pl

A thin Perl wrapper for vislcg3. It will compile the grammar to binary form the first time and re-use that on subsequent runs for the speed boost. Accepts all command line options that vislcg3 does.

Chapter 7. Input/Output Stream Format

Currently, there are two supported data stream formats. Adding more as needed is not a major task, though.

Apertium Format

The cg-proc front-end processes the Apertium stream format.

VISL CG Format

The VISL CG stream format is a verticalized list of word forms with readings and optional plain text in between. For example, the sentence "*They went to the zoo to look at the bear.*" would in VISL format look akin to:

```
"<They>"
    "they" <*> PRON PERS NOM PL3 SUBJ
"<went>"
    "go" V PAST VFIN
"<to>"
    "to" PREP
"<the>"
    "the" DET CENTRAL ART SG/PL
"<zoo>"
    "zoo" N NOM SG
"<to>"
    "to" INFMARK>
"<look>"
    "look" V INF
"<at>"
    "at" PREP
"<the>"
    "the" DET CENTRAL ART SG/PL
"<bear>"
    "bear" N NOM SG
"<.>"
```

Or in CG terms:

```
"<word form>"
    "base form" tags
```

Also known as:

```
"<surface form>"
    "lexeme" tags
```

In more formal rules:

- If the line begins with "< followed by non-quotes and/or escaped quotes followed by >" (regex `/^"<(.|\\")*>"/`) then it opens a new cohort.

- If the line begins with whitespace followed by " followed by non-quotes and/or escaped quotes followed by " (regex `/^\s+"(.|\\")*"/`) then it is parsed as a reading, but only if a cohort is open at the time. Thus, any such lines seen before the first cohort is treated as text.
- Any line not matching the above is treated as text. Text is handled in two ways: If no cohort is open at the time, then it is output immediately. If a cohort is open, then it is appended to that cohort's buffer and output after the cohort. Note that text between readings will thus be moved to after the readings. Re-arranging cohorts will also re-arrange the text attached to them. Removed cohorts will still output their attached text.

This means that you can embed all kinda of extra information in the stream as long as you don't hit those exact patterns. For example, we use `<s id="unique-1234"> </s>` tags around sentences to keep track of them for corpus markup.

Chapter 8. Grammar

INCLUDE

INCLUDE loads and parses another grammar file as if it had been pasted in on the line of the INCLUDE statement, with the exception that line numbers start again from 1. Included rules can thus conflict with rules in other files if they happen to occupy the same line in multiple files. Until this is fixed, including only basic shared information such as delimiters and sets is advised.

```
INCLUDE other-file-name ;
```

The file name should not be quoted and the line must end with semi-colon. The include candidate will be looked for at a path relative to the file performing the include. Be careful not to make circular includes as they will loop forever.

Sections

CG-2 has three seperate grammar sections: SETS, MAPPINGS, and CONSTRAINTS. VISLCG added to these with the CORRECTIONS section. Each of these can only contain certain definitions, such as LIST, MAP, or SELECT. As I understand it, this was due to the original CG parser being written in a language that needed such a format. In any case, I did not see the logic or usability in such a strict format. VISL CG-3 has a single section header SECTION, which can contain any of the set or rule definitions. Sections can also be given a name for easier identification and anchor behavior, but that is optional. The older section headings are still valid and will work as expected, though.

By allowing any set or rule definition anywhere you could write a grammar such as:

```
DELIMITERS = "<$.>" ;
LIST ThisIsASet = "<sometag>" "<othertag>" ;

SECTION
LIST ThisIsAlsoASet = atag btag ctag ;
SET Hubba = ThisIsASet - (ctag) ;
SELECT ThisIsASet IF (-1 (dtag)) ;

LIST AnotherSet =  "<youknowthedrill>" ;
MAP (@bingo) TARGET AnotherSet ;
```

Notice that the first LIST ThisIsASet is outside a section. This is because sets are considered global regardless of where they are declared and can as such be declared anywhere, even before the DELIMITERS declaration should you so desire. A side effect of this is that set names must be unique across the entire grammar, but as this is also the behavior of CG-2 and VISLCG that should not be a surprise nor problem.

Rules are applied in the order they are declared. In the above example that would execute SELECT first and then the MAP rule.

BEFORE-SECTIONS

See BEFORE-SECTIONS. Takes the place of what previously were the MAPPINGS and CORRECTIONS blocks, but may contain any rule type.

SECTION

See SECTION. Takes the place of what previously were the CONSTRAINTS blocks, but may contain any rule type.

AFTER-SECTIONS

See AFTER-SECTIONS. May contain any rule type, and is run once after all other sections. This is new in CG-3.

NULL-SECTION

See NULL-SECTION. May contain any rule type, but is not actually run. This is new in CG-3.

Ordering of sections in grammar

The order and arrangement of BEFORE-SECTIONS and AFTER-SECTIONS in the grammar has no impact on the order normal SECTIONs are applied in.

An order of

```
SECTION
SECTION
BEFORE-SECTIONS
SECTION
NULL-SECTION
AFTER-SECTIONS
SECTION
BEFORE-SECTIONS
SECTION
```

is equivalent to

```
BEFORE-SECTIONS
SECTION
SECTION
SECTION
SECTION
SECTION
AFTER-SECTIONS
NULL-SECTION
```

--sections with ranges

In VISL CG-3, the --sections flag is able to specify ranges of sections to run, and can even be used to skip sections. If only a single number N is given it behaves as if you had written 1-N.

While it is possible to specify a range such as 1,4-6,3 where the selection of sections is not ascending, the actual application order will be 1, 1:4, 1:4:5, 1:4:5:6, 1:3:4:5:6 - that is, the final step will run section 3 in between 1 and 4. This is due to the ordering of rules being adamantly enforced as ascending only. If you wish to customize the order of rules you will currently have to use JUMP or EXECUTE.

```
--sections 6
--sections 3-6
--sections 2-5,7-9,13-15
```

Chapter 9. Rules

Firstly, the CG-2 optional seperation keywords IF and TARGET are completely ignored by VISL CG-3, so only use them for readability. The definitions are given in the following format:

```
KEYWORD <required_element> [optional_element] ;
...and | separates mutually exclusive choices.
```

Cheat Sheet

All rules can take an optional wordform tag before the rule keyword.

```
Reading & Tag manipulations:
    ADD <tags> <target> [contextual_tests] ;
    MAP <tags> <target> [contextual_tests] ;
    SUBSTITUTE <locate tags> <replacement tags> <target> [contextual_tests] ;
    UNMAP <target> [contextual_tests] ;

    REPLACE <tags> <target> [contextual_tests] ;
    APPEND <tags> <target> [contextual_tests] ;
    COPY <extra tags> <target> [contextual_tests] ;

    SELECT <target> [contextual_tests] ;
    REMOVE <target> [contextual_tests] ;
    IFF <target> [contextual_tests] ;

Dependency manipulation:
    SETPARENT <target> [contextual_tests]
        TO|FROM <contextual_target> [contextual_tests] ;
    SETCHILD <target> [contextual_tests]
        TO|FROM <contextual_target> [contextual_tests] ;

Relation manipulation:
    ADDRELATION <name> <target> [contextual_tests]
        TO|FROM <contextual_target> [contextual_tests] ;
    ADDRELATIONS <name> <name> <target> [contextual_tests]
        TO|FROM <contextual_target> [contextual_tests] ;
    SETRELATION <name> <target> [contextual_tests]
        TO|FROM <contextual_target> [contextual_tests] ;
    SETRELATIONS <name> <name> <target> [contextual_tests]
        TO|FROM <contextual_target> [contextual_tests] ;
    REMRELATION <name> <target> [contextual_tests]
        TO|FROM <contextual_target> [contextual_tests] ;
    REMRELATIONS <name> <name> <target> [contextual_tests]
        TO|FROM <contextual_target> [contextual_tests] ;

Cohort manipulation:
    ADDCOHORT <cohort tags> BEFORE|AFTER <target> [contextual_tests] ;
    REMCOHORT <target> [contextual_tests] ;

    MOVE [WITHCHILD <child_set>|NOCHILD] <target> [contextual_tests]
        BEFORE|AFTER [WITHCHILD <child_set>|NOCHILD] <contextual_target> [contextual_
    SWITCH <target> [contextual_tests] WITH <contextual_target> [contextual_tests] ;

Window manipulation:
    DELIMIT <target> [contextual_tests] ;
    EXTERNAL ONCE|ALWAYS <program> <target> [contextual_tests] ;

Variable manipulation:
    SETVARIABLE <name> <value> <target> [contextual_tests] ;
    REMVARIABLE <name> <target> [contextual_tests] ;

Flow control:
    JUMP <anchor_name> <target> [contextual_tests] ;
```

ADD

```
[wordform] ADD <tags> <target> [contextual_tests] ;
```

Appends tags to matching readings. Will not block for adding further tags, but can be blocked if a reading is considered mapped either via rule type MAP or from input.

```
ADD (@func £other) TARGET (target) IF (-1 KC) ;
```

COPY

```
[wordform] COPY <extra tags> <target> [contextual_tests] ;
```

Duplicates a reading and adds tags to it. If you don't want to copy previously copied readings, you will have to keep track of that yourself by adding a marker tag.

```
COPY (¤copy tags) TARGET (target) - (¤copy) ;
```

DELIMIT

```
[wordform] DELIMIT <target> [contextual_tests] ;
```

This will work as an on-the-fly sentence (disambiguation window) delimiter. When a reading matches a DELIMIT rule's context it will cut off all subsequent cohorts in the current window immediately restart disambiguating with the new window size. This is not and must not be used as a substitute for the DELIMITERS list, but can be useful for cases where the delimiter has to be decided from context.

EXTERNAL

```
[wordform] EXTERNAL ONCE <program> <target> [contextual_tests] ;
[wordform] EXTERNAL ALWAYS <program> <target> [contextual_tests] ;
```

Opens up a persistent pipe to the program and passes it the current window. The ONCE version will only be run once per window, while ALWAYS will be run every time the rule is seen. See the Externals chapter for technical and protocol information.

```
EXTERNAL ONCE /usr/local/bin/waffles (V) (-1 N) ;
EXTERNAL ALWAYS program-in-path (V) (-1 N) ;
EXTERNAL ONCE "program with spaces" (V) (-1 N) ;
```

ADDCOHORT

```
[wordform] ADDCOHORT <cohort tags> BEFORE|AFTER <target> [contextual_tests] ;
```

Inserts a new cohort before or after the target.

Caveat: This does NOT affect the rule application order as that is tied to the input order of the cohorts, so inserted cohorts are always last.

```
ADDCOHORT ("<wordform>" "baseform" tags) BEFORE (@waffles) ;
ADDCOHORT ("<wordform>" "baseform" tags) AFTER (@waffles) ;
```

REMCOHORT

```
[wordform] REMCOHORT <target> [contextual_tests] ;
```

This will entirely remove a cohort with all its readings from the window.

MOVE, SWITCH

```
[wordform] MOVE [WITHCHILD <child_set>|NOCHILD] <target> [contextual_tests]
    AFTER [WITHCHILD <child_set>|NOCHILD] <contextual_target> [contextual_tests]
[wordform] MOVE [WITHCHILD <child_set>|NOCHILD] <target> [contextual_tests]
    BEFORE [WITHCHILD <child_set>|NOCHILD] <contextual_target> [contextual_tests
[wordform] SWITCH <target> [contextual_tests] WITH <contextual_target> [contextua
```

Allows re-arranging of cohorts. The option WITHCHILD will cause the movement of the cohort plus all children of the cohort, maintaining their internal order. Default is NOCHILD which moves only the one cohort. SWITCH does not take options.

If you specify WITHCHILD you will need to provide a set that the children you want to apply must match. The (*) set will match all children.

The first WITHCHILD specifies which children you want moved. The second WITHCHILD uses the children of the cohort you're moving to as edges so you can avoid moving into another dependency group.

Caveat: This does NOT affect the rule application order as that is tied to the input order of the cohorts, so after movement you may see some rules touching later cohorts than you'd expect.

REPLACE

```
[wordform] REPLACE <tags> <target> [contextual_tests] ;
```

Removes all tags from a reading except the base form, then appends the given tags. Cannot target readings that are considered mapped due to earlier MAP rules or from input.

```
REPLACE (<v-act> V INF @func) TARGET (target);
```

APPEND

```
[wordform] APPEND <tags> <target> [contextual_tests] ;
```

Appends a reading to the matched cohort, so be sure the tags include a baseform.

```
APPEND ("jump" <v-act> V INF @func) TARGET (target);
```

SUBSTITUTE

```
[wordform] SUBSTITUTE <locate tags> <replacement tags> <target> [contextual_tests
```

Replaces the tags in the first list with the tags in the second list. If none of the tags in the first list are found, the insertion tags are simply appended. To prevent this, also have important tags as part of the target. This works as in VISLCG, but the replacement tags may be the * tag to signify a nil replacement, allowing for clean removal of tags in a reading. For example, to remove TAG do:

```
SUBSTITUTE (TAG) (*) TARGET (TAG) ;
```

SETVARIABLE

```
[wordform] SETVARIABLE <name> <value> <target> [contextual_tests] ;
```

Sets a global variable to a given value. If you don't care about the value you can just set it to 1 or *.

```
SETVARIABLE (news) (*) (@headline) ;
SETVARIABLE (year) (1764) ("Jozef") (-2 ("Arch") LINK 1 ("Duke")) ;
```

REMVARIABLE

```
[wordform] REMVARIABLE <name> <target> [contextual_tests] ;
```

Unsets a global variable.

```
REMVARIABLE (news) (@stanza) ;
REMVARIABLE (year) (@timeless) ;
```

MAP

```
[wordform] MAP <tags> <target> [contextual_tests] ;
```

Appends tags to matching readings, and blocks other MAP, ADD, and REPLACE rules from targetting those readings. Cannot target readings that are considered mapped due to earlier MAP rules or from input.

```
MAP (@func £other) TARGET (target) IF (-1 KC) ;
```

UNMAP

```
[wordform] UNMAP <target> [contextual_tests] ;
```

Removes the mapping tag of a reading and lets ADD and MAP target the reading again. By default it will only act if the cohort has exactly one reading, but marking the rule UNSAFE lets it act on multiple readings.

```
UNMAP (TAG) ;
UNMAP UNSAFE (TAG) ;
```

Tag Lists Can Be Sets

For the rule types MAP, ADD, REPLACE, APPEND, COPY, SUBSTITUTE, ADDRELATION(S), SETRELATION(S), and REMRELATION(S), the tag lists can be sets instead, including $$sets and &&sets. This is useful for manipulating tags that you want to pull in from a context.

The sets are resolved and reduced to a list of tags during rule application. If the set reduces to multiple tags where only one is required (such as for the Relation rules), only the first tag is used.

```
LIST ROLE = <human> <anim> <inanim> (<bench> <table>) ;
MAP $$ROLE (target tags) (-1 KC) (-2C $$ROLE) ;
```

Named Rules

```
[wordform] MAP:rule_name <tag> <target> [contextual_tests] ;
[wordform] SELECT:rule_name <target> [contextual_tests] ;
```

In certain cases you may want to name a rule to be able to refer to the same rule across grammar revisions, as otherwise the rule line number may change. It is optional to name rules, and names do not have to be unique which makes it easier to group rules for statistics or tracing purposes. The name of a rule is used in tracing and debug output in addition to the line number.

Flow Control: JUMP, ANCHOR

JUMP will allow you to mark named anchors and jump to them based on a context. In this manner you can skip or repeat certain rules.

```
ANCHOR <anchor_name> ;
SECTION [anchor_name ;]
[wordform] JUMP <anchor_name> <target> [contextual_tests] ;
```

An anchor can be created explicitly with keyword ANCHOR. Sections can optionally be given a name which will make them explicit anchor points. All named rules are also anchor points, but with lower precedence than explicit anchors, and in the case of multiple rules with the same name the first such named rule is the anchor point for that name. There are also two special anchors `START` and `END` which represent line 0 and infinity respectively; jumping to START will re-run all currently active rules, while jumping to END will skip all remaining rules.

Rule Options

Rules can have options that affect their behavior. Multiple options can be combined per rule and the order is not important, just separate them with space.

```
# Remove readings with (unwanted) even if it is the last reading.
REMOVE UNSAFE (unwanted) ;

# Attach daughter to mother, even if doing so would cause a loop.
SETPARENT ALLOWLOOP (daughter) TO (-1* (mother)) ;
```

NEAREST

Applicable for rules SETPARENT and SETCHILD. Not compatible with option ALLOWLOOP.

Normally, if SETPARENT or SETCHILD cannot attach because doing so would cause a loop, they will seek onwards from that position until a valid target is found that does not cause a loop. Setting NEAREST forces them to stop at the first found candidate.

ALLOWLOOP

Applicable for rules SETPARENT and SETCHILD. Not compatible with option NEAREST.

Normally, SETPARENT and SETCHILD cannot attach if doing so would cause a loop. Setting ALLOWLOOP forces the attachment even in such a case.

ALLOWCROSS

Applicable for rules SETPARENT and SETCHILD.

If command line flag --dep-no-crossing is on, SETPARENT and SETCHILD cannot attach if doing so would cause crossing branches. Setting ALLOWCROSS forces the attachment even in such a case.

DELAYED

Applicable for rules SELECT, REMOVE, and IFF. Not compatible with option IMMEDIATE.

Option DELAYED causes readings that otherwise would have been put in the deleted buffer to be put in a special delayed buffer, in the grey zone between living and dead.

Delayed readings can be looked at by contextual tests of rules that have option LOOKDELAYED, or if the contextual test has position 'd'.

IMMEDIATE

Applicable for rules SELECT, REMOVE, and IFF. Not compatible with option DELAYED.

Option IMMEDIATE causes readings that otherwise would have been put in the special delayed buffer to be put in the delayed buffer. This is mainly used to selectively override a global DELAYED flag as rules are by default immediate.

LOOKDELAYED

Applicable for all rules.

Option LOOKDELAYED puts contextual position 'd' on all tests done by that rule, allowing them all to see delayed readings.

LOOKDELETED

Applicable for all rules.

Option LOOKDELETED puts contextual position 'D' on all tests done by that rule, allowing them all to see deleted readings.

UNMAPLAST

Applicable for rules REMOVE and IFF. Not compatible with option SAFE.

Normally, REMOVE and IFF will consider mapping tags as separate readings, and attempting to remove the last mapping is the same as removing the last reading which requires UNSAFE. Setting flag UNMAPLAST causes it to instead remove the final mapping tag but leave the reading otherwise untouched.

A rule `REMOVE UNMAPLAST @MappingList ;` is logically equivalent to `REMOVE @MappingList ; UNMAP @MappingList ;`

UNSAFE

Applicable for rules REMOVE and IFF and UNMAP. Not compatible with option SAFE.

Normally, REMOVE and IFF cannot remove the last reading of a cohort. Setting option UNSAFE allows them to do so.

For UNMAP, marking it UNSAFE allows it to work on more than the last reading in the cohort.

SAFE

Applicable for rules REMOVE and IFF and UNMAP. Not compatible with option UNSAFE.

SAFE prevents REMOVE and IFF from removing the last reading of a cohort. Mainly used to selectively override global --unsafe mode.

For UNMAP, marking it SAFE only lets it act if the cohort has exactly one reading.

REMEMBERX

Applicable for all rules. Not compatible with option RESETX.

Makes the contextual option X carry over to subsequent tests in the rule, as opposed to resetting itself to the rule's target per test. Useful for complex jumps with the X and x options.

RESETX

Applicable for all rules. Not compatible with option REMEMBERX.

Default behavior. Resets the mark for contextual option x to the rule's target on each test. Used to counter a global REMEMBERX.

KEEPORDER

Applicable for all rules. Not compatible with option VARYORDER.

Prevents the re-ordering of contextual tests. Useful in cases where a unifying set is not in the target of the rule.

VARYORDER

Applicable for all rules. Not compatible with option KEEPORDER.

Allows the re-ordering of contextual tests. Used to selectively override a global KEEPORDER flag as test order is by default fluid.

ENCL_INNER

Applicable for all rules. Not compatible with other ENCL_* options.

Rules with ENCL_INNER will only be run inside the currently active parantheses enclosure. If the current window has no enclosures, the rule will not be run.

ENCL_OUTER

Applicable for all rules. Not compatible with other ENCL_* options.

Rules with ENCL_OUTER will only be run outside the currently active parentheses enclosure. Previously expanded enclosures will be seen as outside on subsequent runs. If the current window has no enclosures, the rule will be run as normal.

ENCL_FINAL

Applicable for all rules. Not compatible with other ENCL_* options.

Rules with ENCL_FINAL will only be run once all parentheses enclosures have been expanded. If the current window has no enclosures, the rule will be run as normal.

ENCL_ANY

Applicable for all rules. Not compatible with other ENCL_* options.

The default behavior. Used to counter other glocal ENCL_* flags.

WITHCHILD

Applicable for rule type MOVE. Not compatible with option NOCHILD.

Normally, MOVE only moves a single cohort. Setting option WITHCHILD moves all its children along with it.

NOCHILD

Applicable for rule type MOVE. Not compatible with option WITHCHILD.

If the global option WITHCHILD is on, NOCHILD will turn it off for a single MOVE rule.

ITERATE

Applicable for all rule types. Not compatible with option NOITERATE.

If the rule does anything that changes the state of the window, ITERATE forces a reiteration of the sections. Normally, only changes caused by rule types SELECT, REMOVE, IFF, DELIMIT, REMCOHORT, MOVE, and SWITCH will rerun the sections.

NOITERATE

Applicable for all rule types. Not compatible with option ITERATE.

Even if the rule does change the state of the window, NOITERATE prevents the rule from causing a reiteration of the sections.

REVERSE

Applicable for rule types SETPARENT, SETCHILD, MOVE, SWITCH, ADDRELATION(S), SETRELATION(S), REMRELATION(S).

Reverses the active cohorts. E.g., effectively turns a SETPARENT into a SETCHILD.

SUB:N

See the Sub-Readings SUB:N section.

Chapter 10. Contextual Tests

Position Element Order

CG-3 is not very strict with how a contextual position looks. Elements such as ** and C can be anywhere before or after the offset number, and even the - for negative offsets does not have to be near the number.

Examples of valid positions:

```
*-1
1**-
W*2C
0**>
cC
CsW
```

NEGATE

NEGATE is similar to, yet not the same as, NOT. Where NOT will invert the result of only the immediately following test, NEGATE will invert the result of the entire LINK'ed chain that follows. NEGATE is thus usually used at the beginning of a test. *VISLCG emulated the NEGATE functionality for tests that started with NOT.*

CBARRIER

Like BARRIER but performs the test in Careful mode, meaning it only blocks if all readings in the cohort matches. This makes it less strict than BARRIER.

```
(**1 SetG CBARRIER (Verb))
```

Spanning Window Boundaries

These options allows a test to find matching cohorts in any window currently in the buffer. The buffer size can be adjusted with the --num-windows cmdline flag and defaults to 2. Meaning, 2 windows on either side of the current one is preserved, so a total of 5 would be in the buffer at any time.

Span Both

Allowing a test to span beyond boundaries in either direction is denoted by 'W'. One could also allow all tests to behave in this manner with the --always-span cmdline flag.

```
(-1*W (someset))
```

Span Left

'<' allows a test to span beyond boundaries in left direction only.

```
        (-3**< (someset))
```

Span Right

'>' allows a test to span beyond boundaries in right direction only.

```
        (2*> (someset))
```

X Marks the Spot

By default, linked tests continue from the immediately preceding test. These options affect behavior.

```
      # Look right for (third), then from there look right for (seventh),
      # then jump back to (target) and from there look right for (fifth)
      SELECT (target) (1* (third) LINK 1* (seventh) LINK 1*x (fifth)) ;

      # Look right for (fourth), then from there look right for (seventh) and set that
      # then from there look left for (fifth), then jump back to (seventh) and from the
      SELECT (target) (1* (fourth) LINK 1*X (seventh) LINK -1* (fifth) LINK -1*x (sixth
```

Set Mark

'X' sets the mark to the currently active cohort of the test's target. If no test sets X then the mark defaults to the cohort from the rule's target.

Jump to Mark

'x' jumps back to the previously set mark (or the rule's target if no mark is set), then proceeds from there.

Attach To

'A' sets the cohort to be attached or related against to the currently active cohort of the test's target.

Test Deleted/Delayed Readings

By default, removed reading are not visible to tests. These options allow tests to look at deleted and delayed readings.

Look at Deleted Readings

'D' allows the current test (and any barrier) to look at readings that have previously been deleted by SELECT/REMOVE/IFF. Delayed readings are not part of 'D', but can be combined as 'Dd' to look at both.

Look at Delayed Readings

'd' allows the current test (and any barrier) to look at readings that have previously been deleted by SELECT/REMOVE/IFF DELAYED. Deleted readings are not part of 'd', but can be combined as 'Dd' to look at both.

Scanning Past Point of Origin

By default, linked scanning tests are allowed to scan past the point of origin. These options affect behavior.

--no-pass-origin, -o

The --no-pass-origin (or -o in short form) changes the default mode to not allow passing the origin, and defines the origin as the target of the currently active rule. This is equivalent to adding 'O' to the first test of each contextual test of all rules.

No Pass Origin

'O' sets the point of origin to the parent of the contextual test, and disallows itself and all linked tests to pass this point of origin. The reason it sets it to the parent is that otherwise there is no way to mark the rule's target as the desired origin.

```
# Will not pass the (origin) cohort when looking for (right):
SELECT (origin) IF (-1*O (left) LINK 1* (right)) ;

# Will not pass the (origin) cohort when looking for (left):
SELECT (something) IF (-1* (origin) LINK 1*O (right) LINK -1* (left)) ;
```

Pass Origin

'o' allows the contextual test and all its linked tests to pass the point of origin, even in --no-pass-origin mode. Used to counter 'O'.

```
# Will pass the (origin) cohort when looking for (right), but not when looking
SELECT (origin) IF (-1*O (left) LINK 1* (middle) LINK 1*o (right)) ;
```

Nearest Neighbor

Usually a '*' or '**' test scans in only one direction, denoted by the value of the offset; if offset is positive it will scan rightwards, and if negative leftwards. In CG-3 the magic offset '0' will scan in both directions; first one to the left, then one to the right, then two to the left, then two to the right, etc. This makes it easy to find the nearest neighbor that matches. *In earlier versions of CG this could be approximated with two seperate rules, and you had to scan entirely in one direction, then come back and do the other direction.*

Caveat: (NOT 0* V) will probably not work as you expect; it will be true if either direction doesn't find set V. What you want instead is (NEGATE 0* V) or split into (NOT -1* V) (NOT 1* V).

```
(0* (someset))
(0**W (otherset))
```

Dependencies

CG-3 also introduces the p, c, and s positions. See the section about those in the Dependencies chapter.

Relations

CG-3 also introduces the r:rel and r:* positions. See the section about those in the Relations chapter.

Chapter 11. Parenthesis Enclosures

A new feature in CG-3 is handling of enclosures by defining pairs of parentheses. Any enclosure found will be omitted from the window on first run, then put back in the window and all rules are re-run on the new larger window. This continues until all enclosures have been put back one-by-one.

The idea is that by omitting the enclosures their noise cannot disrupt disambiguation, thus providing an easy way to write clean grammars that do not have to have special tests for parentheses all over the place.

Example

Example is adapted from the ./test/T_Parentheses/ regression test.

Given the following sentence:

```
There once were (two [three] long red (more like maroon (dark earthy red]), actua
```

...and given the following parenthesis wordform pairs:

```
PARENTHESES = ("<(>" "<)>") ("<[>" "<]>") ("<{>" "<}>") ;
```

...results in the sentence being run in the order of:

```
1: There once were (two long red cars.
2: There once were (two [three] long red cars.
3: There once were (two [three] long red (more like maroon, actually) cars.
4: There once were (two [three] long red (more like maroon (dark earthy red]), ac
5: There once were (two [three] long red (more like maroon (dark earthy red]), ac
```

The example has 2 unmatched parenthesis in the words (two and red] which are left in untouched as they are not really enclosing anything.

Note that enclosures are put back in the window left-to-right and only one at the time. The depth of enclosure has no effect on the order of resurrection. This may seem unintuitive, but it was the most efficient way of handling it.

Also of note is that all rules in all sections will be re-run each time an enclosure is resurrected. This includes BEFORE-SECTIONS and AFTER-SECTIONS. So in the above example, all of those are run 5 times.

Contextual Position L

In a contextual test you can jump to the leftward parenthesis of the currently active enclosure with the L position. It is only valid from within the enclosure.

```
(L (*) LINK 1* (V) BARRIER _RIGHT_)
```

Contextual Position R

In a contextual test you can jump to the rightward parenthesis of the currently active enclosure with the R position. It is only valid from within the enclosure.

```
(R (*) LINK -1* (V) BARRIER _LEFT_)
```

Magic Tag _LEFT_

A magic tag that represents the active enclosure's leftward parenthesis wordform. This tag is only valid when an enclosure is active and only exactly on the leftward parenthesis cohort. Useful for preventing scanning tests from crossing it with a barrier.

Magic Tag _RIGHT_

A magic tag that represents the active enclosure's rightward parenthesis wordform. This tag is only valid when an enclosure is active and only exactly on the rightward parenthesis cohort. Useful for preventing scanning tests from crossing it with a barrier.

Magic Tag _ENCL_

This tag is only valid when an enclosure is hidden away and only on cohorts that own hidden cohorts. Useful for preventing scanning tests from crossing hidden enclosures with a barrier.

Magic Set _LEFT_

A magic set containing the single tag (_LEFT_).

Magic Set _RIGHT_

A magic set containing the single tag (_RIGHT_).

Magic Set _ENCL_

A magic set containing the single tag (_ENCL_).

Magic Set _PAREN_

A magic set defined as _LEFT_ OR _RIGHT_.

Chapter 12. Making use of Dependencies

CG-3 can work with dependency trees in various ways. The input cohorts can have existing dependencies; the grammar can create new attachments; or a combination of the two.

SETPARENT

```
[wordform] SETPARENT <target> [contextual_tests]
    TO|FROM <contextual_target> [contextual_tests] ;
```

Attaches the matching reading to the contextually targetted cohort as a child. The last link of the contextual test is used as target.

If the contextual target is a scanning test and the first found candidate cannot be attached due to loop prevention, SETPARENT will look onwards for the next candidate. This can be controlled with rule option NEAREST and ALLOWLOOP.

```
SETPARENT targetset (-1* ("someword"))
  TO (1* (step) LINK 1* (candidate)) (2 SomeSet) ;
```

SETCHILD

```
[wordform] SETCHILD <target> [contextual_tests]
    TO|FROM <contextual_target> [contextual_tests] ;
```

Attaches the matching reading to the contextually targetted cohort as the parent. The last link of the contextual test is used as target.

If the contextual target is a scanning test and the first found candidate cannot be attached due to loop prevention, SETCHILD will look onwards for the next candidate. This can be controlled with rule option NEAREST and ALLOWLOOP.

```
SETCHILD targetset (-1* ("someword"))
  TO (1* (step) LINK 1* (candidate)) (2 SomeSet) ;
```

Existing Trees in Input

Dependency attachments in input comes in the form of #X->Y tags where X the number of the current node and Y is the number of the parent node. The X must be unique positive integers and should be sequentially enumerated. '0' is reserved and means the root of the tree, so no node may claim to be '0', but nodes may attach to '0'.

If the Y of a reading cannot be located, it will be reattached to itself. If a reading contains more than one attachment, only the last will be honored. If a cohort has conflicting attachments in its readings, the result is undefined.

For example:

```
  "<There>"
  "there" <*> ADV @F-SUBJ #1->0
"<once>"
  "once" ADV @ADVL #2->0
"<was>"
  "be" <SVC/N> <SVC/A> V PAST SG1/3 VFIN IMP @FMV #3->2
"<a>"
  "a" <Indef> ART DET CENTRAL SG @>N #4->5
"<man>"
  "man" N NOM SG @SC #5->0
"<$.>"
```

Creating Trees from Grammar

It is also possible to create or modify the tree on-the-fly with rules. See SETPARENT and SETCHILD. Dependencies created in this fashion will be output in the same format as above.

For example:

```
SETPARENT (@>N) (0 (ART DET))
  TO (1* (N)) ;

SETPARENT (@<P)
  TO (-1* (PRP)) (NEGATE 1* (V)) ;
```

Contextual Tests

Either case, once you have a dependency tree to work with, you can use that in subsequent contextual tests as seen below. These positions can be combined with the window spanning options.

Parent

The 'p' position asks for the parent of the current position.

```
(-1* (ADJ) LINK p (N))
```

Children

The 'c' position asks for a child of the current position.

```
(-1* (N) LINK c (ADJ))
```

Siblings

The 's' position asks for a sibling of the current position.

```
(-1* (ADJ) LINK s (ADJ))
```

Self

The 'S' option allows the test to look at the current target as well. Used in conjunction with p, c, or s to test self and the relations.

```
(cS (ADJ))
```

Deep Scan

The '*' option behaves differently when dealing with dependencies. Here, the '*' option allows the test to perform a deep scan. Used in conjunction with p, c, or s to continue until there are no deeper relations. For example, position 'c*' tests the children, grand-children, grand-grand-children, and so forth.

```
(c* (ADJ))
```

All Scan

The 'ALL' option will require that all of the relations match the set. For example, position 'ALL s' requires that all of the siblings match the set. For now, you can still write 'sC' which will be converted to 'ALL s', but that is deprecated and will change meaning in future versions of CG-3. This still means that if you wish to find a relation where all readings have a set you must do e.g. (c (ADJ) LINK 0C (ADJ)).

```
(ALL s (ADJ))
```

None Scan

The 'NONE' option will require that none of the relations match the set. For example, position 'NONE c' requires that none of the children match the set. For now, you can still write 'NOT c' which will be converted to 'NONE c', but that is deprecated and will change meaning in future versions of CG-3. This still means that if you wish to find a reading not matching a set you must do e.g. (c (*) LINK NOT 0 (ADJ)).

```
(NONE c (ADJ))
```

Chapter 13. Making use of Relations

CG-3 can also work with generic relations. These are analogous to dependency relations, but can have any name, overlap, are directional, and can point to multiple cohorts.

ADDRELATION, ADDRELATIONS

```
[wordform] ADDRELATION <name> <target> [contextual_tests]
    TO|FROM <contextual_target> [contextual_tests] ;
[wordform] ADDRELATIONS <name> <name> <target> [contextual_tests]
    TO|FROM <contextual_target> [contextual_tests] ;
```

ADDRELATION creates a one-way named relation from the current cohort to the found cohort. The name must be an alphanumeric string with no whitespace.

```
ADDRELATION (name) targetset (-1* ("someword"))
  TO (1* (@candidate)) (2 SomeSet) ;
```

ADDRELATIONS creates two one-way named relation; one from the current cohort to the found cohort, and one the other way. The names can be the same if so desired.

```
ADDRELATIONS (name) (name) targetset (-1* ("someword"))
  TO (1* (@candidate)) (2 SomeSet) ;
```

SETRELATION, SETRELATIONS

```
[wordform] SETRELATION <name> <target> [contextual_tests]
    TO|FROM <contextual_target> [contextual_tests] ;
[wordform] SETRELATIONS <name> <name> <target> [contextual_tests]
    TO|FROM <contextual_target> [contextual_tests] ;
```

SETRELATION removes all previous relations with the name, then creates a one-way named relation from the current cohort to the found cohort. The name must be an alphanumeric string with no whitespace.

```
SETRELATION (name) targetset (-1* ("someword"))
  TO (1* (@candidate)) (2 SomeSet) ;
```

SETRELATIONS removes all previous relations in the respective cohorts with the respective names, then creates two one-way named relation; one from the current cohort to the found cohort, and one the other way. The names can be the same if so desired.

```
SETRELATIONS (name) (name) targetset (-1* ("someword"))
  TO (1* (@candidate)) (2 SomeSet) ;
```

REMRELATION, REMRELATIONS

```
[wordform] REMRELATION <name> <target> [contextual_tests]
    TO|FROM <contextual_target> [contextual_tests] ;
[wordform] REMRELATIONS <name> <name> <target> [contextual_tests]
    TO|FROM <contextual_target> [contextual_tests] ;
```

REMRELATION destroys one direction of a relation previously created with either ADDRELATION or SETRELATION.

```
REMRELATION (name) targetset (-1* ("someword"))
  TO (1* (@candidate)) (2 SomeSet) ;
```

REMRELATIONS destroys both directions of a relation previously created with either ADDRELATION or SETRELATION.

```
REMRELATIONS (name) (name) targetset (-1* ("someword"))
  TO (1* (@candidate)) (2 SomeSet) ;
```

Existing Relations in Input

Not currently supported; relations from input are ignored at present stage of development.

Contextual Tests

Once you have relations to work with, you can use that in subsequent contextual tests as seen below. These positions can be combined with the window spanning options.

Specific Relation

The 'r:rel' position asks for cohorts found via the 'rel' relation. 'rel' can be any name previously given via ADDRELATION or SETRELATION. Be aware that for combining positional options, 'r:rel' should be the last in the position; 'r:' will eat anything following it until it meets a space.

```
  (r:rel (ADJ))
```

Any Relation

The 'r:*' position asks for cohorts found via any relation.

```
  (r:* (ADJ))
```

All Scan

The 'ALL' option will require that all of the relations match the set. For example, position 'ALL r:rel' requires that all of the 'rel' relations match the set. For now, you can still write 'Cr:rel' which will be converted to 'ALL r:rel', but that is deprecated and will change meaning in future versions of CG-3. This still means that if you wish to find a relation where all readings have a set you must do e.g. (r:rel (ADJ) LINK 0C (ADJ)).

```
(ALL r:rel (ADJ))
```

None Scan

The 'NONE' option will require that none of the relations match the set. For example, position 'NONE r:rel' requires that none of the 'rel' relations match the set. For now, you can still write 'NOT r:rel' which will be converted to 'NONE r:rel', but that is deprecated and will change meaning in future versions of CG-3. This still means that if you wish to find a reading not matching a set you must do e.g. (r:rel (*) LINK NOT 0 (ADJ)).

```
(NONE r:rel (ADJ))
```

Chapter 14. Making use of Probabilistic / Statistic Input

If your input contains confidence values or similar, you can make use of those in the grammar. *See numeric tags for the specific feature.*

For example, given the input sentence "*Bear left at zoo.*" a statistical tagger may assign confidence and frequency values to the readings:

```
"<Bear>"
  "bear" N NOM SG <Noun:784> <Conf:80> @SUBJ
  "bear" V INF <Verb:140> <Conf:20> @IMV @#ICL-AUX<
"<left>"
  "leave" PED @IMV @#ICL-N<
  "leave" V PAST VFIN @FMV
"<at>"
  "at" PRP @ADVL
"<zoo>"
  "zoo" N NOM SG @P<
"<$.>"
```

which you could query with e.g.

```
# Remove any reading with Confidence below 5%
REMOVE (<Conf<5>) ;
# Select N NOM SG if Confidence is above 60%
SELECT (N NOM SG <Conf>60>) ;
# Remove the Verb reading if the frequency is under 150
# and Noun's frequency is above 700
REMOVE (<Verb<150>) (0 (<Noun>700>)) ;
```

These are just examples of what numeric tags could be used for. There is no reason Confidence values are in % and there is no requirement that they must add up to 100%. The only requirement of a numerical tag is an alphanumeric identifier and a positive integer value.

Chapter 15. Templates

Sets up templates of alternative contextual tests which can later be referred to by multiple rules or other templates. Templates support the full syntax of contextual tests, including all other new CG-3 features. Best way to document them that I can think of at the moment is to give examples of equivalent constructions.

For example, this construction

```
TEMPLATE tmpl = 1 (a) LINK 1 B + C LINK 1 D - (e) ;
SELECT (tag) IF (T:tmpl) ;
```

is equivalent to

```
SELECT (tag) IF (1 (a) LINK 1 B + C LINK 1 D - (e)) ;
```

But with the introduction of templates, CG-3 also allows alternative tests, so this construction

```
TEMPLATE tmpl = (1 ASET LINK 1 BSET) OR (-1 BSET LINK -1 ASET) ;
SELECT (tag) IF (T:tmpl) ;
```

is equivalent to

```
# Yes, inline OR is allowed if you () the tests properly
SELECT (tag) IF ((1 ASET LINK 1 BSET) OR (-1 BSET LINK -1 ASET)) ;
```

which in turn is equivalent to

```
SELECT (tag) IF (1 ASET LINK 1 BSET) ;
SELECT (tag) IF (-1 BSET LINK -1 ASET) ;
```

For very simple lists of LINK 1 constructs, there is a further simplification:

```
# Note the use of [] and , instead of ()
TEMPLATE tmpl = [(a), BSET, CSET - (d)] ;
```

is equivalent to

```
TEMPLATE tmpl = 1 (a) LINK 1 BSET LINK 1 CSET - (d) ;
```

However, the [] construct is not directly allowed in OR constructions, so you cannot write

```
TEMPLATE tmpl = [a, b, c] OR [e, f, g] ; # invalid
```

but you can instead write

```
TEMPLATE tmpl = ([a, b, c]) OR ([e, f, g]) ; # valid
```

The [] construct can also be linked to and from, so

```
TEMPLATE tmpl = [a, b, c] LINK 1* d BARRIER h LINK [e, f, g] ;
```

is equivalent to

```
TEMPLATE tmpl = 1 a LINK 1 b LINK 1 c LINK 1* d BARRIER h LINK 1 e LINK 1 f LINK
```

Templates can be used in place of any normal contextual test, and can be both linked to and from, so

```
TEMPLATE tmpl = 1 (donut) BARRIER (waffle) ;
SELECT (tag) IF (1 VSET LINK T:tmpl LINK 1* FSET) ;
```

is equivalent to

```
SELECT (tag) IF (1 VSET LINK 1 (donut) BARRIER (waffle) LINK 1* FSET) ;
```

It is also possible to override the position of a template, which changes their behavior. E.g:

```
TEMPLATE tmpl = [N, CC, ADJ] ;
# ... or ...
TEMPLATE tmpl = 1 N LINK 1 CC LINK 1 ADJ ;
SELECT (tag) IF (-1 T:tmpl) ;
```

is equivalent to

```
SELECT (tag) IF (-1** N LINK 1 CC LINK 1 ADJ) ;
```

but with a post-condition that the final cohort matched from the exit of the template must be at the position given relative to the origin. In this case, a match of the template is only succesful if ADJ is in position -1 to the origin (tag). This behavior is equivalent to how templates worked in Fred Karlsson's CG-1, but with more flexibility.

Chapter 16. Sets

Set Operators

Union: OR and |

Equivalent to the mathematical set union # operator.

```
LIST a = a b c d ;
LIST b = c d e f ;

# Logically yields a set containing tags: a b c d e f
# Practically a reading must match either set
SET r = a OR b ;
SET r = a | b ;
```

Difference: -

Equivalent to the mathematical set complement # operator.

```
LIST a = a b c d ;
LIST b = c d e f ;

# Logically yields a set containing tags: a b !c !d !e !f
# Practically a reading must match the first set and must not match the second
SET r = a - b ;
```

Symmetric Difference:

Equivalent to the mathematical set symmetric difference # operator. The symbol is the Unicode code point U+2206.

```
LIST a = a b c d ;
LIST b = c d e f ;

# Logically yields a set containing tags: a b e f
SET r = a # b ;
```

Intersection:

Equivalent to the mathematical set intersection # operator. The symbol is the Unicode code point U+2229.

```
LIST a = a b c d ;
LIST b = c d e f ;

# Logically yields a set containing tags: c d
SET r = a # b ;
```

Cartesian Product: +

Equivalent to the mathematical set cartesian product × operator.

```
LIST a = a b c d ;
LIST b = c d e f ;

# Logically yields a set containing tags: (a c) (b c) c (d c) (a d) (b d) d (a
#                                         (b e) (c e) (d e) (a f) (b f) (c f) (
# Practically a reading must match both sets
SET r = a + b ;
```

Fail-Fast: ^

On its own, this is equivalent to set difference -. But, when followed by other sets it becomes a blocker. In `A - B OR C + D` either `A - B` or `C + D` may suffice for a match. However, in `A ^ B OR C + D`, if B matches then it blocks the rest and fails the entire set match without considering C or D.

Magic Sets

(*)

A set containing the (*) tag becomes a magic "any" set and will always match. This saves having to declare a dummy set containing all imaginable tags. Useful for testing whether a cohort exists at a position, without needing details about it. Can also be used to match everything except a few tags with the set operator -.

```
(*-1 (*) LINK 1* SomeSet)
SELECT (*) - NotTheseTags ;
```

_S_DELIMITERS_

The magic set _S_DELIMITERS_ is created from the DELIMITERS definition. This saves having to declare and maintain a seperate set for matching delimiters in tests.

```
SET SomeSet = OtherSet OR _S_DELIMITERS_ ;
```

_S_SOFT_DELIMITERS_

The magic set _S_SOFT_DELIMITERS_ is created from the SOFT-DELIMITERS definition.

```
(**1 _S_SOFT_DELIMITERS_ BARRIER BoogieSet)
```

Magic Set _TARGET_

A magic set containing the single tag (_TARGET_). This set and tag will only match when the currently active cohort is the target of the rule.

Magic Set _MARK_

A magic set containing the single tag (_MARK_). This set and tag will only match when the currently active cohort is the mark set with X, or if no such mark is set it will only match the target of the rule.

Magic Set _ATTACHTO_

A magic set containing the single tag (_ATTACHTO_). This set and tag will only match when the currently active cohort is the mark set with A.

Unification

Tag Unification

Each time a rule is run on a reading, the tag that first satisfied the set must be the same as all subsequent matches of the same set in tests.

A set is marked as a tag unification set by prefixing $$ to the name when used in a rule. You can only prefix existing sets; inline sets in the form of $$(tag tags) will not work, but $$Set + $$OtherSet will; that method will make 2 unification sets, though.

The regex tags <.*>r ".*"r "<.*>"r are special and will unify to the same exact tag of that type. This is useful for e.g. mandating that the baseform must be exactly the same in all places.

For example

```
LIST ROLE = <human> <anim> <inanim> (<bench> <table>) ;
SELECT $$ROLE (-1 KC) (-2C $$ROLE) ;
```

which would logically be the same as

```
SELECT (<human>) (-1 KC) (-2C (<human>)) ;
SELECT (<anim>) (-1 KC) (-2C (<anim>)) ;
SELECT (<inanim>) (-1 KC) (-2C (<inanim>)) ;
SELECT (<bench> <table>) (-1 KC) (-2C (<bench> <table>)) ;
```

Caveat: The exploded form is not identical to the unified form. Unification rules are run as normal rules, meaning once per reading. The exploded form would be run in-order as seperate rules per reading. There may be side effects due to that.

Caveat 2: The behavior of this next rule is undefined:

```
SELECT (tag) IF (0 $$UNISET) (-2* $$UNISET) (1** $$UNISET) ;
```

Since the order of tests is dynamic, the unification of $$UNISET will be initialized with essentially random data, and as such cannot be guaranteed to unify properly. Well defined behavior can be enforced in various ways:

```
# Put $$UNISET in the target
SELECT (tag) + $$UNISET IF (-2* $$UNISET) (1** $$UNISET) ;

# Only refer to $$UNISET in a single linked chain of tests
SELECT (tag) IF (0 $$UNISET LINK -2* $$UNISET LINK 1** $$UNISET) ;

# Use rule option KEEPORDER
SELECT KEEPORDER (tag) IF (0 $$UNISET) (-2* $$UNISET) (1** $$UNISET) ;
```

Having the unifier in the target is usually the best way to enforce behavior.

Top-Level Set Unification

Each time a rule is run on a reading, the top-level set that first satisfied the match must be the same as all subsequent matches of the same set in tests.

A set is marked as a top-level set unification set by prefixing && to the name when used in a rule. You can only prefix existing sets; inline sets in the form of &&(tag tags) will not work, but &&Set + &&OtherSet will; that method will make 2 unification sets, though.

For example

```
LIST SEM-HUM = <human> <person> <sapien> ;
LIST SEM-ANIM = <animal> <beast> <draconic> ;
LIST SEM-INSECT = <insect> <buzzers> ;
SET SEM-SMARTBUG = SEM-INSECT + (<sapien>) ;
SET SAME-SEM = SEM-HUM OR SEM-ANIM + SEM-SMARTBUG ; # During unification, OR
SELECT &&SAME-SEM (-1 KC) (-2C &&SAME-SEM) ;
```

which would logically be the same as

```
SELECT SEM-HUM (-1 KC) (-2C SEM-HUM) ;
SELECT SEM-ANIM (-1 KC) (-2C SEM-ANIM) ;
SELECT SEM-SMARTBUG (-1 KC) (-2C SEM-SMARTBUG) ;
```

Note that the unification only happens on the first level of sets, hence named top-level unification. Note also that the set operators in the prefixed set are ignored during unification.

The same caveats as for Tag Unification apply.

Chapter 17. Tags

First some example tags as we know them from CG-2 and VISLCG:

```
"<wordform>"
"baseform"
<W-max>
ADV
@error
(<civ> N)
```

Now some example tags as they may look in VISL CG-3:

```
"<Wordform>"i
"^[Bb]ase.*"r
<W>65>
(<F>=15> <F<=30>)
!ADV
^<dem>
(N <civ>)
```

Tag Order

Starting with the latter, (N <civ>), as this merely signifies that tags with multiple parts do not have to match in-order; (N <civ>) is the same as (<civ> N). This is different from previous versions of CG, but I deemed it unncecessary to spend extra time checking the tag order when hash lookups can verify the existance so fast. A side effect of this means there is no intersection set operator, nor need of one.

Literal String Modifiers

The first two additions to the feature sheet all display what I refer to as literal string modifiers, and there are two of such: 'i' for case-insensitive, and 'r' for a regular expression match. Using these modifiers will significantly slow down the matching as a hash lookup will no longer be enough. You can combine 'ir' for case-insensitive regular expressions. Regular expressions are evaluated via ICU, so their documentation is a good source. Regular expressions may also contain groupings that can later be used in variable string tags (see below).

Due to tags themselves needing the occasional escaping, regular expressions need double-escaping of some special symbols. E.g. grouping with () needs to be written as "a\(b|c\)d"r, while literal non-grouping () need to be written as "a\\(b\\)c"r. Metacharacters also need double-escaping, so \w needs to be written as \\w.

This will not work for wordforms used as the first qualifier of a rule, e.g:

```
"<wordform>"i SELECT (tag) ;
```

but those can be rewritten in a form such as

```
SELECT ("<wordform>"i) + (tag) ;
```

which will work, but be slightly slower.

Variable Strings

Variable string tags contain markers that are replaced with matches from the previously run grouping regular expression tag. Regular expression tags with no groupings will not have any effect on this behavior. Time also has no effect, so one could theoretically perform a group match in a previous rule and use the results later, though that would be highly unpredictable in practice.

Variable string tags are in the form of "string"v, "<string>"v, and <string>v, where variables matching $1 through $9 will be replaced with the corresponding group from the regular expression match. Multiple occurances of a single variable is allowed, so e.g. "$1$2$1"v would contain group 1 twice.

One can also manipulate the case of the resulting tag via %U, %u, %L, and %l. %U upper-cases the entire following string. %u upper-cases the following single letter. %L lower-cases the entire following string. %l lower-cases the following single letter. The case folding is performed right-to-left one-by-one.

It is also possible to include references to unified $$sets or &&sets in {} where they will be replaced with the tags that the unification resulted in. If there are multiple tags, they will be delimited by an underscore _.

It should be noted that you can use varstring tags anywhere, not just when manipulating tags. When used in a contextual test they are fleshed out with the information available at the time and then attempted matched.

```
# Adds a lower-case <wordform> to all readings.
ADD (<%L$1>v) TARGET ("<(.*)>"r) ;

# Adds a reading with a normalized baseform for all suspicious wordforms ending
APPEND ("$1y"v N P NOM) TARGET N + ("<(.*)ies>"r) IF (1 VFIN) ;

# Merge results from multiple unified $$sets into a single tag
LIST ROLE = human anim inanim (bench table) ;
LIST OTHER = crispy waffles butter ;
MAP (<{$$ROLE}/{$$OTHER}>v) (target tags) (-1 $$OTHER) (-2C $$ROLE) ;
```

Numerical Matches

Then there are the numerical matches, e.g. <W>65>. This will match tags such as <W:204> and <W=156> but not <W:32>. The second tag, (<F>15> <F<30>), matches values 15>F>30. These constructs are also slower than simple hash lookups.

The two special values MIN and MAX (both case-sensitive) will scan the cohort for their respective minimum or maximum value, and use that for the comparison. Internally, MIN is equal to -2147483648 and MAX is 2147483647, and using those values will also act such.

```
# Select the maximum value of W. Readings with no W will also be removed.
SELECT (<W=MAX>) ;

# Remove the minimum F. Readings with no F will not be removed.
REMOVE (<N=MIN>) ;
```

Table 17.1. Valid Operators

Operator	Meaning
=	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
<>	Not equal to

Anywhere that an = is valid you can also use : for backwards compatibility.

Table 17.2. Comparison Truth Table

A	B	Result
= x	= y	True if x = y
= x	!= y	True if x != y
= x	< y	True if x < y
= x	> y	True if x > y
= x	<= y	True if x <= y
= x	>= y	True if x >= y
< x	!= y	Always true
< x	< y	Always true
< x	> y	True if x > y
< x	<= y	Always true
< x	>= y	True if x > y
> x	!= y	Always true
> x	> y	Always true
> x	<= y	True if x < y
> x	>= y	Always true
<= x	!= y	Always true
<= x	<= y	Always true
<= x	>= y	True if x >= y
>= x	!= y	Always true
>= x	>= y	Always true
!= x	!= y	True if x = y

Tag Inversion

You can negate a tag by prepending a !, as in !ADV. This has the effect of matching if some tag other than ADV exists in the reading, but won't match if ADV is actually present. This is mostly useful in tags such as (V TR !ADV) as !ADV alone would match quite a lot.

Global Variables

Global variables are manipulated with rule types SETVARIABLE and REMVARIABLE, plus the stream commands SETVAR and REMVAR. Global variables persist until unset and are not bound to any window, cohort, or reading.

You can query a global variable with the form VAR:name (implemented) or query whether a variable has a specific value with VAR:name=value (not implemented just yet).

```
      REMOVE (@poetry) IF (0 (VAR:news)) ;
      SELECT (<historical>) IF (0 (VAR:year=1764)) ;
```

I recommend keeping VAR tags in the contextual tests, since they cannot currently be cached and will be checked every time.

Fail-Fast Tag

A Fail-Fast tag is the ^ prefix, such as ^<dem>. This will be checked first of a set and if found will block the set from matching, regardless of whether later independent tags could match. It is mostly useful for sets such as LIST SetWithFail = (N <bib>) (V TR) ^<dem>. This set will never match a reading with a <dem> tag, even if the reading matches (V TR).

Chapter 18. Sub-Readings

Sub-readings introduce a bit of hierarchy into readings, letting a reading have a hidden reading attached to it, which in turn may have another hidden reading, and so on. See the `test/T_SubReading_Apertium` and `test/T_SubReading_CG` tests for usage examples.

Apertium Format

The Apertium stream format supports sub-readings via the + delimiter for readings. E.g.

```
^word/aux3<tag>+aux2<tag>+aux1<tag>+main<tag>$
```

is a cohort with 1 reading which has a three level deep sub-reading. The order of which is the primary reading vs. sub-readings depends on the grammar SUBREADINGS setting:

```
SUBREADINGS = RTL ; # Default, right-to-left
SUBREADINGS = LTR ; # Alternate, left-to-right
```

In default RTL mode, the above reading has the primary reading `"main"` with sub-reading `"aux1"` with sub-reading `"aux2"` and finally sub-reading `"aux3"`.

In LTR mode, the above reading has the primary reading `"aux3"` with sub-reading `"aux2"` with sub-reading `"aux1"` and finally sub-reading `"main"`.

CG Format

The CG stream format supports sub-readings via indentation level. E.g.

```
"<word>"
  "main" tag
    "aux1" tag
      "aux2" tag
        "aux3" tag
```

is a cohort with 1 reading which has a three level deep sub-reading. Unlike the Apertium format, the order is strictly defined by indentation and cannot be changed. The above reading has the primary reading `"main"` with sub-reading `"aux1"` with sub-reading `"aux2"` and finally sub-reading `"aux3"`.

The indentation level is detected on a per-cohort basis. All whitespace counts the same for purpose of determining indentation, so 1 tab is same as 1 space is same as 1 no-break space and so on. Since it is per-cohort, it won't matter if previous cohorts has a different indentation style, so it is safe to mix cohorts from multiple sources.

Grammar Syntax

Working with sub-readings involves 2 new grammar features: Rule Option SUB:N and Contextual Option /N.

Rule Option SUB:N

Rule option SUB:N tells a rule which sub-reading it should operate on and which it should test as target. The N is an integer in the range -2^31 to 2^31. SUB:0 is the primary reading and same as not specifying SUB. Positive numbers refer to sub-readings starting from the primary and going deeper, while negative numbers start from the last sub-reading and go towards the primary. Thus, SUB:-1 always refers to the deepest sub-reading.

Given the above CG input and the rules

```
ADD SUB:-1 (mark) (*) ;
ADD SUB:1 (twain) (*) ;
```

the output will be

```
"<word>"
  "main" tag
    "aux1" tag twain
      "aux2" tag
        "aux3" tag mark
```

Note that SUB:N also determines which reading is looked at as target, so it will work for all rule types.

Contextual Option /N

Context option /N tests the N'th sub-reading of the currently active reading, where N follows the same rules as for SUB:N above. The /N must be last in the context position.

Given the above CG input and the rules

```
ADD (mark) (*) (0/-1 ("aux3")) ; # matches 3rd sub-reading "aux3"
ADD (twain) (*) (0/1 ("aux1")) ; # matches 1st sub-reading "aux1"
ADD (writes) (*) (0/1 ("main")) ; # won't match as 1st sub-reading doesn't ha
```

the output will be

```
"<word>"
  "main" tag mark twain
    "aux1" tag
      "aux2" tag
        "aux3" tag
```

Chapter 19. Binary Grammars

Security of Binary vs. Textual

Textual grammars are the human readable plain-text input grammars as described in the rest of this manual. Binary grammars are the compiled versions of textual grammars.

For the purpose of commercial distribution, textual grammars are terribly insecure; anyone can take your carefully constructed grammar and use it as a basis of their own work. Binary grammars are more secure in that it takes an active decompilation to get anything human readable, and even then it will be undocumented and look rather different from the original grammar.

As of release 0.8.9.3142, VISL CG-3 can create and use binary grammars. At this point they are basically memory dumps of the compiled grammars and can be trivially decompiled by modifying the sources a bit. Future releases will have options to strenghten the security of the binary grammars by excluding disused parts from the compilation.

Loading Speed of Binary Grammars

As of release 0.9.7.5729, the speed difference between binary and textual is down to factor 2 slower, where previously it was factor 10. So there is no longer any pressing need to use binary grammars solely for their speed advantage.

How to...

To compile a textual grammar to a binary form, use

```
vislcg3 --grammar inputgrammar.txt --grammar-only --grammar-bin binarygrammar.c
```

(remember to set codepage if needed)

To later on use the binary grammar, simply do

```
vislcg3 --grammar binarygrammar.cg3b
```

VISL CG-3 will auto-detect binary grammars in that the first 4 bytes of the file are ['C','G','3','B']. Binary grammars are neutral with regard to codepage and system endianness; to maximize portability of grammars, strings are stored in UTF-8 and all integers are normalized.

Incompatibilities

vislcg / bincg / gencg

Binary grammars generated with the older 'gencg' tool are not compatible with VISL CG-3 and there are no plans to make them loadable. Nor are binary grammars generated with VISL CG-3 usable with the older 'bincg'.

--grammar-info, --grammar-out, --statistics

Since binary grammars cannot be written back out in textual form, the command line options --grammar-info, --grammar-out, and --statistics will not work in binary mode.

Chapter 20. External Callbacks and Processors

The EXTERNAL rule types spawn and pass a window to an external process, and expect a modified window in return. The external process is spawned only once the first time the rule hits, is then sent an initial message of the current protocol version, and subseqently passed only windows. It is expected that the reply from a window is another window, or a null reply if there are no changes.

What follows is a description of the protocol using C++ structs. For those who want a more hands-on example, the source tree scripts/CG3_External.pm and scripts/external.pl and test/T_External/* is a working example of calling a Perl program that simply adds a tag to every reading.

All datatypes correspond to the C and C++ <stdint.h> types, and no endianness conversion is performed; it is assumed that the external program is native to the same arch as CG-3 is running on. Fields marked 'const' denote they may not be changed by the external. Notice that you may not change the number of cohorts or readings, but you may change the number of tags per reading.

Protocol Datatypes

```
uint32_t protocol_version = 7226;
uint32_t null_response = 0;

struct Text {
  uint16_t length;
  char *utf8_bytes;
};

enum READING_FLAGS {
  R_WAS_MODIFIED = (1 << 0),
  R_INVISIBLE    = (1 << 1),
  R_DELETED      = (1 << 2),
  R_HAS_BASEFORM = (1 << 3),
};

struct Reading {
  uint32_t reading_length; // sum of all data here plus data from all tags
  uint32_t flags;
  Text *baseform; // optional, depends on (flags & R_HAS_BASEFORM)

  uint32_t num_tags;
  Text *tags;
};

enum COHORT_FLAGS {
  C_HAS_TEXT   = (1 << 0),
  C_HAS_PARENT = (1 << 1),
};

struct Cohort {
  uint32_t cohort_length; // sum of all data here plus data from all readings
  const uint32_t number; // which cohort is this
  uint32_t flags;
  uint32_t parent; // optional, depends on (flags & C_HAS_PARENT)
  Text wordform;
  const uint32_t num_readings;
  Reading *readings;

  Text *text; // optional, depends on (flags & C_HAS_TEXT)
};

struct Window {
  uint32_t window_length; // sum of all data here plus data from all cohorts
  const uint32_t number; // which window is this

  const uint32_t num_cohorts;
  Cohort *cohorts;
};
```

Protocol Flow

1. Initial packet is simply the protocol_version. This is used to detect when an external may be out of date. If an external cannot reliably handle a protocol, I recommend that it terminates to avoid subtle bugs. Protocol version is only sent once and no response is allowed.

2. Every time an EXTERNAL rule hits, a Window is sent. If you make no changes to the window, send a null_response. If you do change the window, you must compose a whole Window as response. If you change anything in a Reading, you must set the R_WAS_MODIFIED flag on the Reading. If you change a Cohort's wordform, that automatically sets the R_WAS_MODIFIED flags on all Readings. You must send some response for every window.

3. When CG-3 is shutting down, the final thing it sends to the external is a null_reponse. Use this to clean up if necessary. Any data output after the null_response is ignored.

Chapter 21. Input Stream Commands

These are commands that exist in the input stream. They must be alone and at the beginning of the line to be respected. They will also be passed along to the output stream.

Exit

When encountered, this command will halt input. No further input will be read, no output will be written, and the existing buffers will be silently discarded. After that, the process will exit. If you were considering using Exit, take a look at whether Ignore would suit your needs better. *It is strongly recommended to precede an Exit with a Flush.*

```
<STREAMCMD:EXIT>
```

Flush

When encountered, this command will process all windows in the current buffer before continuing. This means that any window spanning contextual tests would not be able to see beyond a FLUSH point.

```
<STREAMCMD:FLUSH>
```

Ignore

When encountered, this command will cause all subsequent input to be passed along to the output stream without any disambiguation, until it finds a Resume command. Useful for excluding parts of the input if it differs in e.g. genre or language. *It is strongly recommended to precede an Ignore with a Flush.*

```
<STREAMCMD:IGNORE>
```

Resume

When encountered, this command resume disambiguation. If disambiguation is already in progress, it has no effect.

```
<STREAMCMD:RESUME>
```

Set Variable

Sets global variables. Same effect as the SETVARIABLE rule type, but applicable for a whole parsing chain. Takes a comma-separated list of variables to set, where each variable may have a value.

```
<STREAMCMD:SETVAR:poetry,year=1764>
<STREAMCMD:SETVAR:news>
```

```
<STREAMCMD:SETVAR:greek=ancient>
```

Unset Variable

Unsets global variables. Same effect as the REMVARIABLE rule type, but applicable for a whole parsing chain. Takes a comma-separated list of variables to unset.

```
<STREAMCMD:REMVAR:poetry,year>
<STREAMCMD:REMVAR:news>
<STREAMCMD:REMVAR:greek>
```

Chapter 22. FAQ & Tips & Tricks

FAQ

How far will a `(*-1C A)` test scan?

The CG-2 spec dictates that for a test `(*-1C A)`: "There is a cohort to the left containing 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." ...meaning scanning stops at the first A regardless of whether it is carefully A. To scan further than the first A you must use **.

VISLCG2 was not compliant with that and would scan until it found a "careful A". This caused the need for ugly hacks such as `(*1C A BARRIER A)` to emulate the correct behavior.

See this reference thread.

How can I match the tag `*` from my input?

You can't. The single `*` symbol is reserved for many special meanings in CG-3. I suggest replacing it with `**` or `<*>` or anything that isn't a single `*` if you need to work with it in CG.

Tricks

Determining whether a cohort has (un)ambiguous base forms

If you for whatever reason need to determine whether a cohort has readings with (un)ambiguous base forms, the following is how:

```
LIST bform = ".*"r ;

# Determines ambiguous base forms
ADD (@baseform-diff) $$bform (0 (*) - $$bform) ;

# ...so NEGATE to determine unambigious base forms
ADD (@baseform-same) $$bform (NEGATE 0 (*) - $$bform) ;
```

Attach all cohorts without a parent to the root

A final cleanup step of many dependency grammars is to attach anything that was not assigned a parent to the root of the window. This can be done easily with:

```
# For all cohorts that has no parent, attach to 0th cohort
SETPARENT (*) (NEGATE p (*)) TO (@0 (*)) ;
```

Use multiple cohorts as a barrier

The BARRIER and CBARRIER behavior may only refer to a single cohort, but often you want to stop because of a condition expressed in multiple cohorts. This can be solved in a flat manner via MAP, ADD, or SUBSTITUTE.

```
ADD (¤list) Noun (1 Comma) ;
SELECT Noun (-1* Adj BARRIER (¤list)) ;
```

Add a delimiting cohort

ADDCOHORT can be used to add any type of cohort, including delimiter ones. But it will not automatically delimit the window at such a cohort, so you need to add a DELIMIT rule after if that is your intended outcome. Just be mindful that DELIMIT will restart the grammar, so ADDCOHORT may fire again causing an endless loop; break it by conditioning ADDCOHORT, such as

```
ADDCOHORT ("§") AFTER (@title-end) IF (NOT 1 (<<<)) ;
DELIMIT _S_DELIMITERS_ ; # Use the magic set that contains what DELIMITERS defi
```

Chapter 23. Constraint Grammar Glossary

The terms in this glossary are specific to the workings and concepts of constraint grammar; internal and otherwise.

Baseform

The first part of a reading. Every reading must have one of these. It is usually the base form of the word in the containing cohort. Other than that, it is a tag like any other.

```
"defy"
```

Cohort

A single word with all its readings.

```
"<defiable>"
  "defy" adjective
  "defy" adverb
  "defy" verb plural
```

Contextual Target

The part of a dependency rule that locates a suitable cohort to attach to. It is no different from a normal contextual test and can have links, barriers, etc. *While you technically can NEGATE the entire contextual target, it is probably not a good idea to do so.*

```
(-1* Verb LINK 1* Noun)
```

Contextual Test

The part of a disambiguation rule that looks at words surrounding the active cohort. A rule will not act unless all its contextual tests are satisfied.

```
(-1* Verb LINK 1* Noun)
```

Dependency

A tag in #X->Y or #X#Y format denoting that this cohort is the child of another cohort.

Disambiguation Window

An array of cohorts as provided by the input stream. Usually the last cohort in a window is one from the DELIMITERS definition. CG-3 can keep several windows in memory at any time in order to fascilitate cross-window scanning from contextual tests. It is also possible to break a window into smaller windows on-the-fly with a DELIMIT rule.

Mapping Tag

A special type of tag, as defined by the mapping prefix. If a reading contains more than one mapping tag, the reading is multiplied into several readings with one of the mapping tags each and all the normal tags copied.

Mapping Prefix

A single unicode character. If a tag begins with this character, it is considered a mapping tag. The character can be changed with the grammar keyword MAPPING-PREFIX or the command line flag --prefix. Defaults to @.

Reading

Part of a cohort. Defines one variant of the word in question. A reading must begin with a baseform. Readings are the whole basis for how CG operates; they are what we disambiguate between and test against. The goal is to exit with one per cohort.

```
  "defy" adjective
```

Rule

The primary workhorses of CG. They can add, remove, alter readings, cohorts, and windows based on tests and type.

```
ADD (tags) targetset (-1* ("someword")) ;
SELECT targetset (-1* ("someword") BARRIER (tag)) ;
DELIMIT targetset (-1* ("someword")) ;
```

Set

A list of tags or a combination of other sets. Used as targets for rules and contextual tests.

```
LIST SomeSet = tags moretags (etc tags) ;
LIST OtherSet = moretags (etc tags) ;
SET CombiSet = SomeSet + OtherSet - (tag) ;
```

Tag

Any simple string of text. Readings and sets are built from tags.

```
"plumber" noun @jobtitle
```

Wordform

The container word of a cohort. Every cohort has exactly one of these. It is usually the original word of the input text.

```
"<defiable>"
```

Chapter 24. Constraint Grammar Keywords

You should avoid using these keywords as set names or similar.

ADD

Singles out a reading from the cohort that matches the target, and if all contextual tests are satisfied it will add the listed tags to the reading. Unlike MAP it will not block further MAP, ADD, or REPLACE rules from operating on the reading.

```
ADD (tags) targetset (-1* ("someword")) ;
```

ADDCOHORT

Inserts a new cohort before or after the target.

```
ADDCOHORT ("<wordform>" "baseform" tags) BEFORE (@waffles) ;
ADDCOHORT ("<wordform>" "baseform" tags) AFTER (@waffles) ;
```

ADDRELATION

ADDRELATION creates a one-way named relation from the current cohort to the found cohort. The name must be an alphanumeric string with no whitespace.

```
ADDRELATION (name) targetset (-1* ("someword"))
  TO (1* (@candidate)) (2 SomeSet) ;
```

ADDRELATIONS

ADDRELATIONS creates two one-way named relation; one from the current cohort to the found cohort, and one the other way. The names can be the same if so desired.

```
ADDRELATIONS (name) (name) targetset (-1* ("someword"))
  TO (1* (@candidate)) (2 SomeSet) ;
```

AFTER-SECTIONS

Same as SECTION, except it is only run a single time per window, and only after all normal SECTIONs have run.

ALL

An inline keyword put at the start of a contextual test to mandate that all cohorts found via the dependency or relation must match the set. This is a more readable way of saying 'C'.

```
SELECT targetset (ALL c (tag)) ;
```

AND

Deprecated: use "LINK 0" instead. An inline keyword put between contextual tests as shorthand for "LINK 0".

APPEND

Singles out a reading from the cohort that matches the target, and if all contextual tests are satisfied it will create and append a new reading from the listed tags. *Since this creates a raw reading you must include a baseform in the tag list.*

```
APPEND ("baseform" tags) targetset (-1* ("someword")) ;
```

BARRIER

An inline keyword part of a contextual test that will halt a scan if the barrier is encountered. Only meaningful in scanning contexts.

```
SELECT targetset (-1* ("someword") BARRIER (tag)) ;
```

BEFORE-SECTIONS

Same as SECTION, except it is only run a single time per window, and only before all normal SECTIONs have run.

CBARRIER

Careful version of BARRIER. Only meaningful in scanning contexts. See CBARRIER.

```
SELECT targetset (-1* ("someword") CBARRIER (tag)) ;
```

CONSTRAINTS

Deprecated: use SECTION instead. A section of the grammar that can contain SELECT, REMOVE, and IFF entries.

COPY

Duplicates a reading and adds tags to it. If you don't want to copy previously copied readings, you will have to keep track of that yourself by adding a marker tag.

```
COPY (¤copy tags) TARGET (target) - (¤copy) ;
```

CORRECTIONS

Deprecated: use BEFORE-SECTIONS instead. A section of the grammar that can contain APPEND and SUBSTITUTE entries.

DELIMIT

If it finds a reading which satisfies the target and the contextual tests, DELIMIT will cut the disambituation window immediately after the cohort the reading is in. After delimiting in this manner, CG-3 will bail out and disambiguate the newly formed window from the start. *This should not be used instead of DELIMITERS unless you know what you are doing.*

```
DELIMIT targetset (-1* ("someword")) ;
```

DELIMITERS

Sets a list of hard delimiters. If one of these are found the disambuation window is cut immediately after the cohort it was found in. If no delimiters are defined or the window exceeds the hard limit (defaults to 500 cohorts), the window will be cut arbitarily. Internally, this is converted to the magic set _S_DELIMITERS_.

```
DELIMITERS = "<$.>" "<$?>" "<$!>" "<$:>" "<$\;>" ;
```

END

Denotes the end of the grammar. Nothing after this keyword is read. *Deprecated: Remnant from older versions of CG; the end of the file will do just fine.*

EXTERNAL

Opens up a persistent pipe to the program and passes it the current window.

```
EXTERNAL ONCE /usr/local/bin/waffles (V) (-1 N) ;
EXTERNAL ALWAYS program-in-path (V) (-1 N) ;
EXTERNAL ONCE "program with spaces" (V) (-1 N) ;
```

IF

Deprecated: has no effect in CG-3. An inline keyword put before the first contextual test of a rule.

IFF

Singles out a reading from the cohort that matches the target, and if all contextual tests are satisfied it will behave as a SELECT rule. If the tests are not satisfied it will behave as a REMOVE rule.

```
IFF targetset (-1* ("someword")) ;
```

INCLUDE

Loads and parses another grammar file as if it had been pasted in on the line of the INCLUDE statement.

```
  INCLUDE other-file-name ;
```

LINK

An inline keyword part of a contextual test that will chain to another contextual test if the current is satisfied. The chained contextual test will operate from the current position in the window, as opposed to the position of the original cohort that initiated the chain. The chain can be extended to any depth.

```
SELECT targetset (-1* ("someword") LINK 3 (tag)) ;
```

LIST

Defines a new set based on a list of tags.

```
LIST setname = tag othertag (mtag htag) ltag ;
```

MAP

Singles out a reading from the cohort that matches the target, and if all contextual tests are satisfied it will add the listed tags to the reading and block further MAP, ADD, or REPLACE rules from operating on the reading.

```
MAP (tags) targetset (-1* ("someword")) ;
```

MAPPINGS

Deprecated: use BEFORE-SECTIONS instead. A section of the grammar that can contain MAP, ADD, and REPLACE entries.

MAPPING-PREFIX

Defines the single prefix character that should determine whether a tag is considered a mapping tag or not. Defaults to @.

```
MAPPING-PREFIX = @ ;
```

MOVE

Moves cohorts and optionally all children of the cohort to a different position in the window.

```
MOVE targetset (-1* ("someword")) AFTER (1* ("buffalo")) (-1 ("water")) ;
MOVE WITHCHILD (*) targetset (-1* ("someword")) BEFORE (1* ("buffalo")) (-1 ("wat
MOVE targetset (-1* ("someword")) AFTER WITHCHILD (*) (1* ("buffalo")) (-1 ("wate
MOVE WITHCHILD (*) targetset (-1* ("someword")) BEFORE WITHCHILD (*) (1* ("buffal
```

NEGATE

An inline keyword put at the start of a contextual test to invert the combined result of all following contextual tests. Similar to, but not the same as, NOT.

```
SELECT targetset (NEGATE -1* ("someword") LINK NOT 1 (tag)) ;
```

NONE

An inline keyword put at the start of a contextual test to mandate that none of the cohorts found via the dependency or relation must match the set. This is a more readable way of saying 'NOT'.

```
SELECT targetset (NONE c (tag)) ;
```

NOT

An inline keyword put at the start of a contextual test to invert the result of it. Similar to, but not the same as, NEGATE.

```
SELECT targetset (NEGATE -1* ("someword") LINK NOT 1 (tag)) ;
```

NULL-SECTION

Same as SECTION, except it is not actually run. Used for containing ANCHOR'ed lists of rules that you don't want run in the normal course of rule application.

PREFERRED-TARGETS

If the preferred targets are defined, this will influence SELECT, REMOVE, and IFF rules. Normally, these rules will operate until one reading remains in the cohort. If there are preferred targets, these rules are allowed to operate until there are no readings left, after which the preferred target list is consulted to find a reading to "bring back from the dead" and pass on as the final reading to survive the round. *Due to its nature of defying the rule application order, this is bad voodoo. I recommend only using this if you know what you are doing. This currently has no effect in CG-3, but will in the future.*

```
PREFERRED-TARGETS = tag othertag etctag ;
```

REMCOHORT

If it finds a reading which satisfies the target and the contextual tests, REMCOHORT will remove the cohort and all its readings from the current disambiguation window.

```
REMCOHORT targetset (-1* ("someword")) ;
```

REMOVE

Singles out a reading from the cohort that matches the target, and if all contextual tests are satisfied it will delete the mached reading.

```
REMOVE targetset (-1* ("someword")) ;
```

REMRELATION

Destroys one direction of a relation previously created with either SETRELATION or SETRELATIONS.

```
REMRELATION (name) targetset (-1* ("someword"))
  TO (1* (@candidate)) (2 SomeSet) ;
```

REMRELATIONS

Destroys both directions of a relation previously created with either SETRELATION or SETRELATIONS.

```
REMRELATIONS (name) (name) targetset (-1* ("someword"))
  TO (1* (@candidate)) (2 SomeSet) ;
```

REPLACE

Singles out a reading from the cohort that matches the target, and if all contextual tests are satisfied it will remove all existing tags from the reading, then add the listed tags to the reading and block further MAP, ADD, or REPLACE rules from operating on the reading.

```
REPLACE (tags) targetset (-1* ("someword")) ;
```

SECTION

A section of the grammar that can contain all types of rule and set definition entries.

SELECT

Singles out a reading from the cohort that matches the target, and if all contextual tests are satisfied it will delete all other readings except the matched one.

```
SELECT targetset (-1* ("someword")) ;
```

SET

Defines a new set based on operations between existing sets.

```
SET setname = someset + someotherset - (tag) ;
```

SETCHILD

Attaches the matching reading to the contextually targetted cohort as the parent. The last link of the contextual test is used as target.

```
SETCHILD targetset (-1* ("someword"))
  TO (1* (step) LINK 1* (candidate)) (2 SomeSet) ;
```

SETPARENT

Attaches the matching reading to the contextually targetted cohort as a child. The last link of the contextual test is used as target.

```
SETPARENT targetset (-1* ("someword"))
  TO (1* (step) LINK 1* (candidate)) (2 SomeSet) ;
```

SETRELATION

Creates a one-way named relation from the current cohort to the found cohort. The name must be an alphanumeric string with no whitespace.

```
SETRELATION (name) targetset (-1* ("someword"))
```

```
    TO (1* (@candidate)) (2 SomeSet) ;
```

SETRELATIONS

Creates two one-way named relation; one from the current cohort to the found cohort, and one the other way. The names can be the same if so desired.

```
SETRELATIONS (name) (name) targetset (-1* ("someword"))
  TO (1* (@candidate)) (2 SomeSet) ;
```

SETS

Deprecated: has no effect in CG-3. A section of the grammar that can contain SET and LIST entries.

SOFT-DELIMITERS

Sets a list of soft delimiters. If a disambiguation window is approaching the soft-limit (defaults to 300 cohorts), CG-3 will begin to look for a soft delimiter to cut the window after. Internally, this is converted to the magic set _S_SOFT_DELIMITERS_.

```
SOFT-DELIMITERS = "<$,>" ;
```

STATIC-SETS

A list of set names that need to be preserved at runtime to be used with advanced variable strings.

```
STATIC-SETS = VINF ADV ;
```

SUBSTITUTE

Singles out a reading from the cohort that matches the target, and if all contextual tests are satisfied it will remove the tags from the search list, then add the listed tags to the reading. *No guarantee is currently made as to where the replacement tags are inserted, but in the future the idea is that the tags will be inserted in place of the last found tag from the search list. This is a moot point for CG-3 as the tag order does not matter internally, but external tools may expect a specific order.*

```
SUBSTITUTE (search tags) (new tags) targetset (-1* ("someword")) ;
```

SWITCH

Switches the position of two cohorts in the window.

```
SWITCH targetset (-1* ("someword")) WITH (1* ("buffalo")) (-1 ("water")) ;
```

TARGET

Deprecated: has no effect in CG-3. An inline keyword put before the target of a rule.

TEMPLATE

Sets up templates of alternative contextual tests which can later be referred to by multiple rules or templates.

```
TEMPLATE name = (1 (N) LINK 1 (V)) OR (-1 (N) LINK 1 (V)) ;
TEMPLATE other = (T:name LINK 1 (P)) OR (1 (C)) ;
SELECT (x) IF ((T:name) OR (T:other)) ;
```

TO

An inline keyword put before the contextual target of a SETPARENT or SETCHILD rule.

UNMAP

Removes the mapping tag of a reading and lets ADD and MAP target the reading again. By default it will only act if the cohort has exactly one reading, but marking the rule UNSAFE lets it act on multiple readings.

```
UNMAP (TAG) ;
UNMAP UNSAFE (TAG) ;
```

Chapter 25. Drafting Board

Things that are planned but not yet implemented or not yet fully functional.

MATCH

```
[wordform] MATCH <target> [contextual_tests] ;
```

Used for tracing and debugging, it will not alter the readings but will gather information on why it matched, specifically what cohorts that fulfilled the tests. Those numbers will be output in a format yet to be determined. Something along the lines of M:548:1,3,4;2,1;9,9 where the numbers are absolute offsets within the window, first cohort being the 1st one (0th cohort is the magic >>>). As such, 1,3,4 would denote that the first contextual test looked at cohorts 1, 3, and 4 to validate itself, but not at cohort 2 (this can easily happen if you have a (2 ADV LINK 1 V) test or similar). This reveals a great deal about how the rule works.

EXECUTE

```
[wordform] EXECUTE <anchor_name> <anchor_name> <target> [contextual_tests] ;
```

These rules will allow you to mark named anchors and jump to them based on a context. In this manner you can skip or repeat certain rules. JUMP will jump to a location in the grammar and run rules from there till the end (or another JUMP which sends it to a different location), while EXECUTE will run rules in between the two provided ANCHORs and then return to normal.

References

Karlsson, Fred, Atro Voutilainen, Juha Heikkilä, and Arto Anttila, editors. 1995. *"Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text" (Natural Language Processing, No 4)*. Mouton de Gruyter, Berlin and New York. ISBN 3-11-014179-5.

Pasi Tapanainen. 1996. *The Constraint Grammar Parser CG-2*. Number 27 in publications. Department of General Linguistics, University of Helsinki.

Index