

CDG Reference Manual
0.95

Generated by Doxygen 1.3.8

Wed Oct 20 17:42:59 2004

Contents

1	The CDG Reference Manual	1
1.1	Purpose of this document	1
1.2	Overview	1
1.3	General guidelines	3
2	CDG Module Index	5
2.1	CDG Modules	5
3	CDG Data Structure Index	7
3.1	CDG Data Structures	7
4	CDG Page Index	9
4.1	CDG Related Pages	9
5	CDG Module Documentation	11
5.1	Cdg - The Root Module	11
5.2	Cdgdb - Get lexical entries out of a berkeley database	21
5.3	Chunker - Interface to a Chunking Parser	27
5.4	Command - The CDG Scripting Language	41
5.5	Compiler - compile a constraint grammar	59
5.6	Constraintnet - maintainance of constraint nets	81
5.7	Eval - routines to evaluate constraint formulas	100
5.8	HookCore - C Part in the core system	116
5.9	Hook - A callback system	127
5.10	Lexemgraph - maintainance of lexem graphs	128
5.11	Scache - Cache structures for binary LV scores	142
5.12	Scorematrix - The matrix of scores appearing in each ConstraintEdge	145
5.13	Skel - A Skeleton Module	148
5.14	Timer - timekeeping functions	151
5.15	HookBindings - Adaptor to the callback system	155

6	CDG Data Structure Documentation	161
6.1	ApproverStruct Struct Reference	161
6.2	BadnessStruct Struct Reference	163
6.3	ChunkerStruct Struct Reference	165
6.4	ChunkStruct Struct Reference	168
6.5	Command Struct Reference	170
6.6	CompilerStruct Struct Reference	172
6.7	ConstraintEdgeStruct Struct Reference	175
6.8	ConstraintNetStruct Struct Reference	177
6.9	ConstraintNodeStruct Struct Reference	180
6.10	ConstraintViolationStruct Struct Reference	182
6.11	entryStruct Struct Reference	184
6.12	GraphemNodeStruct Struct Reference	185
6.13	HookResultStruct Struct Reference	187
6.14	HookStruct Struct Reference	189
6.15	LexemGraphStruct Struct Reference	191
6.16	LexemNodeStruct Struct Reference	194
6.17	LoggerStruct Struct Reference	196
6.18	MakeInfoStruct Struct Reference	197
6.19	MyExportedTypeStruct Struct Reference	199
6.20	MyPrivateTypeStruct Struct Reference	200
6.21	NodeBindingStruct Struct Reference	201
6.22	ScoreCacheStruct Struct Reference	202
6.23	ScoreMatrixStruct Struct Reference	204
6.24	SMAEntryStruct Struct Reference	206
7	CDG Page Documentation	207
7.1	Todo List	207

Chapter 1

The CDG Reference Manual

Author:

Michael Daum, Ingo Schröder, Kilian A. Foth

1.1 Purpose of this document

This document is a thorough description of all data structures and functions used in the CDG system and serves as a technical documentation of the parser.

Most of the core functionality is exported by the modules in the directory `libcdg`, which can be viewed as yet another C library, although a very specialized one. Application-specific functionality must be implemented by the application that uses the `libcdg` library. Thus, the command-line parser `cdgp` includes code to parse its command-line options and to do I/O on a terminal, and the graphical parser `xcdg` defines extensive routines to display input structures graphically. This document will eventually cover all data structures and functions of the constraint parsing library `libcdg` and the text-based parser `cdgp`.

CDG is written entirely in C (and automatically generated C in some cases). It relies on several non-ANSI extensions to the C programming language (such as local functions and dynamic arrays). The GNU C compiler can compile all of these features; others may or may not.

1.2 Overview

The CDG library is organized into several modules each of which offers a specialized service to the environment. The set of all modules can be further divided into those offering the basic infrastructure for CDG parsing whereas others are build on top and offer the different parsing flavours that are available.

1.2.1 Basic Modules

- [Cdg - The Root Module](#): This the root layer of the CDG library.
- [Chunker - Interface to a Chunking Parser](#): This module offers an interface for an external chunker.
- [Cd gdb - Get lexical entries out of a Berkeley database](#): This module provides access to a Berkeley database for retrieving lexical entries.
- [Command - The CDG Scripting Language](#): This module implements the CDG scripting interface.

- [Compiler - compile a constraint grammar](#): This module implements a compiler in order to translate constraints into C code.
- [Constraintnet - maintenance of constraint nets](#): Herein the basic and most central datastructure of CDG is implemented, the ConstraintNet.
- [Eval - routines to evaluate constraint formulas](#): The Eval module actually implements the constraint interpreter
- [ref Functions](#): Implementation of all function calls
- [Hook - A callback system](#): The Hook module connects the core of the library to its outside world by a callback mechanism.
- [ref Input](#): This module defines the data structures and access functions for constraint grammar.
- [ref Interpreter](#): This module implements the constraint grammar language it consists of the following submodules
- [ref Levelvalue](#)
- [Lexemgraph - maintenance of lexem graphs](#)
- [ref Parse](#)
- [ref Predicates](#): Implementation of all predicates
- [Scache - Cache structures for binary LV scores](#)
- [Scorematrix - The matrix of scores appearing in each ConstraintEdge](#): This module implements the score matrix for a constraint edge.
- [ref Set](#): This module provides a uniform interface for all user-visible variables.
- [ref Statistics](#)
- [ref Tagger](#)
- [ref Testing](#)
- [Timer - timekeeping functions](#)
- [ref Variables](#): This module provides a generic interface for variable encapsulation, and is exclusively used by the set module.
- [ref Write](#)

1.2.2 Parsing Flavours

- [ref Arcconsistency](#)
- [ref Frobbing](#)
- [ref MGLS](#)
- [ref Increment](#)
- [ref Incrementalcompletion](#)
- [ref Netsearch](#)
- [ref Nodeconsistency](#)
- [ref Search](#)

1.3 General guidelines

1.3.1 File structure

Every module consists of a declaration part `<moduleName>.h` and an implementation part `<moduleName>.c`. Another module can use the exported services of this modules by including its declarations in `<moduleName>.h`. Note that many of the functions covered in this manual are not exported and therefore cannot be used from other modules at all. The files `skel.c` and `skel.h` provide a skeleton for a new module.

1.3.2 Terms used in this document

While some of the functions in the `libcdg` library explicitly use variable-length argument lists, other have a prototype like the following:

```
int no, char **args
```

In this case it is always assumed that `args` is an array of valid zero-terminated strings, and that `no` specifies the number of these strings. This is an alternate way of passing a varying number of additional arguments. The strings contained in the array are called *command* words in this document to distinguish them from the actual function arguments.

1.3.3 Coding style

Most identifiers of data structures and algorithms are complete English phrases such as ‘ConstraintNet’ or ‘printLexiconItem’. When a module deals primarily with one data structure, it is common practice to abbreviate the name of this structure and use it as a prefix to all exported identifiers, as in ‘lvNew’, ‘lvPrint’, and ‘lvDelete’ (rather than ‘newLevelValue’, ‘printLevelValue’, and ‘deleteLevelValue’).

To avoid the explicit use of pointer variables, most modules export `typedef` ‘d aliases for all pointer types. These aliases have meaningful names, while the underlying `struct` types have names ending in `Struct`:

```
typedef struct {
    String id;
    Boolean active;
    int counter;
} SectionStruct;
typedef SectionStruct *Section;
```

Although ANSI C allows the programmer to collapse these two `typedef` statements into one, this construct cannot be parsed by the interface generator SWIG, used for building `xcdg`. Therefore it is important to use exactly this way of defining pointers to structures.

One consequence of this definition style is that although very few pointer symbols are used in the code, most `libcdg` data types obey reference semantics: A function called on a variable `s` can usually change the underlying structure, even though C function calls actually use value semantics.

A more practical demonstration of the coding style is given in [Skel - A Skeleton Module](#).

Chapter 2

CDG Module Index

2.1 CDG Modules

Here is a list of all modules:

Cdg - The Root Module	11
Cd gdb - Get lexical entries out of a berkeley database	21
Chunker - Interface to a Chunking Parser	27
Command - The CDG Scripting Language	41
Compiler - compile a constraint grammar	59
Constraintnet - maintenance of constraint nets	81
Eval - routines to evaluate constraint formulas	100
Hook - A callback system	127
HookCore - C Part in the core system	116
HookBindings - Adaptor to the callback system	155
Lexemgraph - maintenance of lexem graphs	128
Scache - Cache structures for binary LV scores	142
Scorematrix - The matrix of scores appearing in each ConstraintEdge	145
Skel - A Skeleton Module	148
Timer - timekeeping functions	151

Chapter 3

CDG Data Structure Index

3.1 CDG Data Structures

Here are the data structures with brief descriptions:

ApproverStruct	161
BadnessStruct	163
ChunkerStruct	165
ChunkStruct	168
Command	170
CompilerStruct	172
ConstraintEdgeStruct	175
ConstraintNetStruct	177
ConstraintNodeStruct	180
ConstraintViolationStruct	182
entryStruct	184
GraphemNodeStruct	185
HookResultStruct	187
HookStruct	189
LexemGraphStruct	191
LexemNodeStruct	194
LoggerStruct	196
MakeInfoStruct	197
MyExportedTypeStruct	199
MyPrivateTypeStruct	200
NodeBindingStruct	201
ScoreCacheStruct	202
ScoreMatrixStruct	204
SMAEntryStruct	206

Chapter 4

CDG Page Index

4.1 CDG Related Pages

Here is a list of all related documentation pages:

Todo List	207
-----------	-----

Chapter 5

CDG Module Documentation

5.1 Cdg - The Root Module

5.1.1 Detailed Description

Initialization of the CDG system on startup.

This is the so called root module of the library with the following purpose:

- initialization of the CDG system at startup
- finalization at shutdown
- container for globally used functions and definitions
- owner of the symbol table

All other modules depend on this module and include [cdg.h](#)

Forward type definitions

This set of declarations help to deal with circular C-structures which introduce circular header file dependencies as a consequence. All type definitions of that kind are moved to this place here to get a one-stop forward declaration.

- typedef [ChunkStruct](#) * [Chunk](#)
- typedef [ChunkerStruct](#) * [Chunker](#)
- typedef [ChunkerStruct](#) [ChunkerStruct](#)
- typedef [ChunkStruct](#) [ChunkStruct](#)
- typedef [GraphemNodeStruct](#) * [GraphemNode](#)
- typedef [GraphemNodeStruct](#) [GraphemNodeStruct](#)
- typedef [LexemNodeStruct](#) * [LexemNode](#)
- typedef [LexemNodeStruct](#) [LexemNodeStruct](#)

Defines

- #define [FALSE](#) 0
- #define [max](#)(a, b) (a) > (b) ? (a) : (b)
- #define [min](#)(a, b) (a) < (b) ? (a) : (b)
- #define [NULL](#) 0
- #define [TRUE](#) 1

Functions

- void [cdgAgInsert](#) (Agenda a, double score, Pointer state)
- void [cdgDeleteComputed](#) (void)
- void [cdgExecPragmas](#) (List pragmas)
- void [cdgFinalize](#) (void)
- void [cdgFreeString](#) (String str)
- void [cdgInitialize](#) (void)

Variables

- Boolean [cdgCtrlCAllowed](#)
- Boolean [cdgCtrlCAllowed](#) = FALSE
- Boolean [cdgCtrlCTrapped](#)
- Boolean [cdgCtrlCTrapped](#) = FALSE
- Boolean [cdgEncodeUmlauts](#)
- Boolean [cdgEncodeUmlauts](#) = FALSE
- Hashtable [cdgNets](#)
- Hashtable [cdgNets](#)
- Hashtable [cdgParses](#)
- Hashtable [cdgParses](#)
- Hashtable [cdgProblems](#)
- Hashtable [cdgProblems](#)
- unsigned long [cdgTimeLimit](#)
- unsigned long [cdgTimeLimit](#) = 0
- String [cdgUser](#)
- String [cdgUser](#) = NULL
- Boolean [cdgXCDG](#)
- Boolean [cdgXCDG](#) = FALSE

5.1.2 Define Documentation

5.1.2.1 #define FALSE 0

this is false. FALSE has the numeric value 0. Definition at line 215 of file cdg.h.

Referenced by [bCompare\(\)](#), [chunkerCommandValidate\(\)](#), [chunkerNew\(\)](#), [cmdActivate\(\)](#), [cmdAnno2Parse\(\)](#), [cmdAnnos2prolog\(\)](#), [cmdAnnotation\(\)](#), [cmdChunk\(\)](#), [cmdCompareParses\(\)](#), [cmdCompile\(\)](#), [cmdConstraint\(\)](#), [cmdDeactivate\(\)](#), [cmdDistance\(\)](#), [cmdEdges\(\)](#), [cmdFrobbing\(\)](#), [cmdHelp\(\)](#), [cmdHierarchy\(\)](#), [cmdHook\(\)](#), [cmdIncrementalCompletion\(\)](#), [cmdInputwordgraph\(\)](#), [cmdISearch\(\)](#),

cmdLevel(), cmdLevelsort(), cmdLexicon(), cmdLoad(), cmdLs(), cmdNet(), cmdNetdelete(), cmdNetsearch(), cmdNewnet(), cmdNonSpecCompatible(), cmdParsedelete(), cmdParses2prolog(), cmdPrintParse(), cmdPrintParses(), cmdQuit(), cmdRenewnet(), cmdReset(), cmdSection(), cmdSet(), cmdShift(), cmdShowlevel(), cmdTagger(), cmdTesting(), cmdUseconstraint(), cmdUselevel(), cmdUseLexicon(), cmdVerify(), cmdWeight(), cmdWordgraph(), cmdWriteAnno(), cmdWriteNet(), cmdWriteParses(), cmdWriteWordgraph(), cnBuildEdges(), cnBuildFinal(), cnBuildIter(), cnBuildLv(), cnBuildTriple(), cnBuildUpdateArcs(), cnCompareViolation(), cnOptimizeNet(), cnOptimizeNode(), cnRenew(), cnTag(), cnUndeleteAllLVs(), comApprove(), comCompareAllLvPairs(), comCompareAllLVs(), comCompareLVs(), comCompareNets(), comCompareWithContext(), comCompile(), comInitGrammar(), comMake(), commandEval(), comNew(), compareChunks(), comPrint(), comReturnTypeInfoFunction(), comTranslateAbs(), comTranslateArithmetics(), comTranslateConstraint(), comTranslateDistance(), comTranslateEquation(), comTranslateHas(), comTranslateLookup(), comTranslateMatch(), comTranslateMinMax(), comTranslatePrint(), comTranslateTerm(), cvCompare(), cvCompareNatural(), cvContains(), dbAvailable(), dbClose(), dbLoad(), dbLoadAll(), dbLoadEntries(), dbOpen(), dbOpenIndexFile(), evalBinary(), evalConstraint(), evalFormula(), evalInitialize(), evalTerm(), evalValidateEvalMethod(), hkFindNoOfHook(), hkInitialize(), hkValidate(), initChunker(), initFakeChunker(), initRealChunker(), lgAreDeletedNodes(), lgCompatibleSets(), lgComputeNoOfPaths(), lgContains(), lgCopySelection(), lgForbiddenBy(), lgIntersectingSets(), lgIsEndNode(), lgIsStartNode(), lgLexemeInLexemNodeList(), lgMakePath(), lgMember(), lgNewFinal(), lgNewIter(), lgSimultaneous(), lgSpuriousUppercase(), lgSubset(), smNew(), timerSetAlarm(), and timerStopAlarm().

5.1.2.2 #define max(a, b) (a) > (b) ? (a) : (b)

maximum of a or b. Be aware that the expressions a and b might be evaluated twice depending on whether a or b are greater. Definition at line 183 of file cdg.h.

Referenced by cnBuildFinal(), cnPrint(), cnPrintInfo(), comConstraintDepth(), comFormulaDepth(), comFunctionDepth(), comMaxLookupStrings(), comMaxLookupStringsInFormula(), comPredicateDepth(), comTermDepth(), and lgNewIter().

5.1.2.3 #define min(a, b) (a) < (b) ? (a) : (b)

minimum of a or b. Be aware that the expressions a and b might be evaluated twice depending on whether a or b are smaller. Definition at line 190 of file cdg.h.

Referenced by cnPrint(), cnPrintInfo(), comIndent(), comOutdent(), and lgNewIter().

5.1.2.4 #define NULL 0

null, zero, nadda, nothing. NULL has the numeric value 0. Definition at line 199 of file cdg.h.

Referenced by annotation_completion_function(), cdgDeleteComputed(), cdgExecPragmas(), cdgInitialize(), cdgPrintf(), chunkerChunk(), chunkerCloneChunk(), chunkerCommandValidate(), chunkerInitialize(), chunkerNew(), cmdActivate(), cmdAnno2Parse(), cmdAnnos2prolog(), cmdAnnotation(), cmdChunk(), cmdCompareParses(), cmdConstraint(), cmdDeactivate(), cmdDistance(), cmdEdges(), cmdHelp(), cmdHierarchy(), cmdHook(), cmdIncrementalCompletion(), cmdInputwordgraph(), cmdISearch(), cmdLevel(), cmdLexicon(), cmdLoad(), cmdLs(), cmdNet(), cmdNetdelete(), cmdNetsearch(), cmdNewnet(), cmdNonSpecCompatible(), cmdParsedelete(), cmdParses2prolog(), cmdPrintParse(), cmdPrintParses(), cmdRenewnet(), cmdSection(), cmdSet(), cmdShowlevel(), cmdStatus(), cmdUseconstraint(), cmdUselevel(), cmdVerify(), cmdWordgraph(), cmdWriteAnno(), cmdWriteNet(), cmdWriteParses(), cmdWriteWordgraph(), cnBuild(), cnBuildEdges(), cnBuildFinal(), cnBuildInit(), cnBuildIter(), cnBuildLevelValues(), cnBuildLv(), cnBuildNodes(), cnBuildTriple(), cnBuildUpdateArcs(), cnDelete(), cnFindNode(), cnGetGraphemNodeFromArc(), cnInitialize(), cnOptimizeNode(), cnPrint(),

cnPrintEdge(), cnPrintInfo(), cnPrintNode(), cnPrintParses(), cnRenew(), cnTag(), cnUnaryPruning(), comApprove(), comCompareAllLvPairs(), comCompareAllLvs(), comCompareLvs(), comCompareNets(), comCompareWithContext(), comCompile(), comEscapeQuotes(), comFindComparableLv(), comFinitGrammar(), comFree(), comFreeApprover(), comFunctionDepth(), comIndexOfConstraint(), comIndexOfHierarchy(), comInitGrammar(), comInitialize(), comLoad(), comMake(), comMakeInfoNew(), command_completion_function(), commandEval(), commandLoop(), comMaxLookupStringsInTerm(), comNew(), comPredicateDepth(), comReturnTypeOfFunction(), comReturnTypeOfPredicate(), comReturnTypeOfTerm(), comTermDepth(), comTranslateBottomPeek(), comTranslateConstraint(), comTranslateExists(), comTranslateHeight(), comTranslateMinMax(), comTranslatePrint(), comTranslateTopPeek(), comWriteDeclarations(), comWriteHeader(), comWriteInitFunction(), constraint_completion_function(), cvAnalyse(), dbClose(), dbGetEntries(), dbLoad(), dbLoadAll(), dbLoadEntries(), dbOpen(), dbOpenCdgFile(), dbOpenIndexFile(), evalBinary(), evalBinaryConstraint(), evalChunker(), evalConstraint(), evalFormula(), evalHookHandle(), evalInContext(), evalInitialize(), evalUnary(), evalUnaryConstraint(), findChunk(), findGrapheme(), frob_method_completion_function(), getCategory(), getCategory(), getChunks(), getFakeChunks(), getFakeChunksAt(), getHookCmd(), getNextArgument(), getsHookHandle(), glsInteractionHookHandle(), gnClone(), hierarchy_completion_function(), hkInitialize(), hook_completion_function(), hooker_init(), ICinteractionHookHandle(), initHookResult(), interface_completion(), level_completion_function(), levelsort_completion_function(), lexicon_completion_function(), lgAreDeletableNodes(), lgAreDeletedNodes(), lgClone(), lgCompatibleNodes(), lgCompatibleSets(), lgComputeNoOfPaths(), lgCopySelection(), lgCopyTagScores(), lgDelete(), lgDeleteNodes(), lgDistanceOfNodes(), lgInitialize(), lgIntersectingSets(), lgIsEndNode(), lgIsStartNode(), lgLexemeInLexemNodeList(), lgMakePath(), lgMember(), lgMostProbablePath(), lgNew(), lgNewFinal(), lgNewInit(), lgNewIter(), lgPartitions(), lgQueryCat(), lgSimultaneous(), lgSubset(), lgUpdateArcs(), logFlush(), logWriteChar(), make_rl_string(), net_completion_function(), netsearchHookHandle(), newChunk(), newDbEntry(), parse_completion_function(), parseGetCategory(), parseGetGrapheme(), parseGetLevelValue(), parseGetRoots(), partialResultHookHandle(), peekValue(), printChunk(), progressHookHandle(), resetChunker(), resetHookHandle(), scInitialize(), search_method_completion_function(), section_completion_function(), set_completion_function(), setHookCmd(), smDelete(), smGetFlag(), smGetScore(), smSetAllFlags(), smSetFlag(), smSetScore(), tclHookHandle(), terminateChild(), timerFree(), timerSetAlarm(), timerStart(), unlock_tree(), word_completion_function(), and wordgraph_completion_function().

5.1.2.5 #define TRUE 1

this is true. TRUE has the numeric value 1. Definition at line 207 of file cdg.h.

Referenced by bCompare(), chunkerCommandValidate(), cmdActivate(), cmdAnno2Parse(), cmdAnnos2prolog(), cmdAnnotation(), cmdChunk(), cmdCompareParses(), cmdCompile(), cmdConstraint(), cmdDeactivate(), cmdDistance(), cmdEdges(), cmdFrobbing(), cmdHelp(), cmdHierarchy(), cmdHook(), cmdIncrementalCompletion(), cmdInputwordgraph(), cmdISearch(), cmdLevel(), cmdLevelsort(), cmdLexicon(), cmdLicense(), cmdLoad(), cmdLs(), cmdNet(), cmdNetsearch(), cmdNewnet(), cmdNonSpecCompatible(), cmdParsedelete(), cmdParses2prolog(), cmdPrintParse(), cmdPrintParses(), cmdQuit(), cmdRenewnet(), cmdReset(), cmdSection(), cmdShift(), cmdShowlevel(), cmdStatus(), cmdTagger(), cmdTesting(), cmdUseconstraint(), cmdUselevel(), cmdUseLexicon(), cmdVerify(), cmdVersion(), cmdWeight(), cmdWordgraph(), cmdWriteAnno(), cmdWriteNet(), cmdWriteParses(), cmdWriteWordgraph(), cnBuildEdges(), cnBuildIter(), cnBuildNodes(), cnBuildTriple(), cnBuildUpdateArcs(), cnCompareViolation(), cnDeleteAllLVs(), cnOptimizeNet(), cnOptimizeNode(), cnRenew(), cnTag(), cnUnaryPruning(), comApprove(), comCompareAllLvPairs(), comCompareAllLvs(), comCompareLvs(), comCompareNets(), comCompareWithContext(), comCompile(), comInitGrammar(), comMake(), commandEval(), commandLoop(), comNew(), compareChunks(), comPrintln(), comTranslate(), comTranslateAbs(), comTranslateArithmetics(), comTranslateBetween(), comTranslateConstraint(), comTranslateDistance(), comTranslateEquation(), comTranslateGuard(), comTranslateHas(), comTranslateMatch(), comTranslateMinMax(), comTranslatePrint(), comTranslateStartStop(), comTranslateSubsumes(), comTranslateUn-

Equation(), comWriteError(), comWriteWarning(), cvCompare(), cvCompareNatural(), cvContains(), dbAvailable(), dbClose(), dbLoad(), dbLoadAll(), dbLoadEntries(), dbOpen(), dbOpenIndexFile(), evalBinary(), evalBinaryConstraint(), evalConstraint(), evalFormula(), evalInContext(), evalInitialize(), evalUnaryConstraint(), evalValidateEvalMethod(), expire(), hkInitialize(), hkValidate(), initFakeChunker(), initRealChunker(), lgAreDeletableNodes(), lgAreDeletedNodes(), lgCompatibleSets(), lgComputeNoOfPaths(), lgContains(), lgCopySelection(), lgDeleteNode(), lgDeleteNodes(), lgForbiddenBy(), lgIntersectingSets(), lgLexemeInLexemNodeList(), lgMakePath(), lgMayModify(), lgNewFinal(), lgNewIter(), lgRequireLexeme(), lgRequireLexemes(), lgSimultaneous(), lgSpuriousUppercase(), and lgSubset().

5.1.3 Typedef Documentation

5.1.3.1 typedef `ChunkStruct*` `Chunk`

type of a chunk pointer Definition at line 237 of file cdg.h.

Referenced by chunkerChunk(), chunkerChunkDelete(), chunkerCloneChunk(), chunkerReplaceGraphemes(), chunkerStringOfChunkType(), cmpChunks(), compareChunks(), countChunks(), embedChunk(), evalChunker(), evalTerm(), findChunk(), getChunks(), getFakeChunksAt(), lgCopyTagScores(), mergeChunk(), newChunk(), postProcessChunks(), and printChunk().

5.1.3.2 typedef `ChunkerStruct*` `Chunker`

type of a chunker pointer Definition at line 231 of file cdg.h.

Referenced by chunkerChunk(), chunkerDelete(), chunkerNew(), cmdChunk(), cmpChunks(), cnTag(), embedChunk(), evalChunker(), getChunks(), getFakeChunks(), getFakeChunksAt(), getFakeChunkType(), initChunker(), initFakeChunker(), initRealChunker(), mergeChunk(), parseGetCategory(), parseGetGrapheme(), parseGetLabel(), parseGetLevelValue(), parseGetModifiee(), parseGetRoots(), postProcessChunks(), and resetChunker().

5.1.3.3 typedef struct `ChunkerStruct` `ChunkerStruct`

type of a chunker structure Definition at line 228 of file cdg.h.

5.1.3.4 typedef struct `ChunkStruct` `ChunkStruct`

type of a chunk structure Definition at line 234 of file cdg.h.

5.1.3.5 typedef `GraphemNodeStruct*` `GraphemNode`

type of a graphem node pointer Definition at line 249 of file cdg.h.

Referenced by chunkerChunk(), cmdDistance(), cmpGraphemes(), cnBuildIter(), cnBuildLevelValues(), cnBuildNodes(), cnBuildTriple(), cnBuildUpdateArcs(), cnGetGraphemNodeFromArc(), cnOptimizeNode(), computeNoOfPathsFromStart(), computeNoOfPathsToEnd(), findGrapheme(), getCategory(), getCategories(), getCategory(), getChunks(), getFakeChunksAt(), gnClone(), lgAreDeletableNodes(), lgClone(), lgComputeDistances(), lgComputeNoOfPaths(), lgContains(), lgCopyTagScores(), lgDelete(), lgIsEndNode(), lgIsStartNode(), lgMayModify(), lgMostProbablePath(), lgNewIter(), lgPartitions(), lgQueryCat(), parseGetGrapheme(), and printChunk().

5.1.3.6 typedef struct **GraphemNodeStruct** **GraphemNodeStruct**

type of a graphem node structure Definition at line 246 of file cdg.h.

5.1.3.7 typedef **LexemNodeStruct*** **LexemNode**

type of a lexem node pointer Definition at line 243 of file cdg.h.

Referenced by cmdDistance(), cnBuildNodes(), cnOptimizeNode(), cnPrintNode(), cnRenew(), cv-Analyse(), evalFormula(), evalTerm(), getCategories(), lgAreDeletableNodes(), lgAreDeletedNodes(), lg-Clone(), lgCompatibleNodes(), lgCompatibleSets(), lgComputeNoOfPaths(), lgCopySelection(), lgCopy-TagScores(), lgDelete(), lgDeleteNode(), lgDeleteNodes(), lgDistanceOfNodes(), lgForbiddenBy(), lg-IntersectingSets(), lgIsDeletedNode(), lgLexemeInLexemNodeList(), lgMakePath(), lgMayModify(), lg-Member(), lgMostProbablePath(), lgNewIter(), lgOverlap(), lgPartitions(), lgPrint(), lgPrintNode(), lg-QueryCat(), lgRequireLexeme(), lgRequireLexemes(), lgSimultaneous(), lgSubset(), lgWidth(), and parse-GetCategory().

5.1.3.8 typedef struct **LexemNodeStruct** **LexemNodeStruct**

type of a lexem node structure Definition at line 240 of file cdg.h.

5.1.4 Function Documentation

5.1.4.1 void **cdgAgInsert** (**Agenda *a***, **double *score***, **Pointer *state***) [inline]

helper function for agInsert. This function calls agInsert and emits truncation warnings using the cdgPrintf mechanism. Definition at line 140 of file cdg.c.

References CDG_WARNING, and cdgPrintf().

5.1.4.2 void **cdgDeleteComputed** (**void**)

Delete all existing constraint nets, parses and problems.

This should always be used after changing the constraint grammar in any way, because there is no guarantee that the previously computed structures still make sense with the new grammar definition. A constraint net might contain an LV that used to be allowed, but is now forbidden by a unary hard constraint so that it would not have been built at all under the new regime. Conversely, if the constraint that forbade building a particular LV were turned off, an existing constraint net would become incomplete, and solving it could miss a solution that would be possible under the new conditions. Definition at line 246 of file cdg.c.

References CDG_INFO, cdgExecHook(), cdgNets, cdgParses, cdgPrintf(), cdgProblems, cnDelete(), cn-MostRecentlyCreatedNet, ConstraintNet, HOOK_RESET, and NULL.

Referenced by cdgFinalize(), cmdActivate(), cmdDeactivate(), cmdLoad(), cmdReset(), cmd-Useconstraint(), and cmdUselevel().

5.1.4.3 void **cdgExecPragmas** (**List *pragmas***)

Execute all #pragma commands seen during last load operation.

The #pragma commands are stored in the input as they are seen during parsing, but they are not executed until after the load. There are two reasons for this:

1. a `#pragma` can refer to structures that are loaded from the same file as the `#pragma` – so executing it immediately would be too early.
2. if a load operation fails, it should not affect the state of the program at all. Definition at line 309 of file `cdg.c`.

References `commandEval()`, and `NULL`.

Referenced by `cmdLoad()`.

5.1.4.4 void `cdgFinalize` (void)

finalize the CDG library. When shutting down the system each module gets the chance to clean up things that might be worth it calling their finalization callback. Be aware that order of finalization might matter. So the hook module gets finalized at last. Definition at line 327 of file `cdg.c`.

References `cdgDeleteComputed()`, `cdgFreeString()`, `cdgUser`, `comFinalize()`, `dbFinalize()`, `evalFinalize()`, and `hkFinalize()`.

5.1.4.5 void `cdgFreeString` (String *str*)

wrapper for `strDelete` This function is our wrapper around the `strDelete()` of the `blah` lib. By default this is a NOP, that is: we don't call `strDelete()`. You might switch on the call to `strDelete()` at compile time, if you are brave enuf and wanna help to debug string programming. Definition at line 127 of file `cdg.c`.

Referenced by `cdgFinalize()`, `chunkerCommandValidate()`, `cnDelete()`, `comFree()`, `comMakeInfoFree()`, `commandEval()`, `evalFinalize()`, `getChunks()`, `interface_completion()`, and `resetChunker()`.

5.1.4.6 void `cdgInitialize` (void)

system initialization. This is the initialization of the root module that initializes all other modules containing an initialization callback. It is absolutely necessary to call this function before starting to use the CDG library. In general this is done by the `cdgp` commandline frontend or by the `tcl` and `perl` language extension while loading the bindings. When finishing the CDG library the cousin of this function - `cdgFinalize()` - is to be called. Definition at line 160 of file `cdg.c`.

References `cdgEncodeUmlauts`, `cdgNets`, `cdgParses`, `cdgProblems`, `cdgTimeLimit`, `cdgUser`, `chunkerInitialize()`, `cnInitialize()`, `comInitialize()`, `dbInitialize()`, `evalInitialize()`, `hkInitialize()`, `lgInitialize()`, `NULL`, `scInitialize()`, and `timerInitialize()`.

5.1.5 Variable Documentation

5.1.5.1 Boolean `cdgCtrlAllowed`

allow a `ctrl-c` event. This flag is used to allow the signal handler to indicate a `ctrl-c` event. If `TRUE` the signal handler will set `cdgCtrlTrapped`, if `FALSE` it will not. Definition at line 84 of file `cdg.c`.

Referenced by `cmdAnno2Parse()`, `cmdAnnotation()`, `cmdChunk()`, `cmdCompareParses()`, `cmdFrobbing()`, `cmdHierarchy()`, `cmdIncrementalCompletion()`, `cmdISearch()`, `cmdLexicon()`, `cmdNet()`, `cmdNetsearch()`, `cmdNewnet()`, `cmdPrintParse()`, `cmdPrintParses()`, `cmdShift()`, `cmdWordgraph()`, `cmdWriteNet()`, `cmdWriteParses()`, `cmdWriteWordgraph()`, `cnBuildFinal()`, and `comApprove()`.

5.1.5.2 Boolean `cdgCtrlAllowed` = FALSE

allow a ctrl-c event. This flag is used to allow the signal handler to indicate a ctrl-c event. If TRUE the signal handler will set `cdgCtrlTrapped`, if FALSE it will not. Definition at line 84 of file `cdg.c`.

Referenced by `cmdAnno2Parse()`, `cmdAnnotation()`, `cmdChunk()`, `cmdCompareParses()`, `cmdFrobbing()`, `cmdHierarchy()`, `cmdIncrementalCompletion()`, `cmdISearch()`, `cmdLexicon()`, `cmdNet()`, `cmdNetsearch()`, `cmdNewnet()`, `cmdPrintParse()`, `cmdPrintParses()`, `cmdShift()`, `cmdWordgraph()`, `cmdWriteNet()`, `cmdWriteParses()`, `cmdWriteWordgraph()`, `cnBuildFinal()`, and `comApprove()`.

5.1.5.3 Boolean `cdgCtrlTrapped`

indicator flag set by the signal handler of ctrl-c. If `cdgCtrlAllowed` is TRUE and the user presses ctrl-c the flag `cdgCtrlTrapped` is set to TRUE. It's the responsibility of the current function to re-act to the interruption. Definition at line 76 of file `cdg.c`.

Referenced by `cmdAnno2Parse()`, `cmdAnnotation()`, `cmdChunk()`, `cmdCompareParses()`, `cmdFrobbing()`, `cmdHierarchy()`, `cmdIncrementalCompletion()`, `cmdISearch()`, `cmdLexicon()`, `cmdNet()`, `cmdNetsearch()`, `cmdNewnet()`, `cmdPrintParse()`, `cmdPrintParses()`, `cmdShift()`, `cmdWordgraph()`, `cmdWriteNet()`, `cmdWriteParses()`, `cmdWriteWordgraph()`, `cnBuild()`, `cnBuildEdges()`, `cnBuildFinal()`, `cnBuildIter()`, `cnPrint()`, `comApprove()`, `comCompareAllLvPairs()`, `comCompareAllLvs()`, and `comCompareWithContext()`.

5.1.5.4 Boolean `cdgCtrlTrapped` = FALSE

indicator flag set by the signal handler of ctrl-c. If `cdgCtrlAllowed` is TRUE and the user presses ctrl-c the flag `cdgCtrlTrapped` is set to TRUE. It's the responsibility of the current function to re-act to the interruption. Definition at line 76 of file `cdg.c`.

Referenced by `cmdAnno2Parse()`, `cmdAnnotation()`, `cmdChunk()`, `cmdCompareParses()`, `cmdFrobbing()`, `cmdHierarchy()`, `cmdIncrementalCompletion()`, `cmdISearch()`, `cmdLexicon()`, `cmdNet()`, `cmdNetsearch()`, `cmdNewnet()`, `cmdPrintParse()`, `cmdPrintParses()`, `cmdShift()`, `cmdWordgraph()`, `cmdWriteNet()`, `cmdWriteParses()`, `cmdWriteWordgraph()`, `cnBuild()`, `cnBuildEdges()`, `cnBuildFinal()`, `cnBuildIter()`, `cnPrint()`, `comApprove()`, `comCompareAllLvPairs()`, `comCompareAllLvs()`, and `comCompareWithContext()`.

5.1.5.5 Boolean `cdgEncodeUmlauts`

If set, pseudo-umlaut sequences typed into the CDG shell will be turned into real 8bit umlauts: "a ==> ä
Definition at line 103 of file `cdg.c`.

Referenced by `cdgInitialize()`.

5.1.5.6 Boolean `cdgEncodeUmlauts` = FALSE

If set, pseudo-umlaut sequences typed into the CDG shell will be turned into real 8bit umlauts: "a ==> ä
Definition at line 103 of file `cdg.c`.

Referenced by `cdgInitialize()`.

5.1.5.7 Hashtable `cdgNets`

all existing ConstraintNets Definition at line 112 of file `cdg.c`.

Referenced by `cdgDeleteComputed()`, `cdgInitialize()`, `cmdEdges()`, `cmdISearch()`, `cmdNet()`, `cmdNetdelete()`, `cmdNetsearch()`, `cmdNewnet()`, `cmdStatus()`, `cmdWriteAnno()`, `cmdWriteNet()`, `cnFindNet()`, `comApprove()`, and `net_completion_function()`.

5.1.5.8 Hashtable `cdgNets`

all existing ConstraintNets Definition at line 112 of file `cdg.c`.

Referenced by `cdgDeleteComputed()`, `cdgInitialize()`, `cmdEdges()`, `cmdISearch()`, `cmdNet()`, `cmdNetdelete()`, `cmdNetsearch()`, `cmdNewnet()`, `cmdStatus()`, `cmdWriteAnno()`, `cmdWriteNet()`, `cnFindNet()`, `comApprove()`, and `net_completion_function()`.

5.1.5.9 Hashtable `cdgParses`

all existing Parses Definition at line 113 of file `cdg.c`.

Referenced by `cdgDeleteComputed()`, `cdgInitialize()`, `cmdCompareParses()`, `cmdParsedelete()`, `cmdParses2prolog()`, `cmdPrintParse()`, `cmdStatus()`, and `parse_completion_function()`.

5.1.5.10 Hashtable `cdgParses`

all existing Parses Definition at line 113 of file `cdg.c`.

Referenced by `cdgDeleteComputed()`, `cdgInitialize()`, `cmdCompareParses()`, `cmdParsedelete()`, `cmdParses2prolog()`, `cmdPrintParse()`, `cmdStatus()`, and `parse_completion_function()`.

5.1.5.11 Hashtable `cdgProblems`

all existing Problems Definition at line 114 of file `cdg.c`.

Referenced by `cdgDeleteComputed()`, `cdgInitialize()`, and `cmdNetdelete()`.

5.1.5.12 Hashtable `cdgProblems`

all existing Problems Definition at line 114 of file `cdg.c`.

Referenced by `cdgDeleteComputed()`, `cdgInitialize()`, and `cmdNetdelete()`.

5.1.5.13 unsigned long `cdgTimeLimit`

time limit in milliseconds used by `netsearch`, `frobbling` and `gls`. The value of `cdgTimeLimit` is normally used to install a timer alarm as in `timerSetAlarm(cdgTimeLimit)`; Definition at line 97 of file `cdg.c`.

Referenced by `cdgInitialize()`, `cmdFrobbling()`, and `cmdShift()`.

5.1.5.14 unsigned long `cdgTimeLimit = 0`

time limit in milliseconds used by `netsearch`, `frobbling` and `gls`. The value of `cdgTimeLimit` is normally used to install a timer alarm as in `timerSetAlarm(cdgTimeLimit)`; Definition at line 97 of file `cdg.c`.

Referenced by `cdgInitialize()`, `cmdFrobbling()`, and `cmdShift()`.

5.1.5.15 String `cdgUser`

user name currently running the CDG system. On startup the user name is stored into this place and propagated to newly computed parses. Definition at line 110 of file `cdg.c`.

Referenced by `cdgFinalize()`, and `cdgInitialize()`.

5.1.5.16 String `cdgUser` = NULL

user name currently running the CDG system. On startup the user name is stored into this place and propagated to newly computed parses. Definition at line 110 of file `cdg.c`.

Referenced by `cdgFinalize()`, and `cdgInitialize()`.

5.1.5.17 Boolean `cdgXCDG`

`xcdg` indicator flag. If TRUE we're running under XCDG. Definition at line 90 of file `cdg.c`.

Referenced by `cnBuildEdges()`, and `getNextArgument()`.

5.1.5.18 Boolean `cdgXCDG` = FALSE

`xcdg` indicator flag. If TRUE we're running under XCDG. Definition at line 90 of file `cdg.c`.

Referenced by `cnBuildEdges()`, and `getNextArgument()`.

5.2 Cdgdb - Get lexical entries out of a berkeley database

5.2.1 Detailed Description

Author:

Othello Maurer

Date:

unknown

This file provides an interface for retrieving lexical entries out of a berkeley database. The database file contains only references to an underlying cdg file which actually holds the entries. For loading the requested lexical entries, a temp file will be created out of the cdg file, containing only these entries. This file will then be parsed with **inpuLoad()** into the memory.

Data Structures

- struct [entryStruct](#)

Typedefs

- typedef [entryStruct](#) * [dbEntry](#)

Functions

- Boolean [dbAvailable](#) (void)
- Boolean [dbClose](#) (void)
- void [dbFinalize](#) (void)
- List [dbGetEntries](#) (String word)
- void [dbInitialize](#) (void)
- Boolean [dbLoad](#) (String key)
- Boolean [dbLoadAll](#) (List forms)
- Boolean [dbLoadEntries](#) (List entries)
- Boolean [dbOpen](#) (String name)
- Boolean [dbOpenCdgFile](#) (String filename)
- Boolean [dbOpenIndexFile](#) (String filename)
- [dbEntry](#) [newDbEntry](#) ()

Variables

- FILE * [cdgstream](#)
- DB * [database](#)
- time_t [dbAge](#)
- String [dbFileName](#)
- String [dbIndexName](#)
- Hashtable [done](#) = NULL
- String [tmpFilename](#) = NULL

5.2.2 Typedef Documentation

5.2.2.1 typedef `entryStruct*` `dbEntry`

a pointer to the `entryStruct` Definition at line 85 of file `cdgdb.c`.

Referenced by `dbGetEntries()`, `dbLoadEntries()`, and `newDbEntry()`.

5.2.3 Function Documentation

5.2.3.1 Boolean `dbAvailable` (void)

checks if there is a database handle available.

Returns:

TRUE if there is a database opened, else FALSE

Definition at line 228 of file `cdgdb.c`.

References `CDG_WARNING`, `cdgPrintf()`, `database`, `dbAge`, `dbClose()`, `dbFileName`, `dbIndexName`, `dbOpenCdgFile()`, `dbOpenIndexFile()`, `FALSE`, and `TRUE`.

Referenced by `cmdAnno2Parse()`, `dbFinalize()`, `dbLoad()`, and `dbLoadAll()`.

5.2.3.2 Boolean `dbClose` (void)

close the database. writes database to disk and closes it.

Returns:

TRUE if success, FALSE otherwise

Definition at line 160 of file `cdgdb.c`.

References `CDG_ERROR`, `cdgPrintf()`, `database`, `FALSE`, `NULL`, and `TRUE`.

Referenced by `cmdCloseDB()`, `dbAvailable()`, and `dbFinalize()`.

5.2.3.3 void `dbFinalize` (void)

finalize close the database, delete **done** hash, remove temporary file Definition at line 457 of file `cdgdb.c`.

References `dbAvailable()`, `dbClose()`, `done`, and `tmpFilename`.

Referenced by `cdgFinalize()`.

5.2.3.4 List `dbGetEntries` (String *word*)

gets lexical entries from the database. gets a result list of `LexiconItem` matching the given word (the key) from the database. This function succeeds only once for a given word form.

Parameters:

word the word which is the key

Returns:

a list of `dbEntry` struct, all of them having the same key and `NULL` if there was an Error

Definition at line 274 of file cdgdb.c.

References CDG_ERROR, cdgPrintf(), database, dbEntry, entryStruct::key, newDbEntry(), NULL, entryStruct::oBegin, and entryStruct::oEnd.

Referenced by dbLoad(), and dbLoadAll().

5.2.3.5 void dbInitialize (void)

Initialize Definition at line 447 of file cdgdb.c.

References done.

Referenced by cdgInitialize().

5.2.3.6 Boolean dbLoad (String key)

wrapper for [dbLoadEntries\(\)](#). generates a temp file which holds lexical entries and parses them into the memory

Parameters:

key the key of the lex entries which have to be parsed

Returns:

TRUE if success, else FALSE

Definition at line 385 of file cdgdb.c.

References CDG_INFO, cdgPrintf(), dbAvailable(), dbGetEntries(), dbLoadEntries(), done, FALSE, NULL, and TRUE.

5.2.3.7 Boolean dbLoadAll (List forms)

Another wrapper for [dbLoadEntries\(\)](#).

It loads all known lexicon items for words in the list FORMS. This has exactly the same effect as calling [dbLoad\(\)](#) repeatedly, but is faster because there will be only one file system operation instead of n. Definition at line 415 of file cdgdb.c.

References CDG_INFO, CDG_PROGRESS, cdgPrintf(), dbAvailable(), dbGetEntries(), dbLoadEntries(), done, FALSE, NULL, and TRUE.

Referenced by cmdAnno2Parse().

5.2.3.8 Boolean dbLoadEntries (List entries)

loads lexical entries into the memory. generates a temp file which holds lexical entries and parses them into the memory

Parameters:

entries a list of dbEntry structs you want to parse

Returns:

True if success, else FALSE

Definition at line 333 of file cdgdb.c.

References CDG_ERROR, CDG_INFO, cdgPrintf(), cdgstream, dbEntry, FALSE, hkVerbosity, NULL, entryStruct::oBegin, entryStruct::oEnd, tmpFilename, and TRUE.

Referenced by dbLoad(), and dbLoadAll().

5.2.3.9 Boolean dbOpen (String *name*)

Open the data base.

NAME is the base name of the cdg and the index file. Definition at line 182 of file cdgdb.c.

References CDG_ERROR, cdgPrintf(), dbFileName, dbIndexName, dbOpenCdgFile(), dbOpenIndexFile(), FALSE, NULL, and TRUE.

Referenced by cmdUseLexicon().

5.2.3.10 Boolean dbOpenCdgFile (String *filename*)

open the cdg file. opens the cdg file, which the database is indexing you have to ensure that it is the right one, otherwise you will get crap

Parameters:

filename the name of the cdg file

Returns:

TRUE if success, else FALSE

Definition at line 139 of file cdgdb.c.

References cdgstream, database, dbAge, dbFileName, and NULL.

Referenced by dbAvailable(), and dbOpen().

5.2.3.11 Boolean dbOpenIndexFile (String *filename*)

opens a database. this function opens a database, specified by the parameter 'filename' and stores the handle in the static variable 'database'. If there is no database file with this filename, a new database is created.

Parameters:

filename the filename of the database file

Returns:

TRUE if success, FALSE otherwise

Definition at line 100 of file cdgdb.c.

References CDG_ERROR, cdgPrintf(), database, dbIndexName, FALSE, NULL, and TRUE.

Referenced by dbAvailable(), and dbOpen().

5.2.3.12 **dbEntry** newDbEntry ()

constructor for dbEntry objects.

Returns:

a new dbEntry

Definition at line 257 of file cdgdb.c.

References dbEntry, entryStruct::key, NULL, entryStruct::oBegin, and entryStruct::oEnd.

Referenced by dbGetEntries().

5.2.4 Variable Documentation

5.2.4.1 FILE* **cdgstream** [static]

the underlying cdg file. This is actually the file, where the lexical entries are stored in. Definition at line 56 of file cdgdb.c.

Referenced by dbLoadEntries(), and dbOpenCdgFile().

5.2.4.2 DB* **database** [static]

database handle. The handle to the database struct Definition at line 52 of file cdgdb.c.

Referenced by dbAvailable(), dbClose(), dbGetEntries(), dbOpenCdgFile(), and dbOpenIndexFile().

5.2.4.3 time_t **dbAge** [static]

age of that file at the time of the last opening Definition at line 65 of file cdgdb.c.

Referenced by dbAvailable(), and dbOpenCdgFile().

5.2.4.4 String **dbFileName** [static]

file name of the file with CDG input in it Definition at line 59 of file cdgdb.c.

Referenced by dbAvailable(), dbOpen(), and dbOpenCdgFile().

5.2.4.5 String **dbIndexName** [static]

file name of the index Definition at line 62 of file cdgdb.c.

Referenced by dbAvailable(), dbOpen(), and dbOpenIndexFile().

5.2.4.6 Hashtable **done** = NULL [static]

table of words already loaded. Holds the word which are yet retrieved from the database, so that the database won't be questioned for them Definition at line 73 of file cdgdb.c.

Referenced by dbFinalize(), dbInitialize(), dbLoad(), and dbLoadAll().

5.2.4.7 String **tmpFilename** = NULL [static]

the filename of the temporary file. This temporary file will contain a set of lexical entries that can then be parsed into the memory. Definition at line 69 of file cdgdb.c.

Referenced by `dbFinalize()`, and `dbLoadEntries()`.

5.3 Chunker - Interface to a Chunking Parser

5.3.1 Detailed Description

Author:

Michael Daum (see also AUTHORS and THANKS for more)

Date:

2002-09-09

Id

[chunker.c](#),v 1.41 2004/09/01 13:42:31 micha Exp

This module offers an interface for an external chunker.

Id

[chunker.h](#),v 1.12 2004/09/01 13:44:12 micha Exp

Data Structures

- struct [ChunkerStruct](#)
- struct [ChunkStruct](#)

Enumerations

- enum [ChunkerMode](#) { [DefaultChunker](#), [FakeChunker](#), [RealChunker](#), [EvalChunker](#) }
- enum [ChunkType](#) {
 [NChunk](#), [PChunk](#), [VChunk](#), [NoChunk](#),
 [UnknownChunk](#) }

Functions

- List [chunkerChunk](#) ([Chunker](#) chunker)
- void [chunkerChunkDelete](#) ([Chunk](#) chunk)
- [ChunkType](#) [chunkerChunkTypeOfString](#) (String tag)
- [Chunk](#) [chunkerCloneChunk](#) ([Chunk](#) chunk)
- Boolean [chunkerCommandValidate](#) (String name, String value, String *var)
- void [chunkerDelete](#) ([Chunker](#) chunker)
- void [chunkerFinalize](#) (void)
- void [chunkerInitialize](#) (void)
- [Chunker](#) [chunkerNew](#) ([ChunkerMode](#) mode, [LexemGraph](#) lg)
- void [chunkerPrintChunks](#) (unsigned long mode, List chunks)
- void [chunkerReplaceGraphemes](#) ([Chunk](#) chunk, [LexemGraph](#) lg)
- String [chunkerStringOfChunkType](#) ([Chunk](#) chunk)
- Boolean [cmpArcs](#) (Arc arc1, Arc arc2)
- Boolean [cmpChunks](#) ([Chunk](#) c1, [Chunk](#) c2, [Chunker](#) chunker)
- Boolean [cmpGraphemes](#) ([GraphemNode](#) g1, [GraphemNode](#) g2)
- Boolean [compareChunks](#) ([Chunk](#) c1, [Chunk](#) c2)
- int [countChunks](#) (List chunks)

- `Chunk embedChunk` (`Chunker` chunker, `Chunk` target, `Chunk` source)
- `int evalChunker` (`Chunker` chunker, `List` annoChunks)
- `Chunk findChunk` (`List` chunks, `int` from, `int` to)
- `GraphemNode findGrapheme` (`LexemGraph` lg, `GraphemNode` gn)
- `List getCategoryes` (`GraphemNode` gn)
- `String getCategory` (`GraphemNode` gn)
- `List getChunks` (`Chunker` chunker)
- `List getFakeChunks` (`Chunker` chunker)
- `List getFakeChunksAt` (`Chunker` chunker, `Chunk` parent, `int` index)
- `ChunkType getFakeChunkType` (`Chunker` chunker, `int` index)
- `Boolean initChunker` (`Chunker` chunker)
- `Boolean initFakeChunker` (`Chunker` chunker)
- `Boolean initRealChunker` (`Chunker` chunker)
- `Chunk mergeChunk` (`Chunker` chunker, `Chunk` target, `Chunk` source)
- `Chunk newChunk` (`ChunkType` type)
- `String parseGetCategory` (`Chunker` chunker, `int` index)
- `GraphemNode parseGetGrapheme` (`Chunker` chunker, `int` index)
- `String parseGetLabel` (`Chunker` chunker, `int` index)
- `LevelValue parseGetLevelValue` (`Chunker` chunker, `int` index)
- `int parseGetModifiee` (`Chunker` chunker, `int` index)
- `List parseGetRoots` (`Chunker` chunker)
- `void postProcessChunks` (`Chunker` chunker, `List` chunks)
- `void printChunk` (`unsigned long` mode, `Chunk` chunk)
- `void resetChunker` (`Chunker` chunker)
- `int terminateChild` (`pid_t` pid)

Variables

- `char ** chunkerArgs` = NULL
- `String chunkerCommand` = NULL
- `ChunkerMode chunkerMode` = `RealChunker`
- `Boolean chunkerUseChunker` = FALSE

5.3.2 Enumeration Type Documentation

5.3.2.1 enum `ChunkerMode`

different modes the chunker can operate in.

Enumeration values:

DefaultChunker the globally set chunker type: one of the below

FakeChunker read chunks from the annotation

RealChunker call the real chunker

EvalChunker call the real chunker and compare it to the fake chunker

Definition at line 34 of file chunker.h.

5.3.2.2 enum `ChunkType`

chunk types.

Enumeration values:

- NChunk* a chunk of a nominal clause
- PChunk* a chunk of a prepositional clause
- VChunk* a chunk of a verbal clause
- NoChunk* things which never go into a chunk
- UnknownChunk* fallback

Definition at line 44 of file chunker.h.

Referenced by `chunkerChunkTypeOfString()`, and `getFakeChunkType()`.

5.3.3 Function Documentation

5.3.3.1 List `chunkerChunk` (`Chunker chunker`)

compute the chunks.

Parameters:

chunker the current chunker

Returns:

a list of chunks or NULL on failure.

The returned list is owned by the chunker and thus needs no extra deallocation besides chunker deletion. The list of computed chunks is then used to assert the chunking information into the LexemGraph lg. Definition at line 1553 of file chunker.c.

References `CDG_ERROR`, `CDG_INFO`, `cdgPrintf()`, `GraphemNodeStruct::chunk`, `Chunk`, `Chunker`, `chunkerChunkDelete()`, `chunkerCloneChunk()`, `chunkerReplaceGraphemes()`, `LexemGraphStruct::chunks`, `ChunkerStruct::chunks`, `evalChunker()`, `EvalChunker`, `FakeChunker`, `getChunks()`, `getFakeChunks()`, `GraphemNode`, `ChunkerStruct::lg`, `ChunkerStruct::mode`, `ChunkStruct::nodes`, `NULL`, `RealChunker`, and `ChunkStruct::subChunks`.

Referenced by `cmdChunk()`, and `cnTag()`.

5.3.3.2 void `chunkerChunkDelete` (`Chunk chunk`)

`chunkerChunkDelete`: destruct a chunk and all its subchunks.

parameters: `chunk` = a chunk to be deallocated Definition at line 504 of file chunker.c.

References `Chunk`, `chunkerChunkDelete()`, `ChunkStruct::nodes`, and `ChunkStruct::subChunks`.

Referenced by `chunkerChunk()`, `chunkerChunkDelete()`, `getFakeChunksAt()`, `lgDelete()`, and `resetChunker()`.

5.3.3.3 `ChunkType chunkerChunkTypeOfString` (`String tag`)

return the string representation of a chunk type. Definition at line 1513 of file chunker.c.

References ChunkType, NChunk, NoChunk, PChunk, and VChunk.

Referenced by getChunks().

5.3.3.4 **Chunk** chunkerCloneChunk (**Chunk** *chunk*)

chunkerCloneChunk: construct a copy of a given chunk including clones of subChunks.

parameters: chunk = the original returns: the copy. Definition at line 477 of file chunker.c.

References Chunk, chunkerCloneChunk(), ChunkStruct::from, ChunkStruct::head, newChunk(), ChunkStruct::nodes, NULL, ChunkStruct::parent, ChunkStruct::subChunks, ChunkStruct::to, and ChunkStruct::type.

Referenced by chunkerChunk(), chunkerCloneChunk(), lgCopyTagScores(), and mergeChunk().

5.3.3.5 **Boolean** chunkerCommandValidate (**String** *name*, **String** *value*, **String** * *var*)

validation command for [chunkerCommand](#).

Parameters:

name name of the variable (chunkerCommand in our case)

value the value to be set

var the address of a possibly converted value

Returns:

TRUE on success

Basically the chunkerCommand value is converted into [chunkerArgs](#) which are then used when actually forking the chunker command.

See also:

[chunkerInitialize](#), [chunkerArgs](#)

Definition at line 1803 of file chunker.c.

References CDG_ERROR, cdgFreeString(), cdgPrintf(), chunkerArgs, FALSE, NULL, and TRUE.

Referenced by chunkerInitialize().

5.3.3.6 **void** chunkerDelete (**Chunker** *chunker*)

chunkerDelete: destroy the chunker representation.

parameters: chunker = the object to be destructed Definition at line 439 of file chunker.c.

References Chunker, and resetChunker().

Referenced by cmdChunk(), and cnTag().

5.3.3.7 **void** chunkerFinalize (**void**)

finalize the chunker module.

This is called by [cdgFinalize](#). (No good module without a finalizer and a initializer.)

See also:

[chunkerInitialize](#)

Definition at line 1930 of file chunker.c.

5.3.3.8 void chunkerInitialize (void)

initialize the chunker module.

This is called only once by [cdgInitialize](#) when the application starts up.

See also:

[chunkerFinalize](#)

Definition at line 1908 of file chunker.c.

References [chunkerCommand](#), [chunkerCommandValidate\(\)](#), [chunkerMode](#), [chunkerUseChunker](#), [EvalChunker](#), [FakeChunker](#), [NULL](#), and [RealChunker](#).

Referenced by [cdgInitialize\(\)](#).

5.3.3.9 Chunker chunkerNew (ChunkerMode mode, LexemGraph lg)

construct a new chunker.

Parameters:

mode one of the defined [ChunkerMode](#)

lg data from which we're going to initialize

Returns:

a new chunker

In case of a [RealChunker](#) a child process is forked in the background specified by the [chunkerCommand](#). This function might return [NULL](#) when chunking is switched off, no [chunkerCommand](#) is defined or the initialization of the chunker object fails.

See also:

[initChunker](#), [initRealChunker](#), [initFakeChunker](#)

Definition at line 190 of file chunker.c.

References [ChunkerStruct::args](#), [CDG_ERROR](#), [cdgPrintf\(\)](#), [Chunker](#), [chunkerArgs](#), [chunkerCommand](#), [chunkerMode](#), [chunkerUseChunker](#), [ChunkerStruct::chunks](#), [DefaultChunker](#), [FakeChunker](#), [FALSE](#), [initChunker\(\)](#), [ChunkerStruct::lg](#), [ChunkerStruct::mode](#), [ChunkerStruct::nrLevels](#), [ChunkerStruct::nrWords](#), [NULL](#), [ChunkerStruct::parse](#), and [ChunkerStruct::pid](#).

Referenced by [cmdChunk\(\)](#), and [cnTag\(\)](#).

5.3.3.10 void chunkerPrintChunks (unsigned long mode, List chunks)

print the chunks of a lattice.

Parameters:

mode print mode, e.g. [CDG_INFO](#)

chunks the list of chunks to be printed

Definition at line 1495 of file chunker.c.

References `cdgPrintf()`, and `printChunk()`.

Referenced by `cmdChunk()`, `cnPrint()`, `cnTag()`, and `lgPrint()`.

5.3.3.11 void chunkerReplaceGraphemes (**Chunk** *chunk*, **LexemGraph** *lg*)

`chunkerReplaceGraphemes`: replace all grapheme references in a chunk with those given in a lexemgraph.

parameters: `chunk` = the structure using the arcs `lg` = the lexemgraph using equivalent arcs Definition at line 1771 of file chunker.c.

References `Chunk`, `findGrapheme()`, `ChunkStruct::from`, `ChunkStruct::head`, `ChunkStruct::nodes`, `ChunkStruct::subChunks`, and `ChunkStruct::to`.

Referenced by `chunkerChunk()`, and `lgCopyTagScores()`.

5.3.3.12 String chunkerStringOfChunkType (**Chunk** *chunk*)

return the string representation of a chunk type. Definition at line 1530 of file chunker.c.

References `Chunk`, `NChunk`, `PChunk`, `ChunkStruct::type`, and `VChunk`.

Referenced by `embedChunk()`, `evalTerm()`, `getChunks()`, `getFakeChunksAt()`, `mergeChunk()`, `postProcessChunks()`, and `printChunk()`.

5.3.3.13 Boolean cmpArcs (**Arc** *arc1*, **Arc** *arc2*) [static]

`cmpArcs`: return true if `arc1` starts before `arc2` Definition at line 525 of file chunker.c.

Referenced by `embedChunk()`.

5.3.3.14 Boolean cmpChunks (**Chunk** *c1*, **Chunk** *c2*, **Chunker** *chunker*) [static]

`cmpChunks`: return true if `c1` starts before `c2`. Definition at line 533 of file chunker.c.

References `GraphemNodeStruct::arc`, `Chunk`, `Chunker`, and `ChunkStruct::from`.

Referenced by `embedChunk()`, `getChunks()`, `getFakeChunks()`, `getFakeChunksAt()`, and `mergeChunk()`.

5.3.3.15 Boolean cmpGraphemes (**GraphemNode** *g1*, **GraphemNode** *g2*) [static]

`cmpGraphemes`: return true if `g1` start before `g2` Definition at line 517 of file chunker.c.

References `GraphemNodeStruct::arc`, and `GraphemNode`.

Referenced by `mergeChunk()`.

5.3.3.16 Boolean compareChunks (**Chunk** *c1*, **Chunk** *c2*) [static]

are two chunks isomorph.

Parameters:

c1 the one chunk
c2 the other chunk

Returns:

TRUE if the two chunks are equal, else FALSE

This is again a recursion where all sub chunks must match aswell. Definition at line 1621 of file chunker.c.

References GraphemNodeStruct::arc, Chunk, FALSE, ChunkStruct::from, ChunkStruct::subChunks, ChunkStruct::to, TRUE, and ChunkStruct::type.

Referenced by evalChunker().

5.3.3.17 int countChunks (List *chunks*) [static]

count the number of chunks.

Parameters:

chunks list of chunks to be counted

Returns:

amount of chunks

This not only counts the list length but also all sub chunks. Definition at line 1331 of file chunker.c.

References Chunk, NChunk, PChunk, ChunkStruct::subChunks, ChunkStruct::type, and VChunk.

Referenced by evalChunker().

5.3.3.18 Chunk embedChunk (Chunker *chunker*, Chunk *target*, Chunk *source*) [static]

embedChunk: embed chunk as a subchunk into the target chunk

parameters target = the resulting chunk source = the chunk to be embedded returns: the target chunk.
Definition at line 846 of file chunker.c.

References GraphemNodeStruct::arc, CDG_DEBUG, cdgPrintf(), Chunk, Chunker, chunkerStringOfChunkType(), cmpArcs(), cmpChunks(), ChunkStruct::from, ChunkStruct::nodes, ChunkStruct::subChunks, and ChunkStruct::to.

Referenced by getFakeChunksAt().

5.3.3.19 int evalChunker (Chunker *chunker*, List *annoChunks*) [static]

evaluate computed agains annotated chunks.

Parameters:

chunker the current chunker
annoChunks the list of chunks extracted from the annotation

Returns:

the number of errors

Definition at line 1655 of file chunker.c.

References GraphemNodeStruct::arc, CDG_INFO, CDG_WARNING, cdgPrintf(), Chunk, Chunker, ChunkerStruct::chunks, compareChunks(), countChunks(), findChunk(), ChunkStruct::from, NChunk, NoChunk, NULL, PChunk, printChunk(), ChunkStruct::to, ChunkStruct::type, and VChunk.

Referenced by chunkerChunk().

5.3.3.20 **Chunk** findChunk (List *chunks*, int *from*, int *to*) [static]

search the chunk that spans over the given indices.

Parameters:

- chunks* a list of chunks
- from* the starting point of the span
- to* the ending point of the span

Returns:

the found Chunk or NULL if the specified span has not been chunked so far

This is a recursive function:

- returns NULL if chunks is empty
- returns NULL if <from-to> is not spanned by any chunk
- returns chunk X if it spans exactly <from-to>
- if a chunk X spans more than <from-to>
 - return X if there's no exactly spanning sub chunk Y
 - or the found sub chunk Y inside X

Definition at line 1366 of file chunker.c.

References GraphemNodeStruct::arc, CDG_DEBUG, cdgPrintf(), Chunk, ChunkStruct::from, NULL, printChunk(), ChunkStruct::subChunks, and ChunkStruct::to.

Referenced by evalChunker(), and getChunks().

5.3.3.21 **GraphemNode** findGrapheme (LexemGraph *lg*, GraphemNode *old*) [static]

find an equivalent grapheme in a given lexemgraph.

parameters: lg = the lexemgraph arc = the arc (possibly not used in the lexemgraph) returns: an equivalent arc. Definition at line 1745 of file chunker.c.

References GraphemNodeStruct::arc, GraphemNode, LexemGraphStruct::graphemnodes, and NULL.

Referenced by chunkerReplaceGraphemes().

5.3.3.22 **List** getCategories (GraphemNode *gn*) [static]

getCategories: get all POS-tags of undeleted lexem nodes

parameter: gn = a lexem node returns: a list of POS tags. Definition at line 646 of file chunker.c.

References CDG_DEBUG, cdgPrintf(), GraphemNode, LexemGraphStruct::isDeletedNode, LexemNodeStruct::lexem, GraphemNodeStruct::lexemes, GraphemNodeStruct::lexemgraph, LexemNode, LexemNodeStruct::no, NULL, and LexemNodeStruct::tagscore.

Referenced by getCategory(), and printChunk().

5.3.3.23 String getCategory (GraphemNode gn) [static]

getCategory: get one POS-tag, warn if there are more than one

parameter: gn = a grapheme returns: the first POS-tag available. Definition at line 614 of file chunker.c.

References CDG_WARNING, cdgPrintf(), getCategoryes(), GraphemNode, and NULL.

Referenced by getChunks(), postProcessChunks(), and printChunk().

5.3.3.24 List getChunks (Chunker chunker) [static]

this is the entry function to the real chunker.

Parameters:

chunker the current chunker to be used

Returns:

a list of chunks

Writes to the chunker, reads from the chunker and builds a list of chunks. Definition at line 1152 of file chunker.c.

References GraphemNodeStruct::arc, CDG_DEBUG, CDG_ERROR, CDG_WARNING, cdgFreeString(), cdgPrintf(), Chunk, Chunker, chunkerChunkTypeOfString(), chunkerStringOfChunkType(), cmpChunks(), findChunk(), ChunkStruct::from, getCategory(), GraphemNode, LexemGraphStruct::graphemnodes, ChunkStruct::head, ChunkerStruct::lg, newChunk(), ChunkStruct::nodes, NULL, ChunkerStruct::pipe1, ChunkerStruct::pipe2, ChunkStruct::subChunks, and ChunkStruct::to.

Referenced by chunkerChunk().

5.3.3.25 List getFakeChunks (Chunker chunker) [static]

this is the entry function to the fake chunker.

Parameters:

chunker the current chunker to be used

Returns:

a list of chunks

Definition at line 1120 of file chunker.c.

References Chunker, cmpChunks(), getFakeChunksAt(), NULL, parseGetRoots(), and postProcessChunks().

Referenced by chunkerChunk().

5.3.3.26 List `getFakeChunksAt` (**Chunker** *chunker*, **Chunk** *parent*, **int** *index*) [static]

get the chunks under the given root node.

Parameters:

- chunker* the current chunker
- parent* dominating chunk
- index* index of the root node in the dependency tree

Definition at line 882 of file chunker.c.

References `GraphemNodeStruct::arc`, `CDG_DEBUG`, `cdgPrintf()`, `Chunk`, `Chunker`, `chunkerChunkDelete()`, `chunkerStringOfChunkType()`, `cmpChunks()`, `embedChunk()`, `ChunkStruct::from`, `getFakeChunkType()`, `GraphemNode`, `ChunkStruct::head`, `ChunkerStruct::mainlevel`, `mergeChunk()`, `NChunk`, `newChunk()`, `ChunkStruct::nodes`, `NULL`, `ChunkStruct::parent`, `ChunkerStruct::parse`, `parseGetGrapheme()`, `parseGetLabel()`, `PChunk`, `ChunkStruct::to`, `ChunkStruct::type`, `UnknownChunk`, and `VChunk`.

Referenced by `getFakeChunks()`.

5.3.3.27 **ChunkType** `getFakeChunkType` (**Chunker** *chunker*, **int** *index*) [static]

`getFakeChunkType` Definition at line 741 of file chunker.c.

References `Chunker`, `ChunkType`, `NChunk`, `NoChunk`, `parseGetCategory()`, `PChunk`, `UnknownChunk`, and `VChunk`.

Referenced by `getFakeChunksAt()`.

5.3.3.28 Boolean `initChunker` (**Chunker** *chunker*) [static]

initialize the chunker with the given data.

Parameters:

- chunker* the current chunker

Returns:

- true on success.

This is called from `chunkerChunk` and from `chunkerNew`. It calls `initFakeChunker` or `initRealChunker` depending on the chunker mode. Definition at line 240 of file chunker.c.

References `CDG_ERROR`, `cdgPrintf()`, `Chunker`, `EvalChunker`, `FakeChunker`, `FALSE`, `initFakeChunker()`, `initRealChunker()`, `ChunkerStruct::mode`, and `RealChunker`.

Referenced by `chunkerNew()`.

5.3.3.29 Boolean `initFakeChunker` (**Chunker** *chunker*) [static]

initialize a fake chunker with the given data.

Parameters:

- chunker* the current chunker

Returns:

true on success.

This function is called by [initChunker\(\)](#) whenever the given Chunker is in mode FakeChunker. In order to work properly the contained Lattice within the LexemGraph lg must have a corresponding Annotation from which the chunks can be faked.

See also:

also [initChunker](#), [initFakeChunker](#), [initRealChunker](#), [chunkerChunk](#).

Definition at line 272 of file chunker.c.

References CDG_ERROR, cdgPrintf(), Chunker, FALSE, LexemGraphStruct::lattice, ChunkerStruct::lg, ChunkerStruct::mainlevel, ChunkerStruct::nrLevels, ChunkerStruct::nrWords, ChunkerStruct::parse, resetChunker(), and TRUE.

Referenced by [initChunker\(\)](#).

5.3.3.30 Boolean initRealChunker (Chunker chunker) [static]

initialize a real chunker with the given data.

Parameters:

chunker the current chunker

Returns:

true on success.

This function is called from [initChunker](#) during the initialization of the chunker, that is just before it starts chunking in [chunkerChunk](#).

See also:

also [initChunker](#), [initFakeChunker](#), [initRealChunker](#), [chunkerChunk](#).

Definition at line 332 of file chunker.c.

References ChunkerStruct::args, CDG_DEBUG, CDG_ERROR, cdgPrintf(), Chunker, FALSE, ChunkerStruct::mainlevel, ChunkerStruct::pid, ChunkerStruct::pipe1, ChunkerStruct::pipe2, and TRUE.

Referenced by [initChunker\(\)](#).

5.3.3.31 Chunk mergeChunk (Chunker chunker, Chunk target, Chunk source) [static]

mergeChunk: add the source to the target chunk. the target chunk spans the words of both chunks.

parameters: chunker = the current chunker target = the resulting chunk source = the chunk to be added to the target

returns: the target chunk. Definition at line 805 of file chunker.c.

References GraphemNodeStruct::arc, CDG_DEBUG, cdgPrintf(), Chunk, Chunker, chunkerCloneChunk(), chunkerStringOfChunkType(), cmpChunks(), cmpGraphemes(), ChunkStruct::from, ChunkStruct::nodes, ChunkStruct::subChunks, and ChunkStruct::to.

Referenced by [getFakeChunksAt\(\)](#).

5.3.3.32 **Chunk** newChunk (**ChunkType** *type*) [static]

construct a new chunk and initialize it.

Parameters:

type one of the ChunkTypes NChunk, PChunk, ...

Returns:

a new empty Chunk.

Definition at line 454 of file chunker.c.

References Chunk, ChunkStruct::from, ChunkStruct::head, ChunkStruct::nodes, NULL, ChunkStruct::parent, ChunkStruct::subChunks, ChunkStruct::to, and ChunkStruct::type.

Referenced by chunkerCloneChunk(), getChunks(), and getFakeChunksAt().

5.3.3.33 **String** parseGetCategory (**Chunker** *chunker*, **int** *index*) [static]

parseGetCategory: get the POS-tag of a given word index

parameters: chunker = the current chunker index = index of a word in the parse.

returns: the POS-tag string or NULL if not defined Definition at line 694 of file chunker.c.

References Chunker, LexemNodeStruct::lexem, LexemNode, NULL, and parseGetLevelValue().

Referenced by getFakeChunkType().

5.3.3.34 **GraphemNode** parseGetGrapheme (**Chunker** *chunker*, **int** *index*) [static]

parseGetGrapheme: get the grapheme node of a given word index

parameters: chunker = the current chunker index = index of a word in the parse.

returns: the arc. Definition at line 720 of file chunker.c.

References CDG_ERROR, cdgPrintf(), Chunker, GraphemNode, NULL, and parseGetLevelValue().

Referenced by getFakeChunksAt().

5.3.3.35 **String** parseGetLabel (**Chunker** *chunker*, **int** *index*) [static]

parseGetLabel: get the label of the dependency of a word (on the main level)

parameters: chunker = the current chunker index = index of a word in the parse returns: the label of that dependency Definition at line 585 of file chunker.c.

References Chunker, ChunkerStruct::mainlevel, ChunkerStruct::nrLevels, and ChunkerStruct::parse.

Referenced by getFakeChunksAt().

5.3.3.36 **LevelValue** parseGetLevelValue (**Chunker** *chunker*, **int** *index*) [static]

parseGetLevelValue: get the dependency arc of a word.

parameters: chunker = the current chunker index = index of a word in the parse.

returns: the level value of this word Definition at line 600 of file chunker.c.

References Chunker, ChunkerStruct::mainlevel, NULL, and ChunkerStruct::parse.

Referenced by parseGetCategory(), and parseGetGrapheme().

5.3.3.37 int parseGetModifiee (Chunker chunker, int index) [static]

parseGetModifiee: get the word this one is modifying (on the main level)

parameters: chunker = the current chunker index = the modifier index in the word vector of the current parse returns: the modifiee index Definition at line 571 of file chunker.c.

References Chunker, ChunkerStruct::mainlevel, ChunkerStruct::nrLevels, and ChunkerStruct::parse.

Referenced by parseGetRoots().

5.3.3.38 List parseGetRoots (Chunker chunker) [static]

parseGetRoots: get all unbound words (on the main level)

parameters: chunker = the current chunker returns: a list of word indices or NULL if there are no root bindings (?)

Note: you become the owner of the returned list container, so deallocate it after you've consumed the result. Definition at line 548 of file chunker.c.

References Chunker, ChunkerStruct::nrWords, NULL, and parseGetModifiee().

Referenced by getFakeChunks().

5.3.3.39 void postProcessChunks (Chunker chunker, List inputList) [static]

postProcessChunks: get rid of unwanted chunks.

parameters: inputList = items to be filtered Definition at line 1091 of file chunker.c.

References GraphemNodeStruct::arc, CDG_DEBUG, cdgPrintf(), Chunk, Chunker, chunkerStringOfChunkType(), ChunkStruct::from, getCategory(), NoChunk, ChunkStruct::to, ChunkStruct::type, and UnknownChunk.

Referenced by getFakeChunks().

5.3.3.40 void printChunk (unsigned long mode, Chunk chunk) [static]

printChunk: print a single chunk and all its subchunks

parameters: mode = print mode, e.g. CDG_INFO chunk = the chunk to be printed Definition at line 1423 of file chunker.c.

References GraphemNodeStruct::arc, cdgPrintf(), Chunk, chunkerStringOfChunkType(), getCategory(), GraphemNode, ChunkStruct::head, NoChunk, ChunkStruct::nodes, NULL, ChunkStruct::subChunks, ChunkStruct::to, and ChunkStruct::type.

Referenced by chunkerPrintChunks(), evalChunker(), and findChunk().

5.3.3.41 void resetChunker (Chunker chunker) [static]

resetChunker: set the chunker in a state of innocence.

parameters: chunker = the object of desire Definition at line 404 of file chunker.c.

References ChunkerStruct::args, cdgFreeString(), Chunker, chunkerChunkDelete(), ChunkerStruct::chunks, ChunkerStruct::lg, ChunkerStruct::nrLevels, ChunkerStruct::nrWords, NULL, ChunkerStruct::parse, ChunkerStruct::pid, ChunkerStruct::pipe1, ChunkerStruct::pipe2, and terminateChild().

Referenced by chunkerDelete(), and initFakeChunker().

5.3.3.42 int terminateChild (pid_t pid)

terminateChild

This function waits for child with the specified pid to terminate.

Return values:

-1 error 0 child died already 1 child died after SIGTERM 2 child died after SIGKILL Definition at line 137 of file chunker.c.

References CDG_ERROR, CDG_WARNING, cdgPrintf(), and NULL.

Referenced by resetChunker().

5.3.4 Variable Documentation

5.3.4.1 char** chunkerArgs = NULL [static]

NULL terminated array of command arguments used for real chunking Definition at line 81 of file chunker.c.

Referenced by chunkerCommandValidate(), and chunkerNew().

5.3.4.2 String chunkerCommand = NULL [static]

string representation of the current command used for real chunking Definition at line 78 of file chunker.c.

Referenced by chunkerInitialize(), and chunkerNew().

5.3.4.3 ChunkerMode chunkerMode = RealChunker [static]

set the default chunker mode,

See also:

[DefaultChunker](#)

Definition at line 75 of file chunker.c.

Referenced by chunkerInitialize(), and chunkerNew().

5.3.4.4 Boolean chunkerUseChunker = FALSE [static]

indicates whether the chunker is used or not Definition at line 72 of file chunker.c.

Referenced by chunkerInitialize(), and chunkerNew().

5.4 Command - The CDG Scripting Language

5.4.1 Detailed Description

Author:

Ingo Schroeder (see also AUTHORS and THANKS for more)

Id

[command.c](#),v 1.309 2004/10/04 14:36:12 micha Exp

This module implements the CDG scripting interface, that is, all the commands that are available on the `cdgp` prompt or executable via `commandEval()`.

Note: do not confuse the scripting interface language with the constraint grammar language or the language bindings. They are all separate things.

Each valid command corresponds to a C function `cmdCommand` that implements this command all commands are listed in [interface_commands](#). Each command is defined by a [Command - The CDG Scripting Language](#) structure, where the name, the interface completion commands, a little help message for that command etc are defined.

The following commands exist:

- activate, `cmdActivate()`: activates a section
- annotation, `cmdAnnotation()`: prints out one or all annotation
- anno2parse, `cmdAnno2Parse()`: build a parse from an annotation and evaluate it
- chunk, `cmdChunk()`: call a chunk parser on a lattice
- closedb, `cmdCloseDB()`: closes the database
- compareparses, `cmdCompareParses()`: compares the structures of two parses
- compile, `cmdCompile()`: compiles the current grammar
- constraint, `cmdConstraint()`: prints out one or all constraints
- deactivate, `cmdDeactivate()`: deactivates a section
- distance, `cmdDistance()`: prints the distance matrix of a net
- edges, `cmdEdges()`: prints out one or all edges of a constraint net
- frobbing, `cmdFrobbing()`: transforms an arbitrary assignment into a solution
- gls, `cmdGls()`: solve a constraintnet using guided local search
- help, `cmdHelp()`: gives a short help
- hierarchy, `cmdHierarchy()`: prints out one or all hierarchies
- hook, `cmdHook()`: toggle hook-system on or off"
- incrementalcompletion, `cmdIncrementalCompletion()`: incremental completion of parses
- inputwordgraph, `cmdInputwordgraph()`: creates new wordgraph
- isearch, `cmdISearch()`: incremental parsing of a wordgraph
- level, `cmdLevel()`: prints out one or all level declarations

- `levelsort`, `cmdLevelsort()`: set/check the priority of the defined levels
- `lexicon`, `cmdLexicon()`: prints out one or all lexical entries
- `license`, `cmdLicense()`: prints out the license
- `load`, `cmdLoad()`: loads a file
- `ls`, `cmdLs()`: lists one or more files
- `net`, `cmdNet()`: prints out one or all constraint net
- `netdelete`, `cmdNetdelete()`: deletes a constraint net
- `netsearch`, `cmdNetsearch()`: searches a net for globally best solutions
- `newnet`, `cmdNewnet()`: builds a new constraint net from a wordgraph
- `nonspeccompatible`, `cmdNonSpecCompatible()`: check compatibility of constraints for incremental parsing
- `parsedelete`, `cmdParsedelete()`: deletes a parse
- `printparse`, `cmdPrintParse()`: prints the structure of a parse
- `printparses`, `cmdPrintParses()`: prints out all parses of a constraint net
- `quit`, `cmdQuit()`: quits
- `renewnet`, `cmdRenewnet()`: restores a net to pristine state
- `reset`, `cmdReset()`: discards all loaded and computed information
- `section`, `cmdSection()`: prints out one or all section
- `set`, `cmdSet()`: set various system variables
- `showlevel`, `cmdShowlevel()`: toggles whether a level is shown or not
- `status`, `cmdStatus()`: describes the status of the system
- `tagger`, `cmdTagger()`: start/stop POS tagger
- `testing`, `cmdTesting()`: undocumented command for testing
- `useconstraint`, `cmdUseconstraint()`: toggles whether a constraint is used or not
- `uselevel`, `cmdUselevel()`: toggles whether a level is used or not
- `uselexicon`, `cmdUseLexicon()`: specify which lexicon and which index database to use
- `version`, `cmdVersion()`: prints version
- `verify`, `cmdVerify()`: compares parse to annotation
- `weight`, `cmdWeight()`: changes a constraint's weight
- `wordgraph`, `cmdWordgraph()`: prints out one or all wordgraph
- `writenet`, `cmdWriteNet()`: writes a TeX representation of the net
- `writeparses`, `cmdWriteParses()`: writes XFig and LaTeX files with the parses
- `writewordgraph`, `cmdWriteWordgraph()`: writes a wordgraph in an XFig file

Data Structures

- struct [Command](#)

Defines

- #define [STARTUPMSG](#)

Typedefs

- typedef Boolean [CommandFunction](#) (int no, char **args)

Functions

- char * [annotation_completion_function](#) (const char *, int)
- Boolean [cmdActivate](#) (int no, char **args)
- Boolean [cmdAnno2Parse](#) (int no, char **args)
- Boolean [cmdAnnos2prolog](#) (int no, char **args)
- Boolean [cmdAnnotation](#) (int no, char **args)
- Boolean [cmdChart](#) (int no, char **args)
- Boolean [cmdChunk](#) (int no, char **args)
- Boolean [cmdCloseDB](#) ()
- Boolean [cmdCompareParses](#) (int no, char **args)
- Boolean [cmdCompile](#) (int no, char **args)
- Boolean [cmdConstraint](#) (int no, char **args)
- Boolean [cmdDeactivate](#) (int no, char **args)
- Boolean [cmdDistance](#) (int no, char **args)
- Boolean [cmdEdges](#) (int no, char **args)
- Boolean [cmdFrobbing](#) (int no, char **args)
- Boolean [cmdGls](#) (int no, char **args)
- Boolean [cmdHelp](#) (int no, char **args)
- Boolean [cmdHierarchy](#) (int no, char **args)
- Boolean [cmdHook](#) (int no, char **args)
- Boolean [cmdIncrementalCompletion](#) (int no, char **args)
- Boolean [cmdInputwordgraph](#) (int no, char **args)
- Boolean [cmdISearch](#) (int no, char **args)
- Boolean [cmdLevel](#) (int no, char **args)
- Boolean [cmdLevelsort](#) (int no, char **args)
- Boolean [cmdLexicon](#) (int no, char **args)
- Boolean [cmdLicense](#) (int no, char **args)
- Boolean [cmdLoad](#) (int no, char **args)
- Boolean [cmdLs](#) (int no, char **args)
- Boolean [cmdNet](#) (int no, char **args)
- Boolean [cmdNetdelete](#) (int no, char **args)
- Boolean [cmdNetsearch](#) (int no, char **args)
- Boolean [cmdNewnet](#) (int no, char **args)
- Boolean [cmdNonSpecCompatible](#) (int no, char **args)
- Boolean [cmdParsedelete](#) (int no, char **args)
- Boolean [cmdParses2prolog](#) (int no, char **args)

- Boolean `cmdPrintParse` (int no, char **args)
- Boolean `cmdPrintParses` (int no, char **args)
- Boolean `cmdQuit` (int no, char **args)
- Boolean `cmdRenewnet` (int no, char **args)
- Boolean `cmdReset` (int no, char **args)
- Boolean `cmdSection` (int no, char **args)
- Boolean `cmdSet` (int no, char **args)
- Boolean `cmdShift` (int no, char **args)
- Boolean `cmdShowlevel` (int no, char **args)
- Boolean `cmdStatus` (int no, char **args)
- Boolean `cmdTagger` (int no, char **args)
- Boolean `cmdTesting` (int no, char **args)
- Boolean `cmdUseconstraint` (int no, char **args)
- Boolean `cmdUselevel` (int no, char **args)
- Boolean `cmdUseLexicon` (int no, char **args)
- Boolean `cmdVerify` (int no, char **args)
- Boolean `cmdVersion` (int no, char **args)
- Boolean `cmdWeight` (int no, char **args)
- Boolean `cmdWordgraph` (int no, char **args)
- Boolean `cmdWriteAnno` (int no, char **args)
- Boolean `cmdWriteNet` (int no, char **args)
- Boolean `cmdWriteParses` (int no, char **args)
- Boolean `cmdWriteWordgraph` (int no, char **args)
- char * `command_completion_function` (const char *, int)
- Boolean `commandEval` (String line)
- Boolean `commandLoop` (String prompt)
- char * `constraint_completion_function` (const char *, int)
- char * `frob_method_completion_function` (const char *, int)
- String `getNextArgument` (String s, int *index, int stop)
- char * `hierarchy_completion_function` (const char *, int)
- char * `hook_completion_function` (const char *, int)
- char ** `interface_completion` (String text, int start, int end)
- char * `level_completion_function` (const char *, int)
- char * `levelsort_completion_function` (const char *, int)
- char * `lexicon_completion_function` (const char *, int)
- char * `make_rl_string` (char *)
- char * `net_completion_function` (const char *, int)
- char * `parse_completion_function` (const char *, int)
- char * `search_method_completion_function` (const char *, int)
- char * `section_completion_function` (const char *, int)
- char * `set_completion_function` (const char *, int)
- char * `word_completion_function` (const char *, int)
- char * `wordgraph_completion_function` (const char *, int)

Variables

- Boolean `commandLoopFlag` = TRUE
- List `interface_cell` = NULL
- Command `interface_commands` []
- HashIterator `interface_hashiterator` = NULL
- int `interface_index` = 0
- int `interface_length` = 0
- List `interface_list` = NULL

5.4.2 Define Documentation

5.4.2.1 #define STARTUPMSG

Value:

```

"\nCDG parser version " VERSION ", build " BUILD "\n" \
"\n" \
"Copyright (C) 1997-2004 The CDG Team\n"\
"The CDG parser comes with ABSOLUTELY NO WARRANTY.\n"\
"This is free software, and you are welcome to redistribute it\n"\
"under certain conditions; type 'license' for details.\n"\
"\n"\
"For more information see\n"\
"\n"\
" http://nats-www.informatik.uni-hamburg.de/~dawai\n"\
"\n"\
"The CDG Team can be contacted at\n"\
"\n"\
" cdg@nats.informatik.uni-hamburg.de\n"\
"\n"\
"Type 'help' for a short help.\n"\
"\n"\
"\n"\
"

```

message to be displayed at startup Definition at line 30 of file command.h.

Referenced by cmdVersion().

5.4.3 Typedef Documentation

5.4.3.1 typedef Boolean [CommandFunction](#)(int no, char **args)

type of a function that implements a scripting command.

The first parameter `no` is the number of arguments entered by the user after the command name. The array `args` holds these arguments as a list of plain strings. For example, the command `"\c newnet \c s1"` contains one command and argument. To handle it, the function [cmdNewnet\(\)](#) will be called with two arguments: the number 1 (as we have one argument) and an array containing the argument strings, which is one here: `"s1"`.

The job of a [CommandFunction\(\)](#) is to check the arguments for consistency and then either execute the action specified by the command or give reasons why they are incorrect.

Each command returns a Boolean which is [TRUE](#) on a successful command execution and otherwise [FALSE](#), that is when an error occurred. Definition at line 161 of file command.c.

5.4.4 Function Documentation

5.4.4.1 char * [annotation_completion_function](#) (const char * text, int state)

Compute completions for annotations. Definition at line 643 of file command.c.

References [interface_cell](#), [interface_length](#), [interface_list](#), [make_rl_string\(\)](#), and [NULL](#).

5.4.4.2 Boolean cmdActivate (int no, char ** args)

Activate constraint classes.

All constraint classes that are named as arguments are activated, that is, their field `->active` is set to `TRUE`. The effect is that these constraints are used (again) for evaluation. Since this changes the basis of evaluation, this is treated like a change in the grammar, i.e. all previously computed structures become invalid and are deleted. Definition at line 1894 of file `command.c`.

References `cdgDeleteComputed()`, `FALSE`, `NULL`, and `TRUE`.

5.4.4.3 Boolean cmdAnno2Parse (int no, char ** args)

Build a parse from an annotation and evaluate it against the current grammar. Definition at line 1633 of file `command.c`.

References `CDG_ERROR`, `CDG_INFO`, `CDG_PROGRESS`, `CDG_WARNING`, `cdgCtrlCAllowed`, `cdgCtrlCTrapped`, `cdgPrintf()`, `dbAvailable()`, `dbLoadAll()`, `FALSE`, `lgDelete()`, `lgNew()`, `NULL`, and `TRUE`.

5.4.4.4 Boolean cmdAnnos2prolog (int no, char ** args)

Save all annotations to a file in Prolog format. Definition at line 4008 of file `command.c`.

References `CDG_ERROR`, `CDG_INFO`, `cdgPrintf()`, `FALSE`, `NULL`, and `TRUE`.

5.4.4.5 Boolean cmdAnnotation (int no, char ** args)

Print the specified annotation entries. Definition at line 1593 of file `command.c`.

References `CDG_DEFAULT`, `CDG_ERROR`, `cdgCtrlCAllowed`, `cdgCtrlCTrapped`, `cdgPrintf()`, `FALSE`, `NULL`, and `TRUE`.

5.4.4.6 Boolean cmdChart (int no, char ** args)

Perform chart-based search. Definition at line 4049 of file `command.c`.

5.4.4.7 Boolean cmdChunk (int no, char ** args)

Chunk the specified wordgraph. Definition at line 3825 of file `command.c`.

References `CDG_ERROR`, `CDG_INFO`, `CDG_PROFILE`, `cdgCtrlCAllowed`, `cdgCtrlCTrapped`, `cdgPrintf()`, `Chunker`, `chunkerChunk()`, `chunkerDelete()`, `chunkerNew()`, `chunkerPrintChunks()`, `DefaultChunker`, `FALSE`, `lgDelete()`, `lgNew()`, `NULL`, `Timer`, `timerElapsed()`, `timerFree()`, `timerNew()`, and `TRUE`.

5.4.4.8 Boolean cmdCloseDB ()

Close the lexicon database. Definition at line 3960 of file `command.c`.

References `dbClose()`.

5.4.4.9 Boolean cmdCompareParses (int no, char ** args)

Compare the specified parses. Definition at line 1926 of file command.c.

References CDG_ERROR, cdgCtrlCAllowed, cdgCtrlCTrapped, cdgParses, cdgPrintf(), FALSE, NULL, and TRUE.

5.4.4.10 Boolean cmdCompile (int no, char ** args)

Compile current grammar.

Currently broken. Definition at line 1198 of file command.c.

References CDG_ERROR, CDG_INFO, cdgPrintf(), comCompile(), comLoad(), evalEvaluationMethod, FALSE, and TRUE.

5.4.4.11 Boolean cmdConstraint (int no, char ** args)

Print the specified constraints. Definition at line 1229 of file command.c.

References CDG_DEFAULT, CDG_ERROR, cdgPrintf(), FALSE, NULL, and TRUE.

5.4.4.12 Boolean cmdDeactivate (int no, char ** args)

Deactivate constraint classes.

All constraint classes that are named as arguments are deactivated, that is, their field `->active` is set to FALSE. The effect is that these constraints are not used for evaluation, as if they had never been defined. Since this changes the basis of evaluation, this is treated like a change in the grammar, i.e. all previously computed structures become invalid and are deleted. Definition at line 1855 of file command.c.

References cdgDeleteComputed(), FALSE, NULL, and TRUE.

5.4.4.13 Boolean cmdDistance (int no, char ** args)

Print the distance matrix of the specified net. Definition at line 3392 of file command.c.

References CDG_DEFAULT, CDG_ERROR, cdgPrintf(), cnFindNet(), ConstraintNet, FALSE, GraphemNode, LexemGraphStruct::graphemnodes, LexemNodeStruct::lexem, ConstraintNetStruct::lexemgraph, LexemNode, lgDistanceOfNodes(), NULL, and TRUE.

5.4.4.14 Boolean cmdEdges (int no, char ** args)

Print edges of the specified constraint net. Definition at line 2440 of file command.c.

References CDG_DEFAULT, CDG_ERROR, cdgNets, cdgPrintf(), cnPrintEdge(), ConstraintEdge, ConstraintNet, ConstraintNode, ConstraintNetStruct::edges, FALSE, ConstraintNetStruct::id, ConstraintNetStruct::nodes, NULL, ConstraintEdgeStruct::start, ConstraintEdgeStruct::stop, and TRUE.

5.4.4.15 Boolean cmdFrobbing (int no, char ** args)

Try to find a complete solution by transformation. Definition at line 2241 of file command.c.

References `cdgCtrlCAllowed`, `cdgCtrlCTrapped`, `cdgTimeLimit`, `FALSE`, `timerSetAlarm()`, `timerStopAlarm()`, and `TRUE`.

5.4.4.16 Boolean `cmdGls` (`int no`, `char ** args`)

Subject the specified net to a guided local search for global solutions. Definition at line 3770 of file `command.c`.

5.4.4.17 Boolean `cmdHelp` (`int no`, `char ** args`)

Print the help string for the specified CDG command. Definition at line 3905 of file `command.c`.

References `CDG_DEFAULT`, `CDG_ERROR`, `cdgPrintf()`, `FALSE`, `interface_commands`, `name`, `NULL`, and `TRUE`.

5.4.4.18 Boolean `cmdHierarchy` (`int no`, `char ** args`)

Print the specified hierarchies. Definition at line 1769 of file `command.c`.

References `CDG_DEFAULT`, `CDG_ERROR`, `cdgCtrlCAllowed`, `cdgCtrlCTrapped`, `cdgPrintf()`, `FALSE`, `NULL`, and `TRUE`.

5.4.4.19 Boolean `cmdHook` (`int no`, `char ** args`)

Set or query hooks. Definition at line 3680 of file `command.c`.

References `HookStruct::active`, `CDG_DEFAULT`, `CDG_ERROR`, `CDG_HOOK`, `CDG_INFO`, `cdgPrintf()`, `HookStruct::count`, `FALSE`, `hkFindNoOfHook()`, `hkHooks`, `hkVerbosity`, `Hook`, `HookStruct::name`, `NULL`, and `TRUE`.

5.4.4.20 Boolean `cmdIncrementalCompletion` (`int no`, `char ** args`)

Parsing problems are created incrementally over prefixes of the word graph and solved individually, with the solution of one problem guiding the search in the next one. Definition at line 3503 of file `command.c`.

References `CDG_ERROR`, `CDG_INFO`, `CDG_PROFILE`, `cdgCtrlCAllowed`, `cdgCtrlCTrapped`, `cdgPrintf()`, `cnEdgesFlag`, `CnEdgesType`, `FALSE`, `NULL`, `scUseCache`, `Timer`, `timerElapsed()`, `timerFree()`, `timerNew()`, and `TRUE`.

5.4.4.21 Boolean `cmdInputwordgraph` (`int no`, `char ** args`)

Build a new wordgraph. Definition at line 3346 of file `command.c`.

References `CDG_ERROR`, `CDG_INFO`, `cdgPrintf()`, `FALSE`, `NULL`, and `TRUE`.

5.4.4.22 Boolean `cmdISearch` (`int no`, `char ** args`)

Perform a complete search on increasing prefixes of the word graph. Definition at line 3563 of file `command.c`.

References CDG_ERROR, CDG_INFO, CDG_PROFILE, cdgCtrlCAllowed, cdgCtrlCTrapped, cdgNets, cdgPrintf(), cnMostRecentlyCreatedNet, ConstraintNet, FALSE, NULL, Timer, timerElapsed(), timerFree(), timerNew(), and TRUE.

5.4.4.23 Boolean cmdLevel (int no, char ** args)

Print the specified level declarations. Definition at line 1253 of file command.c.

References CDG_DEFAULT, CDG_ERROR, cdgPrintf(), FALSE, NULL, and TRUE.

5.4.4.24 Boolean cmdLevelsort (int no, char ** args)

Sort all levels according to the specified order and display the new order. Definition at line 3660 of file command.c.

References CDG_ERROR, CDG_INFO, cdgPrintf(), FALSE, and TRUE.

5.4.4.25 Boolean cmdLexicon (int no, char ** args)

Print the specified lexicon entries. Definition at line 1454 of file command.c.

References CDG_DEFAULT, CDG_INFO, cdgCtrlCAllowed, cdgCtrlCTrapped, cdgPrintf(), FALSE, NULL, and TRUE.

5.4.4.26 Boolean cmdLicense (int no, char ** args)

Print the General Public License. Definition at line 2948 of file command.c.

References CDG_DEFAULT, cdgPrintf(), and TRUE.

5.4.4.27 Boolean cmdLoad (int no, char ** args)

Load one or more files. Definition at line 2871 of file command.c.

References CDG_ERROR, CDG_INFO, cdgDeleteComputed(), cdgExecPragmas(), cdgPrintf(), evalEvaluationMethod, FALSE, NULL, and TRUE.

5.4.4.28 Boolean cmdLs (int no, char ** args)

List one or more files.

This simply spawns the system command `ls`. Definition at line 1163 of file command.c.

References CDG_ERROR, CDG_WARNING, cdgPrintf(), FALSE, NULL, and TRUE.

5.4.4.29 Boolean cmdNet (int no, char ** args)

Print the specified constraint net. Definition at line 2021 of file command.c.

References CDG_DEFAULT, CDG_ERROR, cdgCtrlCAllowed, cdgCtrlCTrapped, cdgNets, cdgPrintf(), cnMostRecentlyCreatedNet, cnPrint(), ConstraintNet, FALSE, NULL, and TRUE.

5.4.4.30 Boolean `cmdNetdelete` (`int no`, `char ** args`)

Delete constraint nets.

If no argument is given, delete all known constraint nets. Otherwise, delete all named nets. Definition at line 2389 of file `command.c`.

References `CDG_ERROR`, `cdgNets`, `cdgPrintf()`, `cdgProblems`, `cnDelete()`, `cnMostRecentlyCreatedNet`, `ConstraintNet`, `FALSE`, and `NULL`.

5.4.4.31 Boolean `cmdNetsearch` (`int no`, `char ** args`)

Perform a complete search for globally best solution. Definition at line 2115 of file `command.c`.

References `CDG_ERROR`, `CDG_INFO`, `CDG_SEARCHRESULT`, `cdgCtrlCAllowed`, `cdgCtrlCTrapped`, `cdgNets`, `cdgPrintf()`, `cnMostRecentlyCreatedNet`, `cnPrintInfo()`, `cnPrintParses()`, `ConstraintNet`, `FALSE`, `hkVerbosity`, `ConstraintNetStruct::id`, `NULL`, `ConstraintNetStruct::parses`, and `TRUE`.

5.4.4.32 Boolean `cmdNewnet` (`int no`, `char ** args`)

Build a new constraint net. Definition at line 3246 of file `command.c`.

References `CDG_ERROR`, `CDG_INFO`, `CDG_PROFILE`, `CDG_WARNING`, `cdgCtrlCAllowed`, `cdgCtrlCTrapped`, `cdgNets`, `cdgPrintf()`, `cnBuild()`, `cnPrintInfo()`, `ConstraintNet`, `FALSE`, `ConstraintNetStruct::id`, `NULL`, `Timer`, `timerElapsed()`, `timerFree()`, `timerNew()`, and `TRUE`.

5.4.4.33 Boolean `cmdNonSpecCompatible` (`int no`, `char ** args`)

Report whether the specified constraints are suitable for processing underspecified subordinations. Definition at line 3449 of file `command.c`.

References `CDG_DEFAULT`, `CDG_ERROR`, `CDG_INFO`, `cdgPrintf()`, `FALSE`, `NULL`, and `TRUE`.

5.4.4.34 Boolean `cmdParsedelete` (`int no`, `char ** args`)

Delete the specified parse. Definition at line 2305 of file `command.c`.

References `CDG_ERROR`, `CDG_INFO`, `cdgParses`, `cdgPrintf()`, `FALSE`, `NULL`, and `TRUE`.

5.4.4.35 Boolean `cmdParses2prolog` (`int no`, `char ** args`)

Save all parses to a file in Prolog format. Definition at line 3969 of file `command.c`.

References `CDG_ERROR`, `CDG_INFO`, `cdgParses`, `cdgPrintf()`, `FALSE`, `NULL`, and `TRUE`.

5.4.4.36 Boolean `cmdPrintParse` (`int no`, `char ** args`)

Prints the specified Parse on stdout. Definition at line 1975 of file `command.c`.

References `CDG_ERROR`, `cdgCtrlCAllowed`, `cdgCtrlCTrapped`, `cdgParses`, `cdgPrintf()`, `FALSE`, `NULL`, and `TRUE`.

5.4.4.37 Boolean cmdPrintParses (int no, char ** args)

Print parses of the specified net. Definition at line 2266 of file command.c.

References CDG_ERROR, cdgCtrlCAllowed, cdgCtrlCTrapped, cdgPrintf(), cnFindNet(), cnMostRecentlyCreatedNet, cnPrintInfo(), cnPrintParses(), ConstraintNet, FALSE, NULL, ConstraintNetStruct::parses, and TRUE.

5.4.4.38 Boolean cmdQuit (int no, char ** args)

Leave the command loop.

See also:

[commandLoop](#)

Definition at line 1151 of file command.c.

References commandLoopFlag, FALSE, and TRUE.

Referenced by commandLoop().

5.4.4.39 Boolean cmdRenewnet (int no, char ** args)

Restore the specified net to original state. Definition at line 3306 of file command.c.

References CDG_ERROR, CDG_INFO, cdgPrintf(), cnFindNet(), cnMostRecentlyCreatedNet, cnPrintInfo(), cnRenew(), ConstraintNet, FALSE, NULL, and TRUE.

5.4.4.40 Boolean cmdReset (int no, char ** args)

Discard all loaded and computed information. Definition at line 2847 of file command.c.

References CDG_ERROR, cdgDeleteComputed(), cdgPrintf(), FALSE, and TRUE.

5.4.4.41 Boolean cmdSection (int no, char ** args)

Print the specified constraint sections. Definition at line 1799 of file command.c.

References CDG_DEFAULT, CDG_ERROR, cdgPrintf(), FALSE, NULL, and TRUE.

5.4.4.42 Boolean cmdSet (int no, char ** args)

Set the specified CDG variable. Definition at line 1820 of file command.c.

References CDG_ERROR, CDG_INFO, cdgPrintf(), FALSE, and NULL.

5.4.4.43 Boolean cmdShift (int no, char ** args)

Simulate shift-reduce parsing. Definition at line 1318 of file command.c.

References cdgCtrlCAllowed, cdgCtrlCTrapped, cdgTimeLimit, FALSE, timerSetAlarm(), timerStopAlarm(), and TRUE.

5.4.4.44 Boolean cmdShowlevel (int no, char ** args)

Toggle whether the specified level is shown or not. Definition at line 1343 of file command.c.

References CDG_ERROR, CDG_INFO, cdgPrintf(), FALSE, NULL, and TRUE.

5.4.4.45 Boolean cmdStatus (int no, char ** args)

Print status information. Definition at line 2659 of file command.c.

References CDG_DEBUG, CDG_DEFAULT, CDG_ERROR, CDG_EVAL, CDG_HINT, CDG_HOOK, CDG_INFO, CDG_PROFILE, CDG_PROGRESS, CDG_PROLOG, CDG_SEARCHRESULT, CDG_WARNING, CDG_XML, cdgNets, cdgParses, cdgPrintf(), cnEdgesFlag, cnShowDeletedFlag, cnSortNodesMethod, cnUnaryPruningFraction, evalEvaluationMethod, evalPeekValueMethod, evalSloppySubsumesWarnings, hkVerbosity, lgCompactLVs, NULL, scUseCache, and TRUE.

5.4.4.46 Boolean cmdTagger (int no, char ** args)

Start or stop the tagger. Definition at line 2168 of file command.c.

References CDG_ERROR, cdgPrintf(), FALSE, and TRUE.

5.4.4.47 Boolean cmdTesting (int no, char ** args)

Undocumented command for testing mode=1: determine base line for accuracy mode=2: determine base line for accuracy (TODO: is that correct?!) mode=3: compare two lexica bit for bit mode=4: approve compiled grammar Definition at line 2193 of file command.c.

References CDG_ERROR, cdgPrintf(), comApprove(), FALSE, and TRUE.

5.4.4.48 Boolean cmdUseconstraint (int no, char ** args)

Toggle whether the specified constraint is used or not. Definition at line 1380 of file command.c.

References CDG_ERROR, CDG_INFO, cdgDeleteComputed(), cdgPrintf(), FALSE, NULL, and TRUE.

5.4.4.49 Boolean cmdUselevel (int no, char ** args)

Toggle whether the specified level is used or not. Definition at line 1411 of file command.c.

References CDG_ERROR, CDG_INFO, cdgDeleteComputed(), cdgPrintf(), FALSE, NULL, and TRUE.

5.4.4.50 Boolean cmdUseLexicon (int no, char ** args)

Specifies a file to be used for autoloading lexicon items.

The parameter must be string that is used as the file name. The file of cdg input must be called S.cdg, and the index must be called S.db.

If these files are not in the current working directory they are also searched in other directories that input was loaded from. Definition at line 3942 of file command.c.

References CDG_ERROR, cdgPrintf(), dbOpen(), FALSE, and TRUE.

5.4.4.51 Boolean cmdVerify (int no, char ** args)

Compare parse to annotation. Definition at line 3778 of file command.c.

References CDG_ERROR, CDG_INFO, cdgPrintf(), FALSE, NULL, and TRUE.

5.4.4.52 Boolean cmdVersion (int no, char ** args)

Print the CDG version. Definition at line 3236 of file command.c.

References CDG_DEFAULT, cdgPrintf(), STARTUPMSG, and TRUE.

5.4.4.53 Boolean cmdWeight (int no, char ** args)

Adjust the weight of a constraint. Definition at line 2060 of file command.c.

References CDG_ERROR, CDG_INFO, cdgPrintf(), FALSE, and TRUE.

5.4.4.54 Boolean cmdWordgraph (int no, char ** args)

Print wordgraphs. Definition at line 1508 of file command.c.

References CDG_DEFAULT, CDG_ERROR, cdgCtrlCAllowed, cdgCtrlCTrapped, cdgPrintf(), FALSE, NULL, and TRUE.

5.4.4.55 Boolean cmdWriteAnno (int no, char ** args)

Write best Parse of a net to disk as an annotation. Definition at line 2540 of file command.c.

References bCompare(), CDG_ERROR, cdgNets, cdgPrintf(), cnMostRecentlyCreatedNet, ConstraintNet, FALSE, ConstraintNetStruct::id, NULL, ConstraintNetStruct::parses, and TRUE.

5.4.4.56 Boolean cmdWriteNet (int no, char ** args)

Write the specified constraint net to an XFig file. Definition at line 2499 of file command.c.

References CDG_ERROR, cdgCtrlCAllowed, cdgCtrlCTrapped, cdgNets, cdgPrintf(), cnMostRecentlyCreatedNet, ConstraintNet, FALSE, NULL, and TRUE.

5.4.4.57 Boolean cmdWriteParses (int no, char ** args)

Write the specified parses to XFig and LaTeX files. Definition at line 2586 of file command.c.

References CDG_ERROR, cdgCtrlCAllowed, cdgCtrlCTrapped, cdgPrintf(), cnFindNet(), cnMostRecentlyCreatedNet, ConstraintNet, FALSE, NULL, ConstraintNetStruct::parses, and TRUE.

5.4.4.58 Boolean cmdWriteWordgraph (int no, char ** args)

Write the specified wordgraphs to XFig file. Definition at line 1547 of file command.c.

References CDG_ERROR, cdgCtrlCAllowed, cdgCtrlCTrapped, cdgPrintf(), FALSE, NULL, and TRUE.

5.4.4.59 `char * command_completion_function (const char * text, int state)`

Compute the completions for commands.

This is one of many functions provided to pass information to the readline library. These functions all have basically the same structure which is required by readline: they must return a newly allocated string every time they are called. They vary only in which strings are returned as possible completions.

The parameter "text" is the partial string typed by the user so far. The completion function must now offer the readline library suitable strings to complete it. This particular function knows what commands the CDG shell understands, so it iterates over the array `interface_commands[]` internally. The other completion functions will iterate over other structures, obviously. Definition at line 462 of file `command.c`.

References `interface_commands`, `interface_index`, `interface_length`, `make_rl_string()`, `name`, and `NULL`.

Referenced by `interface_completion()`.

5.4.4.60 `Boolean commandEval (String line)`

CDG command dispatch routine.

This function interprets one command of the scripting language.

Parameters:

line the command line to be evaluated

Returns:

`TRUE` on a successful command execution, otherwise `FALSE`.

Definition at line 1070 of file `command.c`.

References `CDG_ERROR`, `CDG_XML`, `cdgFlush()`, `cdgFreeString()`, `cdgPrintf()`, `FALSE`, `function`, `getNextArgument()`, `hkVerbosity`, `interface_commands`, `name`, `NULL`, and `TRUE`.

Referenced by `cdgExecPragmas()`, and `commandLoop()`.

5.4.4.61 `Boolean commandLoop (String prompt)`

CDG main event loop.

This function reads and executes commands and terminates when the `commandLoopFlag` has been set to `FALSE`.

Parameters:

prompt to be displayed at the command line

Returns:

the boolean result of the last command, that is `TRUE` on success and else `FALSE`.

Definition at line 1003 of file `command.c`.

References `CDG_DEFAULT`, `CDG_WARNING`, `cdgPrintf()`, `cmdQuit()`, `commandEval()`, `commandLoopFlag`, `interface_completion()`, `NULL`, and `TRUE`.

5.4.4.62 `char * constraint_completion_function (const char * text, int state)`

Compute completions for constraints. Definition at line 499 of file `command.c`.

References `interface_cell`, `interface_length`, `interface_list`, `make_rl_string()`, and `NULL`.

5.4.4.63 char * frob_method_completion_function (const char * text, int state)

Compute completions for frobbing methods. Definition at line 787 of file command.c.

References interface_index, interface_length, make_rl_string(), and NULL.

5.4.4.64 String getNextArgument (String s, int * index, int stop)

Return a copy of the next command word in S.

Command words are searched in the part of S starting at INDEX and not exceeding STOP. **Command** words are separated by instances of the characters defined in rl_basic_word_break_characters by the readline library.

(void)strncpy(buffer, &s[start], *index - start - 1); buffer[*index - start - 1] = ”; Definition at line 859 of file command.c.

References CDG_WARNING, cdgPrintf(), cdgXCDG, and NULL.

Referenced by commandEval(), and interface_completion().

5.4.4.65 char * hierarchy_completion_function (const char * text, int state)

Compute completions for hierarchies. Definition at line 668 of file command.c.

References interface_cell, interface_length, interface_list, make_rl_string(), and NULL.

5.4.4.66 char * hook_completion_function (const char * text, int state)

Compute completions for known hooks. Definition at line 808 of file command.c.

References hkHooks, Hook, interface_index, interface_length, make_rl_string(), and NULL.

5.4.4.67 char ** interface_completion (String text, int start, int end)

Provide context sensitive command completion.

This function is passed to the readline library as the rl_attempted_completion_function. See the readline documentation for details of its behaviour. Definition at line 932 of file command.c.

References args, cdgFreeString(), command_completion_function(), getNextArgument(), interface_commands, name, NULL, and same.

Referenced by commandLoop().

5.4.4.68 char * level_completion_function (const char * text, int state)

Compute completions for the command level. Definition at line 523 of file command.c.

References interface_length, interface_list, make_rl_string(), and NULL.

5.4.4.69 char * levelsort_completion_function (const char * text, int state)

Compute completions for the command levelsort. Definition at line 544 of file command.c.

References interface_length, interface_list, make_rl_string(), and NULL.

5.4.4.70 char * lexicon_completion_function (const char * text, int state)

Compute completions for lexical entries. Definition at line 569 of file command.c.

References interface_hashiterator, interface_length, make_rl_string(), and NULL.

5.4.4.71 char * make_rl_string (char * s)

Produce a string suitable for internal processing by readline.

This function serves two purposes:

1. it accepts a string and returns a clone of it. This is necessary because the readline library insists on doing its own deallocating.
1. if the string passed contains whitespace, it wraps quotes around it so that the CDG shell will parse it as one string later. That way, completion works even for identifiers with whitespace in them (highly discouraged but possible).

TODO: This is not yet quite correct. It can handle identifiers like "SYN", "um zu", and "it's". But it can fail if a string contains both whitespace and single quotes:

```
"Hand me the `scope, skipper!"
```

My excuse is that I can think of no reason whatsoever why anybody would use that sort of identifier in a constraint grammar. Definition at line 432 of file command.c.

References NULL.

Referenced by annotation_completion_function(), command_completion_function(), constraint_completion_function(), frob_method_completion_function(), hierarchy_completion_function(), hook_completion_function(), level_completion_function(), levelsort_completion_function(), lexicon_completion_function(), net_completion_function(), parse_completion_function(), search_method_completion_function(), section_completion_function(), set_completion_function(), and wordgraph_completion_function().

5.4.4.72 char * net_completion_function (const char * text, int state)

Compute completions for constraint nets. Definition at line 693 of file command.c.

References cdgNets, interface_cell, interface_length, interface_list, make_rl_string(), and NULL.

5.4.4.73 char * parse_completion_function (const char * text, int state)

Compute completions for parses. Definition at line 830 of file command.c.

References cdgParses, interface_cell, interface_length, interface_list, make_rl_string(), and NULL.

5.4.4.74 char * search_method_completion_function (const char * text, int state)

Compute completions for search methods. Definition at line 718 of file command.c.

References interface_index, interface_length, make_rl_string(), and NULL.

5.4.4.75 char * section_completion_function (const char * text, int state)

Compute completions for sections. Definition at line 738 of file command.c.

References interface_length, interface_list, make_rl_string(), and NULL.

5.4.4.76 char * set_completion_function (const char * text, int state)

Compute completions for the set command Definition at line 759 of file command.c.

References interface_index, interface_length, make_rl_string(), and NULL.

5.4.4.77 char * word_completion_function (const char * text, int state)

Compute completions for orthographic words. Definition at line 593 of file command.c.

References interface_hashiterator, interface_length, and NULL.

5.4.4.78 char * wordgraph_completion_function (const char * text, int state)

Compute completions for wordgraphs. Definition at line 620 of file command.c.

References interface_length, interface_list, make_rl_string(), and NULL.

5.4.5 Variable Documentation**5.4.5.1 Boolean `commandLoopFlag = TRUE` [static]**

flag indicating the end of the command main loop. Entering the `commandLoop()` this is set to `TRUE`. `commandLoop()` exits if this global variable is set to `FALSE`. Definition at line 202 of file command.c.

Referenced by `cmdQuit()`, and `commandLoop()`.

5.4.5.2 List `interface_cell = NULL` [static]**Todo**

Write an explanation of the interface and the completion gadget

Definition at line 192 of file command.c.

Referenced by `annotation_completion_function()`, `constraint_completion_function()`, `hierarchy_completion_function()`, `net_completion_function()`, and `parse_completion_function()`.

5.4.5.3 Command `interface_commands[]`

array of all registered wcdg scripting commands. Definition at line 227 of file command.c.

Referenced by `cmdHelp()`, `command_completion_function()`, `commandEval()`, and `interface_completion()`.

5.4.5.4 HashIterator `interface_hashiterator = NULL` [static]

Todo

Write an explanation of the interface and the completion gadget

Definition at line 195 of file `command.c`.

Referenced by `lexicon_completion_function()`, and `word_completion_function()`.

5.4.5.5 `int interface_index = 0` [static]

Todo

Write an explanation of the interface and the completion gadget

Definition at line 183 of file `command.c`.

Referenced by `command_completion_function()`, `frob_method_completion_function()`, `hook_completion_function()`, `search_method_completion_function()`, and `set_completion_function()`.

5.4.5.6 `int interface_length = 0` [static]

Todo

Write an explanation of the interface and the completion gadget

Definition at line 186 of file `command.c`.

Referenced by `annotation_completion_function()`, `command_completion_function()`, `constraint_completion_function()`, `frob_method_completion_function()`, `hierarchy_completion_function()`, `hook_completion_function()`, `level_completion_function()`, `levelsort_completion_function()`, `lexicon_completion_function()`, `net_completion_function()`, `parse_completion_function()`, `search_method_completion_function()`, `section_completion_function()`, `set_completion_function()`, `word_completion_function()`, and `wordgraph_completion_function()`.

5.4.5.7 List `interface_list = NULL` [static]

Todo

Write an explanation of the interface and the completion gadget

Definition at line 189 of file `command.c`.

Referenced by `annotation_completion_function()`, `constraint_completion_function()`, `hierarchy_completion_function()`, `level_completion_function()`, `levelsort_completion_function()`, `net_completion_function()`, `parse_completion_function()`, `section_completion_function()`, and `wordgraph_completion_function()`.

5.5 Compiler - compile a constraint grammar

5.5.1 Detailed Description

Author:

Michael Daum (see also AUTHORS and THANKS for more)

Id

[compile.c](#),v 1.87 2004/10/04 14:36:12 micha Exp

This module implements a compiler in order to translate constraints into C code.

Data Structures

- struct [ApproverStruct](#)
- struct [CompilerStruct](#)
- struct [MakeInfoStruct](#)

Defines

- `#define FINIT_GRAMMAR "finitGrammar"`
- `#define INIT_GRAMMAR "initGrammar"`

Typedefs

- typedef [ApproverStruct](#) * [Approver](#)
- typedef [CompilerStruct](#) * [Compiler](#)
- typedef [CompilerStruct](#) [CompilerStruct](#)
- typedef [MakeInfoStruct](#) * [MakeInfo](#)
- typedef [MakeInfoStruct](#) [MakeInfoStruct](#)

Enumerations

- enum [ReturnType](#) {
 [RTBoolean](#) = (1L << 0), [RTNumber](#) = (1L << 1), [RTString](#) = (1L << 2), [RTAVNode](#) = (1L << 3),
 [RTList](#) = (1L << 4), [RTConjunction](#) = (1L << 5), [RTDisjunction](#) = (1L << 6), [RTError](#) = (1L << 7),
 [RTLexemNode](#) = (1L << 8), [RTGraphemNode](#) = (1L << 9), [RTNoError](#) = (1L << 10), [RTPeek](#) = (1L << 11),
 [RTLexemPosition](#) = (1L << 12) }

Functions

- void [comAnalyzeGrammar](#) (void)
- void [comApprove](#) (int no, char **args)
- Boolean [comCompareAllLvPairs](#) ([Approver](#), [ConstraintNet](#) net)
- Boolean [comCompareAllLvs](#) ([Approver](#), [ConstraintNet](#), [ConstraintNet](#))

- Boolean [comCompareLvs](#) (LevelValue, LevelValue)
- Boolean [comCompareNets](#) (Approver, ConstraintNet, ConstraintNet)
- Boolean [comCompareWithContext](#) (Approver, ConstraintNet net, List contextList)
- String [comCompile](#) (int no, char **args)
- String [comConnexionToString](#) (Connexion)
- int [comConstraintDepth](#) (Constraint)
- String [comDirectionToString](#) (Direction)
- String [comEscapeQuotes](#) (String)
- void [comFinalize](#) (void)
- LevelValue [comFindComparableLv](#) (ConstraintNet, LevelValue)
- void [comFinitGrammar](#) (Input thisInput)
- int [comFormulaDepth](#) (Formula)
- String [comFormulaTypeToString](#) (Formula)
- void [comFree](#) (Compiler)
- void [comFreeApprover](#) (Approver)
- int [comFunctionDepth](#) (String, List)
- void [comIndent](#) (void)
- int [comIndexOfConstraint](#) (String)
- int [comIndexOfHierarchy](#) (String)
- int [comIndexOfVarInfo](#) (VarInfo)
- Boolean [comInitGrammar](#) (Input thisInput)
- void [comInitialize](#) (void)
- Input [comLoad](#) (String filename)
- Boolean [comMake](#) (void)
- void [comMakeInfoFree](#) (MakeInfo)
- [MakeInfo](#) [comMakeInfoNew](#) (void)
- int [comMaxLookupStrings](#) (Constraint)
- int [comMaxLookupStringsInFormula](#) (Formula)
- int [comMaxLookupStringsInTerm](#) (Term)
- [Compiler](#) [comNew](#) (void)
- [Approver](#) [comNewApprover](#) (void)
- void [comOutdent](#) (void)
- int [comPredicateDepth](#) (String, List)
- void [comPrint](#) (String,...)
- void [comPrintln](#) (String,...)
- int [comRegisterString](#) (String)
- [ReturnType](#) [comReturnTypeOfFormula](#) (Formula)
- [ReturnType](#) [comReturnTypeOfFunction](#) (String, List)
- [ReturnType](#) [comReturnTypeOfPredicate](#) (String, List)
- [ReturnType](#) [comReturnTypeOfTerm](#) (Term)
- String [comReturnTypeToString](#) ([ReturnType](#))
- int [comTermDepth](#) (Term)
- String [comTermTypeToString](#) (Term)
- Boolean [comTranslate](#) (void)
- void [comTranslateAbs](#) (List, int)
- void [comTranslateArithmetics](#) (Term, int)
- void [comTranslateBetween](#) (String, List, int)
- void [comTranslateBinaryConstraints](#) (void)
- void [comTranslateBottomPeek](#) (Term, int)
- void [comTranslateChunkHead](#) (String, List, int)

- void [comTranslateConnected](#) (String, List, int)
- void [comTranslateConnexion](#) (Formula)
- void [comTranslateConstraint](#) (Constraint)
- void [comTranslateDirection](#) (Formula)
- void [comTranslateDistance](#) (List, int)
- void [comTranslateEquation](#) (Formula, int)
- void [comTranslateExists](#) (List)
- void [comTranslateFormula](#) (Formula, int)
- void [comTranslateFunction](#) (String, List, int)
- void [comTranslateGuard](#) (String, List, int)
- void [comTranslateHas](#) (String, List, int)
- void [comTranslateHeight](#) (List, int)
- void [comTranslatels](#) (String, List, int)
- void [comTranslateLexemNodeAccess](#) (Term, int)
- void [comTranslateLexicalAccess](#) (Term)
- void [comTranslateLookup](#) (List, int)
- void [comTranslateMatch](#) (List, int)
- void [comTranslateMinMax](#) (String, List, int)
- void [comTranslateNumber](#) (Term, int)
- void [comTranslateParens](#) (List, int)
- void [comTranslateParent](#) (List, int)
- void [comTranslatePhrasequotes](#) (List, int)
- void [comTranslatePredicate](#) (String, List, int)
- void [comTranslatePrint](#) (List, int)
- void [comTranslatePts](#) (List, int)
- void [comTranslateQuotes](#) (List, int)
- void [comTranslateStartStop](#) (String, List, int)
- void [comTranslateString](#) (Term, int)
- void [comTranslateSubsumes](#) (String, List, int)
- void [comTranslateTerm](#) (Term, Boolean, Boolean, int)
- void [comTranslateTopPeek](#) (Term, int)
- void [comTranslateUnaryConstraints](#) (void)
- void [comTranslateUnder](#) (String, List, int)
- void [comTranslateUnEquation](#) (Formula, int)
- String [comValueTypeToString](#) (Value)
- void [comWriteDeclarations](#) (void)
- void [comWriteError](#) (String,...)
- void [comWriteFinitFunction](#) (void)
- void [comWriteFunctions](#) (void)
- void [comWriteHeader](#) (void)
- void [comWriteInitFunction](#) (void)
- void [comWriteWarning](#) (String,...)

Variables

- [Compiler](#) `com`
- [MakeInfo](#) `makeInfo`

5.5.2 Define Documentation

5.5.2.1 `#define FINIT_GRAMMAR "finitGrammar"`

abbrev. Definition at line 51 of file compile.c.

Referenced by `comFinitGrammar()`, `comWriteDeclarations()`, and `comWriteFinitFunction()`.

5.5.2.2 `#define INIT_GRAMMAR "initGrammar"`

abbrev. Definition at line 50 of file compile.c.

Referenced by `comInitGrammar()`, `comWriteDeclarations()`, and `comWriteInitFunction()`.

5.5.3 Typedef Documentation

5.5.3.1 `typedef ApproverStruct* Approver`

type of an Approver Definition at line 89 of file compile.c.

Referenced by `comApprove()`, and `comNewApprover()`.

5.5.3.2 `typedef struct CompilerStruct* Compiler`

type of a Compiler Definition at line 76 of file compile.c.

Referenced by `comFree()`, and `comNew()`.

5.5.3.3 `typedef struct CompilerStruct CompilerStruct`

definition of a CDG compiler object.

5.5.3.4 `typedef struct MakeInfoStruct* MakeInfo`

A Pointer to a `MakeInfoStruct` Definition at line 106 of file compile.c.

Referenced by `comMakeInfoFree()`, and `comMakeInfoNew()`.

5.5.3.5 `typedef struct MakeInfoStruct MakeInfoStruct`

default make settings. This structure bundles all the information that is needed to compile and link a binary CDG grammar. This is called `MakeInfo` in according to the normal place where such information is kept.

5.5.4 Enumeration Type Documentation

5.5.4.1 `enum ReturnTypes`

possible return types of formulas, predicates, functions, terms.

Enumeration values:

RTBoolean Boolean return type

RTNumber Number return type

RTString String return type

RTAVNode AVNode return type

RTList List return type

RTConjunction Conjunction return type

RTDisjunction Disjunction return type

RTErrors Syntax error "return type"

RTLexemNode LexemNode return type

RTGraphemNode GraphemNode return type

RTNoError indicate that no error is possible in the returning function

RTPeek indicate a lexicon access in the returning function

RTLexemPosition indicate the position where the returning function is peeking 0: modifier, 1: modify

Definition at line 111 of file compile.c.

Referenced by comReturnTypeOfFormula(), comReturnTypeOfFunction(), comReturnTypeOfPredicate(), comReturnTypeOfTerm(), comTranslateAbs(), comTranslateArithmetics(), comTranslateBetween(), comTranslateChunkHead(), comTranslateConnected(), comTranslateConstraint(), comTranslateDistance(), comTranslateEquation(), comTranslateFormula(), comTranslateGuard(), comTranslateHas(), comTranslateIs(), comTranslateLookup(), comTranslateMatch(), comTranslateMinMax(), comTranslateParens(), comTranslateParent(), comTranslatePhrasequotes(), comTranslatePrint(), comTranslateQuotes(), comTranslateStartStop(), comTranslateSubsumes(), comTranslateTerm(), comTranslateUnder(), and comTranslateUnEquation().

5.5.5 Function Documentation

5.5.5.1 void comAnalyzeGrammar (void)

analyze the currently loaded grammar Definition at line 4582 of file compile.c.

References com, comConstraintDepth(), comMaxLookupStrings(), CompilerStruct::maxLookupStrings, and CompilerStruct::maxValues.

Referenced by comTranslate().

5.5.5.2 void comApprove (int no, char ** args)

approve a compiled grammar Definition at line 5700 of file compile.c.

References Approver, CDG_DEFAULT, CDG_ERROR, CDG_WARNING, cdgCtrlAllowed, cdgCtrlTrapped, cdgFlush(), cdgNets, cdgPrintf(), cnBuild(), cnDelete(), comCompareNets(), comFreeApprover(), comNewApprover(), ConstraintNet, evalEvaluationMethod, EvalMethodType, FALSE, hkVerbosity, ConstraintNetStruct::id, NULL, ApproverStruct::timer, timerElapsed(), timerStart(), ApproverStruct::totalCompiledBinaryTime, ApproverStruct::totalCompiledUnaryTime, ApproverStruct::totalInterpretedBinaryTime, ApproverStruct::totalInterpretedUnaryTime, and TRUE.

Referenced by cmdTesting().

5.5.5.3 Boolean `comCompareAllLvPairs` ([Approver *ap*](#), [ConstraintNet *net*](#))

compare each pair of levelvalues in both networks Definition at line 5308 of file compile.c.

References `ConstraintNetStruct::cache`, `CDG_ERROR`, `cdgCtrlCTrapped`, `cdgPrintf()`, `ConstraintViolationStruct::constraint`, `ConstraintNet`, `ConstraintViolation`, `cvCompare()`, `cvDelete()`, `evalBinary()`, `evalEvaluationMethod`, `FALSE`, `BadnessStruct::hard`, `ConstraintViolationStruct::lv1`, `ConstraintViolationStruct::lv2`, `BadnessStruct::no`, `ApproverStruct::noRounds`, `NULL`, `ConstraintViolationStruct::penalty`, `scDelete()`, `scNew()`, `scUseCache`, `BadnessStruct::soft`, `ApproverStruct::timer`, `timerElapsed()`, `timerStart()`, `ApproverStruct::totalCompiledBinaryTime`, `ApproverStruct::totalInterpretedBinaryTime`, `TRUE`, and `ConstraintNetStruct::values`.

Referenced by `comCompareNets()`.

5.5.5.4 Boolean `comCompareAllLvs` ([Approver *ap*](#), [ConstraintNet *net1*](#), [ConstraintNet *net2*](#))

compare each levelvalue in both networks returns TRUE if all are equal, else FALSE Definition at line 5266 of file compile.c.

References `CDG_ERROR`, `cdgCtrlCTrapped`, `cdgPrintf()`, `comCompareLvs()`, `comFindComparableLv()`, `ConstraintNet`, `FALSE`, `NULL`, `TRUE`, and `ConstraintNetStruct::values`.

Referenced by `comCompareNets()`.

5.5.5.5 Boolean `comCompareLvs` ([LevelValue *lv1*](#), [LevelValue *lv2*](#))

compare two levelvalues returns TRUE if equal, else FALSE Definition at line 5157 of file compile.c.

References `CDG_ERROR`, `cdgPrintf()`, `FALSE`, `NULL`, and `TRUE`.

Referenced by `comCompareAllLvs()`.

5.5.5.6 Boolean `comCompareNets` ([Approver *ap*](#), [ConstraintNet *net1*](#), [ConstraintNet *net2*](#))

compare two constraint networks Definition at line 5643 of file compile.c.

References `CDG_ERROR`, `cdgPrintf()`, `comCompareAllLvPairs()`, `comCompareAllLvs()`, `comCompareWithContext()`, `ConstraintNet`, `ConstraintNode`, `FALSE`, `ConstraintNetStruct::nodes`, `NULL`, `TRUE`, `ConstraintNodeStruct::values`, and `ConstraintNetStruct::values`.

Referenced by `comApprove()`.

5.5.5.7 Boolean `comCompareWithContext` ([Approver *ap*](#), [ConstraintNet *net*](#), [List *contextList*](#))

compare the nets after applying context_sensitive constraints Definition at line 5473 of file compile.c.

References `ConstraintNetStruct::cache`, `CDG_ERROR`, `cdgCtrlCTrapped`, `cdgPrintf()`, `ConstraintViolationStruct::constraint`, `ConstraintNet`, `ConstraintViolation`, `cvCompare()`, `cvDelete()`, `evalBinary()`, `evalEvaluationMethod`, `FALSE`, `BadnessStruct::hard`, `ConstraintViolationStruct::lv1`, `ConstraintViolationStruct::lv2`, `BadnessStruct::no`, `ApproverStruct::noRounds`, `NULL`, `ConstraintViolationStruct::penalty`, `scDelete()`, `scNew()`, `scUseCache`, `BadnessStruct::soft`, `ApproverStruct::timer`, `timerElapsed()`, `timerStart()`, `ApproverStruct::totalCompiledBinaryTime`, `ApproverStruct::totalInterpretedBinaryTime`, `TRUE`, and `ConstraintNetStruct::values`.

Referenced by `comCompareNets()`.

5.5.5.8 String comCompile (int *no*, char ** *args*)

comCompile: entry function to this package.

- translate the currently loaded cdg into C
- make a shared object out of it
- load it

returns the name of the so file, NULL on failure Definition at line 4922 of file compile.c.

References MakeInfoStruct::cc, CompilerStruct::ccFile, CompilerStruct::ccFileName, CDG_DEFAULT, CDG_ERROR, cdgPrintf(), MakeInfoStruct::cFlags, com, comFree(), comMake(), comNew(), comTranslate(), FALSE, MakeInfoStruct::includes, MakeInfoStruct::ld, MakeInfoStruct::ldFlags, MakeInfoStruct::ldLibs, makeInfo, NULL, CompilerStruct::objFileName, CompilerStruct::soFileName, CompilerStruct::translateOnly, and TRUE.

Referenced by cmdCompile().

5.5.5.9 String comConnexionToString (Connexion *connexion*)

string representation of a connexion Definition at line 579 of file compile.c.

Referenced by comTranslateConnexion(), and comTranslateConstraint().

5.5.5.10 int comConstraintDepth (Constraint *constraint*)

compute the number of variables needed to evaluate a constraint Definition at line 4698 of file compile.c.

References comFormulaDepth(), comTermDepth(), and max.

Referenced by comAnalyzeGrammar().

5.5.5.11 String comDirectionToString (Direction *direction*)

string representation of a direction Definition at line 597 of file compile.c.

Referenced by comTranslateConstraint(), and comTranslateDirection().

5.5.5.12 String comEscapeQuotes (String *str*)

escape quotes in strings Definition at line 394 of file compile.c.

References NULL.

Referenced by comTranslateConstraint(), comTranslateEquation(), comTranslateLookup(), comTranslateMatch(), comTranslateSubsumes(), comTranslateUnEquation(), comWriteError(), comWriteInitFunction(), and comWriteWarning().

5.5.5.13 void comFinalize (void)

finalize the compiler module Definition at line 5087 of file compile.c.

References comMakeInfoFree(), and makeInfo.

Referenced by cdgFinalize().

5.5.5.14 LevelValue comFindComparableLv (ConstraintNet net, LevelValue lv)

find a lv using the same modifier-lexem, label and modifiee-lexem Definition at line 5123 of file compile.c.

References ConstraintNet, NULL, and ConstraintNetStruct::values.

Referenced by comCompareAllLvs().

5.5.5.15 void comFinitGrammar (Input thisInput)

finalize a dll grammar Definition at line 5065 of file compile.c.

References CDG_ERROR, cdgPrintf(), FINIT_GRAMMAR, and NULL.

5.5.5.16 int comFormulaDepth (Formula formula)

compute the number of variables needed to evaluate a formula Definition at line 4709 of file compile.c.

References comPredicateDepth(), comTermDepth(), and max.

Referenced by comConstraintDepth(), and comTranslateFormula().

5.5.5.17 String comFormulaTypeToString (Formula formula)

string representation of a formula type Definition at line 535 of file compile.c.

Referenced by comReturnTypeInfoOfFormula(), and comTranslateFormula().

5.5.5.18 void comFree (Compiler thisCom)

destructor for a Compiler Definition at line 297 of file compile.c.

References cdgFreeString(), Compiler, CompilerStruct::connexions, CompilerStruct::directions, CompilerStruct::indentStrings, NULL, and CompilerStruct::strings.

Referenced by comCompile().

5.5.5.19 void comFreeApprover (Approver ap)

free an approver Definition at line 5111 of file compile.c.

References NULL, ApproverStruct::timer, and timerFree().

Referenced by comApprove().

5.5.5.20 int comFunctionDepth (String functor, List args)

compute the number of variables needed to evaluate a function Definition at line 4755 of file compile.c.

References comTermDepth(), max, and NULL.

Referenced by comTermDepth().

5.5.5.21 void comIndent (void)

increase indentation Definition at line 355 of file compile.c.

References com, CompilerStruct::indent, CompilerStruct::indentString, CompilerStruct::indentStrings, and min.

Referenced by comTranslateArithmetics(), comTranslateConnexion(), comTranslateConstraint(), comTranslateDirection(), comTranslateEquation(), comTranslateFormula(), comTranslateFunction(), comTranslateHas(), comTranslatePredicate(), comTranslateTopPeek(), comTranslateUnEquation(), comWriteDeclarations(), comWriteFinitFunction(), comWriteFunctions(), and comWriteInitFunction().

5.5.5.22 int comIndexOfConstraint (String id)

get the index of a constraint Definition at line 1238 of file compile.c.

References comWriteError(), and NULL.

Referenced by comTranslateConstraint().

5.5.5.23 int comIndexOfHierarchy (String id)

get the index of a hierarchy Definition at line 1217 of file compile.c.

References comWriteError(), and NULL.

Referenced by comTranslateMatch(), and comTranslateSubsumes().

5.5.5.24 int comIndexOfVarInfo (VarInfo varInfo)

get index of assigned variable Definition at line 1259 of file compile.c.

References com, and CompilerStruct::currentConstraint.

Referenced by comTranslateBottomPeek(), comTranslateConnexion(), comTranslateDirection(), comTranslateExists(), comTranslateLexemNodeAccess(), comTranslateLexicalAccess(), comTranslateString(), and comTranslateTopPeek().

5.5.5.25 Boolean comInitGrammar (Input thisInput)

initialize a dll grammar Definition at line 5043 of file compile.c.

References CDG_ERROR, cdgPrintf(), FALSE, INIT_GRAMMAR, NULL, and TRUE.

5.5.5.26 void comInitialize (void)

initialize the compiler module Definition at line 5018 of file compile.c.

References MakeInfoStruct::cc, MakeInfoStruct::cFlags, comMakeInfoNew(), MakeInfoStruct::includes, MakeInfoStruct::ld, MakeInfoStruct::ldFlags, MakeInfoStruct::ldLibs, makeInfo, and NULL.

Referenced by cdgInitialize().

5.5.5.27 Input comLoad (String *filename*)

load a dll grammar Definition at line 3790 of file compile.c.

References CDG_ERROR, CDG_INFO, cdgPrintf(), com, NULL, and CompilerStruct::soFileName.

Referenced by cmdCompile().

5.5.5.28 Boolean comMake (void)

compile a shared object from the recently translated code Definition at line 3751 of file compile.c.

References MakeInfoStruct::cc, CompilerStruct::ccFileName, CDG_DEBUG, CDG_ERROR, CDG_INFO, cdgFlush(), cdgPrintf(), MakeInfoStruct::cFlags, com, FALSE, MakeInfoStruct::includes, MakeInfoStruct::ld, MakeInfoStruct::ldFlags, MakeInfoStruct::ldLibs, makeInfo, NULL, CompilerStruct::objFileName, CompilerStruct::soFileName, and TRUE.

Referenced by comCompile().

5.5.5.29 void comMakeInfoFree ([MakeInfo](#) *thisMakeInfo*)

destructor of a MakeInfo Definition at line 338 of file compile.c.

References MakeInfoStruct::cc, cdgFreeString(), MakeInfoStruct::cFlags, MakeInfoStruct::includes, MakeInfoStruct::ld, MakeInfoStruct::ldFlags, MakeInfoStruct::ldLibs, and MakeInfo.

Referenced by comFinalize().

5.5.5.30 [MakeInfo](#) comMakeInfoNew (void)

constructor for a MakeInfo Definition at line 321 of file compile.c.

References MakeInfoStruct::cc, MakeInfoStruct::cFlags, MakeInfoStruct::includes, MakeInfoStruct::ld, MakeInfoStruct::ldFlags, MakeInfoStruct::ldLibs, MakeInfo, and NULL.

Referenced by comInitialize().

5.5.5.31 int comMaxLookupStrings (Constraint *constraint*)

compute the number of strings needed for evaluationg 'lookup' functions in a constraint Definition at line 4616 of file compile.c.

References comMaxLookupStringsInFormula(), comMaxLookupStringsInTerm(), and max.

Referenced by comAnalyzeGrammar().

5.5.5.32 int comMaxLookupStringsInFormula (Formula *formula*)

compute the number of strings needed for evaluationg 'lookup' functions in a formla Definition at line 4627 of file compile.c.

References comMaxLookupStringsInTerm(), and max.

Referenced by comMaxLookupStrings().

5.5.5.33 int comMaxLookupStringsInTerm (Term term)

compute the number of strings needed for evaluationg 'lookup' functions in a term Definition at line 4665 of file compile.c.

References NULL.

Referenced by comMaxLookupStrings(), and comMaxLookupStringsInFormula().

5.5.5.34 Compiler comNew (void)

constructor for a Compiler Definition at line 249 of file compile.c.

References CompilerStruct::ccFileName, Compiler, CompilerStruct::connexions, CompilerStruct::current-Constraint, CompilerStruct::directions, FALSE, CompilerStruct::indent, CompilerStruct::indentString, CompilerStruct::indentStrings, CompilerStruct::needsIndent, NULL, CompilerStruct::objFileName, CompilerStruct::soFileName, CompilerStruct::strings, CompilerStruct::translateOnly, and TRUE.

Referenced by comCompile().

5.5.5.35 Approver comNewApprover (void)

allocate a new approver Definition at line 5094 of file compile.c.

References Approver, ApproverStruct::noRounds, ApproverStruct::timer, timerNew(), ApproverStruct::totalCompiledBinaryTime, ApproverStruct::totalCompiledUnaryTime, ApproverStruct::totalInterpretedBinaryTime, and ApproverStruct::totalInterpretedUnaryTime.

Referenced by comApprove().

5.5.5.36 void comOutdent (void)

decrease indentation Definition at line 365 of file compile.c.

References com, CompilerStruct::indent, CompilerStruct::indentString, CompilerStruct::indentStrings, and min.

Referenced by comTranslateArithmetics(), comTranslateConnexion(), comTranslateConstraint(), comTranslateDirection(), comTranslateEquation(), comTranslateFormula(), comTranslateFunction(), comTranslateHas(), comTranslatePredicate(), comTranslateTopPeek(), comTranslateUnEquation(), comWriteDeclarations(), comWriteFinitFunction(), comWriteFunctions(), and comWriteInitFunction().

5.5.5.37 int comPredicateDepth (String functor, List args)

compute the number of variables needed to evaluate a predicate Definition at line 4816 of file compile.c.

References comTermDepth(), max, and NULL.

Referenced by comFormulaDepth().

5.5.5.38 void comPrint (String format, ...)

no good c program without its own printf Definition at line 462 of file compile.c.

References CompilerStruct::ccFile, com, FALSE, CompilerStruct::indentString, and CompilerStruct::needsIndent.

Referenced by `comTranslateAbs()`, `comTranslateArithmetics()`, `comTranslateBetween()`, `comTranslateBottomPeek()`, `comTranslateChunkHead()`, `comTranslateConnected()`, `comTranslateConstraint()`, `comTranslateDirection()`, `comTranslateDistance()`, `comTranslateEquation()`, `comTranslateExists()`, `comTranslateFormula()`, `comTranslateFunction()`, `comTranslateGuard()`, `comTranslateHas()`, `comTranslateHeight()`, `comTranslateIs()`, `comTranslateLexemNodeAccess()`, `comTranslateLexicalAccess()`, `comTranslateLookup()`, `comTranslateMatch()`, `comTranslateMinMax()`, `comTranslateNumber()`, `comTranslateParens()`, `comTranslateParent()`, `comTranslatePhrasequotes()`, `comTranslatePredicate()`, `comTranslatePrint()`, `comTranslatePts()`, `comTranslateQuotes()`, `comTranslateStartStop()`, `comTranslateString()`, `comTranslateSubsumes()`, `comTranslateTerm()`, `comTranslateTopPeek()`, `comTranslateUnder()`, `comTranslateUnEquation()`, and `comWriteDeclarations()`.

5.5.5.39 void `comPrintln` (String *format*, ...)

grant a newline at the end Definition at line 376 of file `compile.c`.

References `CompilerStruct::ccFile`, `com`, `CompilerStruct::indentString`, `CompilerStruct::needsIndent`, and `TRUE`.

Referenced by `comTranslateAbs()`, `comTranslateArithmetics()`, `comTranslateBetween()`, `comTranslateBottomPeek()`, `comTranslateChunkHead()`, `comTranslateConnected()`, `comTranslateConnexion()`, `comTranslateConstraint()`, `comTranslateDirection()`, `comTranslateDistance()`, `comTranslateEquation()`, `comTranslateExists()`, `comTranslateFormula()`, `comTranslateFunction()`, `comTranslateGuard()`, `comTranslateHas()`, `comTranslateHeight()`, `comTranslateIs()`, `comTranslateLookup()`, `comTranslateMatch()`, `comTranslateMinMax()`, `comTranslateParens()`, `comTranslateParent()`, `comTranslatePhrasequotes()`, `comTranslatePredicate()`, `comTranslatePrint()`, `comTranslatePts()`, `comTranslateQuotes()`, `comTranslateStartStop()`, `comTranslateSubsumes()`, `comTranslateTerm()`, `comTranslateTopPeek()`, `comTranslateUnder()`, `comTranslateUnEquation()`, `comWriteDeclarations()`, `comWriteFinitFunction()`, `comWriteFunctions()`, `comWriteHeader()`, and `comWriteInitFunction()`.

5.5.5.40 int `comRegisterString` (String *string*)

register a string to be allocated statically later returns the index in strings Definition at line 1272 of file `compile.c`.

References `com`, and `CompilerStruct::strings`.

Referenced by `comTranslateString()`.

5.5.5.41 **ReturnType** `comReturnTypeOfFormula` (Formula *formula*)

check the return type of a formula Definition at line 4477 of file `compile.c`.

References `comFormulaTypeToString()`, `comReturnTypeOfPredicate()`, `comReturnTypeOfTerm()`, `comReturnTypeToString()`, `comWriteWarning()`, `ReturnType`, `RTBoolean`, `RTError`, `RTLexemNode`, `RTLexemPosition`, `RTNoError`, `RTNumber`, and `RTPeek`.

Referenced by `comTranslateFormula()`.

5.5.5.42 **ReturnType** `comReturnTypeOfFunction` (String *functor*, List *args*)

check the return type of a function Definition at line 4137 of file `compile.c`.

References `comReturnTypeOfTerm()`, `comReturnTypeToString()`, `comWriteError()`, `comWriteWarning()`, `FALSE`, `NULL`, `ReturnType`, `RTError`, `RTLexemNode`, `RTLexemPosition`, `RTList`, `RTNoError`, `RTNumber`, `RTPeek`, and `RTString`.

Referenced by `comReturn.TypeOfTerm()`.

5.5.5.43 **ReturnType** `comReturn.TypeOfPredicate` (*String functor*, *List args*)

check the return type of a predicate Definition at line 3820 of file `compile.c`.

References `comReturn.TypeOfTerm()`, `comReturn.TypeToString()`, `comWriteError()`, `comWriteWarning()`, `NULL`, `ReturnType`, `RTBoolean`, `RTError`, `RTLexemNode`, `RTLexemPosition`, `RTNoError`, `RTNumber`, `RTPeek`, and `RTString`.

Referenced by `comReturn.TypeOfFormula()`.

5.5.5.44 **ReturnType** `comReturn.TypeOfTerm` (*Term term*)

check the return type of a term Definition at line 4394 of file `compile.c`.

References `comReturn.TypeOfFunction()`, `comReturn.TypeToString()`, `comTermTypeToString()`, `comWriteWarning()`, `NULL`, `ReturnType`, `RTAVNode`, `RTError`, `RTLexemNode`, `RTLexemPosition`, `RTNoError`, `RTNumber`, `RTPeek`, and `RTString`.

Referenced by `comReturn.TypeOfFormula()`, `comReturn.TypeOfFunction()`, `comReturn.TypeOfPredicate()`, `comTranslateAbs()`, `comTranslateArithmetics()`, `comTranslateBetween()`, `comTranslateChunkHead()`, `comTranslateConnected()`, `comTranslateConstraint()`, `comTranslateDistance()`, `comTranslateEquation()`, `comTranslateGuard()`, `comTranslateHas()`, `comTranslateIs()`, `comTranslateLookup()`, `comTranslateMatch()`, `comTranslateMinMax()`, `comTranslateParens()`, `comTranslateParent()`, `comTranslatePhrasequotes()`, `comTranslatePrint()`, `comTranslateQuotes()`, `comTranslateStartStop()`, `comTranslateSubsumes()`, `comTranslateTerm()`, `comTranslateUnder()`, and `comTranslateUnEquation()`.

5.5.5.45 **String** `comReturn.TypeToString` (*ReturnType returnType*)

string representation of a returntype Definition at line 479 of file `compile.c`.

References `RTAVNode`, `RTBoolean`, `RTConjunction`, `RTDisjunction`, `RTError`, `RTGraphemNode`, `RTLexemNode`, `RTLexemPosition`, `RTLList`, `RTNoError`, `RTNumber`, `RTPeek`, and `RTString`.

Referenced by `comReturn.TypeOfFormula()`, `comReturn.TypeOfFunction()`, `comReturn.TypeOfPredicate()`, `comReturn.TypeOfTerm()`, `comTranslateAbs()`, `comTranslateArithmetics()`, `comTranslateChunkHead()`, `comTranslateDistance()`, `comTranslateEquation()`, `comTranslateGuard()`, `comTranslateStartStop()`, and `comTranslateUnEquation()`.

5.5.5.46 **int** `comTermDepth` (*Term term*)

compute the number of variables needed to evaluate a term Definition at line 4871 of file `compile.c`.

References `comFunctionDepth()`, `max`, and `NULL`.

Referenced by `comConstraintDepth()`, `comFormulaDepth()`, `comFunctionDepth()`, and `comPredicateDepth()`.

5.5.5.47 **String** `comTermTypeToString` (*Term term*)

string representation of a term type Definition at line 514 of file `compile.c`.

Referenced by `comReturn.TypeOfTerm()`.

5.5.5.48 Boolean `comTranslate` (void)

translate the current cdg to C Definition at line 3721 of file `compile.c`.

References `CompilerStruct::ccFileName`, `CDG_INFO`, `cdgFlush()`, `cdgPrintf()`, `com`, `comAnalyzeGrammar()`, `comTranslateBinaryConstraints()`, `comTranslateUnaryConstraints()`, `comWriteDeclarations()`, `comWriteFunctions()`, `comWriteHeader()`, and `TRUE`.

Referenced by `comCompile()`.

5.5.5.49 void `comTranslateAbs` (List *args*, int *tmpIndex*)

translate the abs function Definition at line 1482 of file `compile.c`.

References `comPrint()`, `comPrintln()`, `comReturnTypeInfoOfTerm()`, `comReturnTypeInfoToString()`, `comTranslateLexicalAccess()`, `comTranslateNumber()`, `comTranslateTerm()`, `comWriteError()`, `FALSE`, `ReturnTypeInfo`, `RTNoError`, `RTNumber`, `RTPeek`, and `TRUE`.

Referenced by `comTranslateFunction()`.

5.5.5.50 void `comTranslateArithmetics` (Term *term*, int *tmpIndex*)

translate the arithmetic term operations Definition at line 2010 of file `compile.c`.

References `comIndent()`, `comOutdent()`, `comPrint()`, `comPrintln()`, `comReturnTypeInfoOfTerm()`, `comReturnTypeInfoToString()`, `comTranslateLexicalAccess()`, `comTranslateNumber()`, `comTranslateTerm()`, `comWriteError()`, `FALSE`, `ReturnTypeInfo`, `RTNoError`, `RTNumber`, `RTPeek`, and `TRUE`.

Referenced by `comTranslateTerm()`.

5.5.5.51 void `comTranslateBetween` (String *functor*, List *args*, int *tmpIndex*)

translate the 'between' predicate Definition at line 2634 of file `compile.c`.

References `comPrint()`, `comPrintln()`, `comReturnTypeInfoOfTerm()`, `comTranslateLexemNodeAccess()`, `comTranslateNumber()`, `comTranslateString()`, `comTranslateTerm()`, `ReturnTypeInfo`, `RTLexemNode`, `RTLexemPosition`, `RTNumber`, `RTString`, and `TRUE`.

Referenced by `comTranslatePredicate()`.

5.5.5.52 void `comTranslateBinaryConstraints` (void)

translate each binary constraint into a separate c-function Definition at line 3704 of file `compile.c`.

References `com`, `comTranslateConstraint()`, and `CompilerStruct::indent`.

Referenced by `comTranslate()`.

5.5.5.53 void `comTranslateBottomPeek` (Term *term*, int *tmpIndex*)

translate a bottom peek Definition at line 2119 of file `compile.c`.

References `comIndexInfoOfVarInfo()`, `comPrint()`, `comPrintln()`, `comTranslateLexicalAccess()`, and `NULL`.

Referenced by `comTranslateTerm()`.

5.5.5.54 void comTranslateChunkHead (String functor, List args, int tmpIndex)

translate the chunk_head predicate Definition at line 2560 of file compile.c.

References comPrint(), comPrintln(), comReturnTypeInfo(), comReturnTypeInfoToString(), comTranslateLexemNodeAccess(), comWriteError(), ReturnTypeInfo, RTLexemNode, RTLexemPosition, and RTNoError.

Referenced by comTranslatePredicate().

5.5.5.55 void comTranslateConnected (String functor, List args, int tmpIndex)

translate the 'connected' predicate Definition at line 2768 of file compile.c.

References com, comPrint(), comPrintln(), comReturnTypeInfo(), comTranslateLexemNodeAccess(), CompilerStruct::currentConstraint, ReturnTypeInfo, RTLexemNode, and RTLexemPosition.

Referenced by comTranslatePredicate().

5.5.5.56 void comTranslateConnexion (Formula formula)

translate a connexion in case of the unsymmetric connexions 'Under' & 'Over' a distinction between the order (lv1, lv2) and (lv2, lv1) is made. in the other cases this doesn't matter. Definition at line 3563 of file compile.c.

References comConnexionToString(), comIndent(), comIndexofVarInfo(), comOutdent(), and comPrintln().

Referenced by comTranslateFormula().

5.5.5.57 void comTranslateConstraint (Constraint constraint)

translate a constraint into a c-function Definition at line 3617 of file compile.c.

References com, comConnexionToString(), comDirectionToString(), comEscapeQuotes(), comIndent(), comIndexofConstraint(), comOutdent(), comPrint(), comPrintln(), comReturnTypeInfo(), comTranslateFormula(), comTranslateLexicalAccess(), comTranslateNumber(), comTranslateTerm(), CompilerStruct::currentConstraint, FALSE, NULL, ReturnTypeInfo, RTNoError, RTPeek, and TRUE.

Referenced by comTranslateBinaryConstraints(), and comTranslateUnaryConstraints().

5.5.5.58 void comTranslateDirection (Formula formula)

translate the direction formulas Definition at line 3406 of file compile.c.

References comDirectionToString(), comIndent(), comIndexofVarInfo(), comOutdent(), comPrint(), comPrintln(), and comWriteError().

Referenced by comTranslateFormula().

5.5.5.59 void comTranslateDistance (List args, int tmpIndex)

translate the distance function Definition at line 1435 of file compile.c.

References comPrint(), comPrintln(), comReturnTypeInfo(), comReturnTypeInfoToString(), comTranslateLexemNodeAccess(), comTranslateTerm(), comWriteError(), FALSE, ReturnTypeInfo, RTLexemNode, RTLexemPosition, RTNoError, and TRUE.

Referenced by `comTranslateFunction()`.

5.5.5.60 void `comTranslateEquation (Formula formula, int tmpIndex)`

translate the equal and not equal formulas Definition at line 3061 of file `compile.c`.

References `com`, `comEscapeQuotes()`, `comIndent()`, `comOutdent()`, `comPrint()`, `comPrintln()`, `comReturn-
TypeOfTerm()`, `comReturnTypeToString()`, `comTranslateLexemNodeAccess()`, `comTranslateLexical-
Access()`, `comTranslateNumber()`, `comTranslateString()`, `comTranslateTerm()`, `comWriteError()`, `com-
WriteWarning()`, `CompilerStruct::currentConstraint`, `FALSE`, `ReturnType`, `RTLexemNode`, `RTLexem-
Position`, `RTNoError`, `RTNumber`, `RTPeek`, `RTString`, and `TRUE`.

Referenced by `comTranslateFormula()`.

5.5.5.61 void `comTranslateExists (List args)`

translate the exists predicate Definition at line 2504 of file `compile.c`.

References `comIndexOfVarInfo()`, `comPrint()`, `comPrintln()`, and `NULL`.

Referenced by `comTranslatePredicate()`.

5.5.5.62 void `comTranslateFormula (Formula formula, int tmpIndex)`

translate a formula to c code Definition at line 3459 of file `compile.c`.

References `com`, `comFormulaDepth()`, `comFormulaTypeToString()`, `comIndent()`, `comOutdent()`, `com-
Print()`, `comPrintln()`, `comReturnTypeOfFormula()`, `comTranslateConnexion()`, `comTranslateDirection()`,
`comTranslateEquation()`, `comTranslatePredicate()`, `comTranslateUnEquation()`, `comWriteWarning()`,
`CompilerStruct::currentFormula`, `ReturnType`, `RTBoolean`, `RTError`, and `RTNoError`.

Referenced by `comTranslateConstraint()`.

5.5.5.63 void `comTranslateFunction (String functor, List args, int tmpIndex)`

translate a function to c code Definition at line 1361 of file `compile.c`.

References `comIndent()`, `comOutdent()`, `comPrint()`, `comPrintln()`, `comTranslateAbs()`, `comTranslate-
Distance()`, `comTranslateHeight()`, `comTranslateLookup()`, `comTranslateMatch()`, `comTranslateMin-
Max()`, `comTranslateParens()`, `comTranslateParent()`, `comTranslatePhrasequotes()`, `comTranslatePts()`,
`comTranslateQuotes()`, and `comWriteError()`.

Referenced by `comTranslateTerm()`.

5.5.5.64 void `comTranslateGuard (String functor, List args, int tmpIndex)`

translate the root, spec and nonspec guard-predicates Definition at line 2989 of file `compile.c`.

References `comPrint()`, `comPrintln()`, `comReturnTypeOfTerm()`, `comReturnTypeToString()`, `comTranslate-
LexemNodeAccess()`, `comTranslateTerm()`, `comWriteError()`, `ReturnType`, `RTLexemNode`, `RTLexem-
Position`, `RTNoError`, and `TRUE`.

Referenced by `comTranslatePredicate()`.

5.5.5.65 void comTranslateHas (String functor, List args, int tmpIndex)

translate the 'has' predicate Definition at line 2818 of file compile.c.

References com, comIndent(), comOutdent(), comPrint(), comPrintln(), comReturnTypeOfTerm(), comTranslateLexemNodeAccess(), comTranslateLexicalAccess(), comTranslateNumber(), comTranslateString(), comTranslateTerm(), CompilerStruct::currentFormula, FALSE, ReturnType, RTLexemPosition, RTPeek, RTString, and TRUE.

Referenced by comTranslatePredicate().

5.5.5.66 void comTranslateHeight (List args, int tmpIndex)

translate the 'height' function Definition at line 1681 of file compile.c.

References comPrint(), comPrintln(), comTranslateLexemNodeAccess(), comWriteError(), and NULL.

Referenced by comTranslateFunction().

5.5.5.67 void comTranslateIs (String functor, List args, int tmpIndex)

translate the 'is' predicate Definition at line 2715 of file compile.c.

References com, comPrint(), comPrintln(), comReturnTypeOfTerm(), comTranslateLexemNodeAccess(), comTranslateString(), comWriteWarning(), CompilerStruct::currentConstraint, ReturnType, RTLexemNode, and RTLexemPosition.

Referenced by comTranslatePredicate().

5.5.5.68 void comTranslateLexemNodeAccess (Term term, int tmpIndex)

translate a ^id and @id Definition at line 1292 of file compile.c.

References comIndexOfVarInfo(), comPrint(), and comWriteError().

Referenced by comTranslateBetween(), comTranslateChunkHead(), comTranslateConnected(), comTranslateDistance(), comTranslateEquation(), comTranslateGuard(), comTranslateHas(), comTranslateHeight(), comTranslateIs(), comTranslateParens(), comTranslateParent(), comTranslatePhrasequotes(), comTranslatePts(), comTranslateQuotes(), comTranslateStartStop(), and comTranslateUnder().

5.5.5.69 void comTranslateLexicalAccess (Term term)

translate a lexical access Definition at line 1328 of file compile.c.

References comIndexOfVarInfo(), comPrint(), and comWriteError().

Referenced by comTranslateAbs(), comTranslateArithmetics(), comTranslateBottomPeek(), comTranslateConstraint(), comTranslateEquation(), comTranslateHas(), comTranslateLookup(), comTranslateMatch(), comTranslatePrint(), comTranslateSubsumes(), comTranslateTopPeek(), and comTranslateUnEquation().

5.5.5.70 void comTranslateLookup (List args, int tmpIndex)

translate the lookup function Definition at line 1536 of file compile.c.

References `com`, `comEscapeQuotes()`, `comPrint()`, `comPrintln()`, `comReturnTypeInfoOfTerm()`, `comTranslateLexicalAccess()`, `comTranslateString()`, `comTranslateTerm()`, `CompilerStruct::currentConstraint`, `FALSE`, `ReturnTypeInfo`, `RTNoError`, `RTPeek`, and `RTString`.

Referenced by `comTranslateFunction()`.

5.5.5.71 void comTranslateMatch (List args, int tmpIndex)

translate the match function Definition at line 1763 of file `compile.c`.

References `com`, `comEscapeQuotes()`, `comIndexofHierarchy()`, `comPrint()`, `comPrintln()`, `comReturnTypeInfoOfTerm()`, `comTranslateLexicalAccess()`, `comTranslateString()`, `comTranslateTerm()`, `comWriteError()`, `CompilerStruct::currentConstraint`, `FALSE`, `ReturnTypeInfo`, `RTList`, `RTNoError`, `RTPeek`, `RTString`, and `TRUE`.

Referenced by `comTranslateFunction()`.

5.5.5.72 void comTranslateMinMax (String functor, List args, int tmpIndex)

translate the min and max function Definition at line 1708 of file `compile.c`.

References `comPrint()`, `comPrintln()`, `comReturnTypeInfoOfTerm()`, `comTranslateNumber()`, `comTranslateTerm()`, `FALSE`, `NULL`, `ReturnTypeInfo`, `RTNoError`, `RTPeek`, and `TRUE`.

Referenced by `comTranslateFunction()`.

5.5.5.73 void comTranslateNumber (Term term, int tmpIndex)

translate a number access Definition at line 1902 of file `compile.c`.

References `comPrint()`.

Referenced by `comTranslateAbs()`, `comTranslateArithmetics()`, `comTranslateBetween()`, `comTranslateConstraint()`, `comTranslateEquation()`, `comTranslateHas()`, `comTranslateMinMax()`, and `comTranslateUnEquation()`.

5.5.5.74 void comTranslateParens (List args, int tmpIndex)

translate the parens function Definition at line 1610 of file `compile.c`.

References `comPrint()`, `comPrintln()`, `comReturnTypeInfoOfTerm()`, `comTranslateLexemNodeAccess()`, `ReturnTypeInfo`, and `RTLexemPosition`.

Referenced by `comTranslateFunction()`.

5.5.5.75 void comTranslateParent (List args, int tmpIndex)

translate the parent function Definition at line 1634 of file `compile.c`.

References `com`, `comPrint()`, `comPrintln()`, `comReturnTypeInfoOfTerm()`, `comTranslateLexemNodeAccess()`, `CompilerStruct::currentConstraint`, `ReturnTypeInfo`, and `RTLexemPosition`.

Referenced by `comTranslateFunction()`.

5.5.5.76 void comTranslatePhrasequotes (List args, int tmpIndex)

translate the phrasequotes function Definition at line 1512 of file compile.c.

References comPrint(), comPrintln(), comReturnTypeInfoOfTerm(), comTranslateLexemNodeAccess(), ReturnTypeInfo, and RTLexemPosition.

Referenced by comTranslateFunction().

5.5.5.77 void comTranslatePredicate (String functor, List args, int tmpIndex)

translate a predicate to c code Definition at line 2319 of file compile.c.

References comIndent(), comOutdent(), comPrint(), comPrintln(), comTranslateBetween(), comTranslateChunkHead(), comTranslateConnected(), comTranslateExists(), comTranslateGuard(), comTranslateHas(), comTranslateIs(), comTranslatePrint(), comTranslateStartStop(), comTranslateSubsumes(), and comTranslateUnder().

Referenced by comTranslateFormula().

5.5.5.78 void comTranslatePrint (List args, int tmpIndex)

translate the print predicate Definition at line 2294 of file compile.c.

References comPrint(), comPrintln(), comReturnTypeInfoOfTerm(), comTranslateLexicalAccess(), comTranslateTerm(), FALSE, NULL, ReturnTypeInfo, RTNoError, RTPeek, and TRUE.

Referenced by comTranslatePredicate().

5.5.5.79 void comTranslatePts (List args, int tmpIndex)

translate the pts function Definition at line 1662 of file compile.c.

References comPrint(), comPrintln(), and comTranslateLexemNodeAccess().

Referenced by comTranslateFunction().

5.5.5.80 void comTranslateQuotes (List args, int tmpIndex)

translate the quotes function Definition at line 1586 of file compile.c.

References comPrint(), comPrintln(), comReturnTypeInfoOfTerm(), comTranslateLexemNodeAccess(), ReturnTypeInfo, and RTLexemPosition.

Referenced by comTranslateFunction().

5.5.5.81 void comTranslateStartStop (String functor, List args, int tmpIndex)

translate the start and stop predicate Definition at line 3024 of file compile.c.

References comPrint(), comPrintln(), comReturnTypeInfoOfTerm(), comReturnTypeInfoToString(), comTranslateLexemNodeAccess(), comTranslateTerm(), comWriteError(), ReturnTypeInfo, RTLexemNode, RTLexemPosition, RTNoError, and TRUE.

Referenced by comTranslatePredicate().

5.5.5.82 void comTranslateString (Term term, int tmpIndex)

translate a string access Definition at line 1916 of file compile.c.

References comIndexOfVarInfo(), comPrint(), and comRegisterString().

Referenced by comTranslateBetween(), comTranslateEquation(), comTranslateHas(), comTranslateIs(), comTranslateLookup(), comTranslateMatch(), and comTranslateSubsumes().

5.5.5.83 void comTranslateSubsumes (String functor, List args, int tmpIndex)

translate the subsumes and compatible predicates Definition at line 2390 of file compile.c.

References com, comEscapeQuotes(), comIndexOfHierarchy(), comPrint(), comPrintln(), comReturnTypeOfTerm(), comTranslateLexicalAccess(), comTranslateString(), comTranslateTerm(), comWriteError(), CompilerStruct::currentConstraint, Return Type, RTNoError, RTPeek, RTString, and TRUE.

Referenced by comTranslatePredicate().

5.5.5.84 void comTranslateTerm (Term term, Boolean insideFormula, Boolean needsSeparator, int tmpIndex)

translate a term to c code Definition at line 1939 of file compile.c.

References comPrint(), comPrintln(), comReturnTypeOfTerm(), comTranslateArithmetics(), comTranslateBottomPeek(), comTranslateFunction(), comTranslateTopPeek(), FALSE, Return Type, and RTNoError.

Referenced by comTranslateAbs(), comTranslateArithmetics(), comTranslateBetween(), comTranslateConstraint(), comTranslateDistance(), comTranslateEquation(), comTranslateGuard(), comTranslateHas(), comTranslateLookup(), comTranslateMatch(), comTranslateMinMax(), comTranslatePrint(), comTranslateStartStop(), comTranslateSubsumes(), and comTranslateUnEquation().

5.5.5.85 void comTranslateTopPeek (Term term, int tmpIndex)

translate a top peek Definition at line 2192 of file compile.c.

References comIndent(), comIndexOfVarInfo(), comOutdent(), comPrint(), comPrintln(), comTranslateLexicalAccess(), and NULL.

Referenced by comTranslateTerm().

5.5.5.86 void comTranslateUnaryConstraints (void)

translate each unary constraint into a separate c-function Definition at line 3600 of file compile.c.

References com, comTranslateConstraint(), and CompilerStruct::indent.

Referenced by comTranslate().

5.5.5.87 void comTranslateUnder (String functor, List args, int tmpIndex)

translate the 'under' predicate Definition at line 2592 of file compile.c.

References comPrint(), comPrintln(), comReturnTypeOfTerm(), comTranslateLexemNodeAccess(), comWriteError(), Return Type, RTLexemNode, and RTNoError.

Referenced by `comTranslatePredicate()`.

5.5.5.88 void `comTranslateUnEquation` (Formula *formula*, int *tmpIndex*)

translate the `<`, `<=`, `>`, `>=` formulas Definition at line 3289 of file `compile.c`.

References `com`, `comEscapeQuotes()`, `comIndent()`, `comOutdent()`, `comPrint()`, `comPrintln()`, `comReturn-
TypeOfTerm()`, `comReturnTypeToString()`, `comTranslateLexicalAccess()`, `comTranslateNumber()`, `com-
TranslateTerm()`, `comWriteError()`, `CompilerStruct::currentConstraint`, `ReturnType`, `RTNoError`, `RTNum-
ber`, `RTPeek`, and `TRUE`.

Referenced by `comTranslateFormula()`.

5.5.5.89 String `comValueTypeToString` (Value *value*)

string representation of a value type Definition at line 560 of file `compile.c`.

5.5.5.90 void `comWriteDeclarations` (void)

write declaration section Definition at line 650 of file `compile.c`.

References `com`, `comIndent()`, `comOutdent()`, `comPrint()`, `comPrintln()`, `FINIT_GRAMMAR`, `INIT_-
GRAMMAR`, `CompilerStruct::maxValues`, and `NULL`.

Referenced by `comTranslate()`.

5.5.5.91 void `comWriteError` (String *format*, ...)

write a preprocessor error Definition at line 440 of file `compile.c`.

References `CompilerStruct::ccFile`, `com`, `comEscapeQuotes()`, `CompilerStruct::currentConstraint`, `CompilerStruct::needsIndent`, and `TRUE`.

Referenced by `comIndexOfConstraint()`, `comIndexOfHierarchy()`, `comReturnTypeOfFunction()`, `com-
ReturnTypeOfPredicate()`, `comTranslateAbs()`, `comTranslateArithmetics()`, `comTranslateChunkHead()`, `com-
TranslateDirection()`, `comTranslateDistance()`, `comTranslateEquation()`, `comTranslateFunction()`, `com-
TranslateGuard()`, `comTranslateHeight()`, `comTranslateLexemNodeAccess()`, `comTranslateLexical-
Access()`, `comTranslateMatch()`, `comTranslateStartStop()`, `comTranslateSubsumes()`, `comTranslate-
Under()`, and `comTranslateUnEquation()`.

5.5.5.92 void `comWriteFinitFunction` (void)

write the grammar finalization function Definition at line 1086 of file `compile.c`.

References `com`, `comIndent()`, `comOutdent()`, `comPrintln()`, `FINIT_GRAMMAR`, and `Compiler-
Struct::strings`.

Referenced by `comWriteFunctions()`.

5.5.5.93 void `comWriteFunctions` (void)

write helper functions Definition at line 755 of file `compile.c`.

References `com`, `comIndent()`, `comOutdent()`, `comPrintln()`, `comWriteFinitFunction()`, `comWriteInitFunction()`, and `CompilerStruct::indent`.

Referenced by `comTranslate()`.

5.5.5.94 void comWriteHeader (void)

write header Definition at line 612 of file `compile.c`.

References `comPrintln()`, and `NULL`.

Referenced by `comTranslate()`.

5.5.5.95 void comWriteInitFunction (void)

write the grammar initialization function Definition at line 1122 of file `compile.c`.

References `com`, `comEscapeQuotes()`, `comIndent()`, `comOutdent()`, `comPrintln()`, `INIT_GRAMMAR`, `CompilerStruct::maxLookupStrings`, `NULL`, and `CompilerStruct::strings`.

Referenced by `comWriteFunctions()`.

5.5.5.96 void comWriteWarning (String format, ...)

write a preprocessor warning Definition at line 418 of file `compile.c`.

References `CompilerStruct::ccFile`, `com`, `comEscapeQuotes()`, `CompilerStruct::currentConstraint`, `CompilerStruct::needsIndent`, and `TRUE`.

Referenced by `comReturnTypeInfoOfFormula()`, `comReturnTypeInfoOfFunction()`, `comReturnTypeInfoOfPredicate()`, `comReturnTypeInfoOfTerm()`, `comTranslateEquation()`, `comTranslateFormula()`, and `comTranslateIs()`.

5.5.6 Variable Documentation

5.5.6.1 Compiler com [static]

this is allocated on every call to `comCompile` Definition at line 132 of file `compile.c`.

Referenced by `comAnalyzeGrammar()`, `comCompile()`, `comIndent()`, `comIndexOfVarInfo()`, `comLoad()`, `comMake()`, `comOutdent()`, `comPrint()`, `comPrintln()`, `comRegisterString()`, `comTranslate()`, `comTranslateBinaryConstraints()`, `comTranslateConnected()`, `comTranslateConstraint()`, `comTranslateEquation()`, `comTranslateFormula()`, `comTranslateHas()`, `comTranslateIs()`, `comTranslateLookup()`, `comTranslateMatch()`, `comTranslateParent()`, `comTranslateSubsumes()`, `comTranslateUnaryConstraints()`, `comTranslateUnEquation()`, `comWriteDeclarations()`, `comWriteError()`, `comWriteFinitFunction()`, `comWriteFunctions()`, `comWriteInitFunction()`, and `comWriteWarning()`.

5.5.6.2 MakeInfo makeInfo [static]

this is allocated once on `comInitialize` Definition at line 133 of file `compile.c`.

Referenced by `comCompile()`, `comFinalize()`, `comInitialize()`, and `comMake()`.

5.6 Constraintnet - maintainance of constraint nets

5.6.1 Detailed Description

Author:

Ingo Schroeder

Date:

6/3/97

Data Structures

- struct [ConstraintEdgeStruct](#)
- struct [ConstraintNetStruct](#)
- struct [ConstraintNodeStruct](#)
- struct [ConstraintViolationStruct](#)
- struct [NodeBindingStruct](#)

Typedefs

- typedef [ConstraintEdgeStruct](#) * [ConstraintEdge](#)
- typedef [ConstraintEdgeStruct](#) [ConstraintEdgeStruct](#)
- typedef [ConstraintNetStruct](#) * [ConstraintNet](#)
- typedef [ConstraintNodeStruct](#) * [ConstraintNode](#)
- typedef [ConstraintViolationStruct](#) * [ConstraintViolation](#)
- typedef [NodeBindingStruct](#) * [NodeBinding](#)

Enumerations

- enum [CnEdgesType](#) { [cnEdgesOn](#), [cnEdgesOff](#), [cnEdgesFew](#), [cnEdgesAll](#) }

Functions

- [ConstraintNet](#) [cnBuild](#) ([Lattice](#) lat, Boolean buildLVs)
- void [cnBuildEdges](#) ([ConstraintNet](#) net)
- [ConstraintNet](#) [cnBuildFinal](#) ([ConstraintNet](#) net, Boolean buildLVs)
- [ConstraintNet](#) [cnBuildInit](#) ()
- Boolean [cnBuildIter](#) ([ConstraintNet](#) net, [GraphemNode](#) gn, Boolean buildLVs)
- void [cnBuildLevelValues](#) ([ConstraintNode](#) node, Level level, [GraphemNode](#) modifier, [GraphemNode](#) modiffee)
- void [cnBuildLv](#) ([ConstraintNode](#) node, List modifiers, Level level, String label, List modifiees)
- Boolean [cnBuildNodes](#) ([ConstraintNet](#) net, Boolean buildLVs)
- Boolean [cnBuildTriple](#) ([ConstraintNet](#) net, int a, int b, int levelNo)
- Boolean [cnBuildUpdateArcs](#) ([ConstraintNet](#) net, List listArcs)
- void [cnCallback](#) (String name, float *var)
- Boolean [cnCompareViolation](#) ([ConstraintViolation](#) a, [ConstraintViolation](#) b)
- Boolean [cnConnectedByArc](#) ([ConstraintNode](#) a, [ConstraintNode](#) b)
- void [cnDelete](#) ([ConstraintNet](#) net)
- void [cnDeleteAllLVs](#) ([ConstraintNet](#) net)

- void `cnDeleteBinding` (`NodeBinding` nb)
- void `cnDeleteEdge` (`ConstraintEdge` edge)
- void `cnDeleteNode` (`ConstraintNode` node)
- `ConstraintNet` `cnFindNet` (`String` id)
- `ConstraintNode` `cnFindNode` (`ConstraintNet` net, `LevelValue` lv)
- `GraphemNode` `cnGetGraphemNodeFromArc` (`ConstraintNet` net, `Arc` arc)
- `Lattice` `cnGetLattice` (`ConstraintNet` cn)
- void `cnInitialize` ()
- Boolean `cnIsEndNode` (`ConstraintNode` n)
- Boolean `cnIsStartNode` (`ConstraintNode` n)
- Boolean `cnNodeComparePrio` (`ConstraintNode` a, `ConstraintNode` b)
- Boolean `cnNodeCompareSmallest` (`ConstraintNode` a, `ConstraintNode` b)
- int `cnOptimizeNet` (`ConstraintNet` net)
- int `cnOptimizeNode` (`ConstraintNet` net, `ConstraintNode` node)
- void `cnPrint` (long unsigned int mode, `ConstraintNet` net)
- void `cnPrintActiveLVs` (`ConstraintNet` net)
- void `cnPrintEdge` (long unsigned int mode, `ConstraintEdge` e)
- void `cnPrintInfo` (`ConstraintNet` net)
- void `cnPrintNode` (long unsigned int mode, `ConstraintNode` cn)
- void `cnPrintParses` (`ConstraintNet` net)
- Boolean `cnRenew` (`ConstraintNet` net)
- void `cnSortLVs` (`ConstraintNet` net)
- void `cnSortNodes` (`ConstraintNet` net)
- Boolean `cnTag` (`ConstraintNet` net, `Lattice` lat)
- void `cnUnaryPruning` (`ConstraintNode` node)
- Boolean `cnUnaryPruningCompare` (`LevelValue` a, `LevelValue` b)
- void `cnUndeleteAllLVs` (`ConstraintNet` net)
- int `countValidValues` (`ConstraintNet` net)
- void `cvAnalyse` (`ConstraintViolation` cv, `Vector` context)
- `ConstraintViolation` `cvClone` (`ConstraintViolation` cv)
- Boolean `cvCompare` (`ConstraintViolation` a, `ConstraintViolation` b)
- Boolean `cvCompareNatural` (`ConstraintViolation` a, `ConstraintViolation` b)
- Boolean `cvContains` (`List` conflicts, `ConstraintViolation` cv)
- void `cvDelete` (`ConstraintViolation` cv)
- `ConstraintViolation` `cvNew` (`Constraint` c, `LevelValue` lva, `LevelValue` lvb)
- void `cvPrint` (unsigned long mode, `ConstraintViolation` cv)

Variables

- int `cnCounter`
- int `cnCounter` = 0
- `CnEdgesType` `cnEdgesFlag`
- `CnEdgesType` `cnEdgesFlag` = `cnEdgesOff`
- `ConstraintNet` `cnMostRecentlyCreatedNet`
- `ConstraintNet` `cnMostRecentlyCreatedNet` = `NULL`
- Boolean `cnShowDeletedFlag`
- Boolean `cnShowDeletedFlag` = `FALSE`
- int `cnSortNodesMethod`
- int `cnSortNodesMethod` = 0
- Number `cnUnaryPruningFraction`
- Number `cnUnaryPruningFraction` = 1.0
- Boolean `cnUseNonSpec`
- Boolean `cnUseNonSpec` = `FALSE`

5.6.2 Typedef Documentation

5.6.2.1 typedef [ConstraintEdgeStruct](#)* [ConstraintEdge](#)

Pointer to the [ConstraintEdgeStruct](#) Definition at line 125 of file constraintnet.h.

Referenced by [cmdEdges\(\)](#), [cnBuildEdges\(\)](#), [cnDelete\(\)](#), [cnDeleteEdge\(\)](#), and [cnPrintEdge\(\)](#).

5.6.2.2 typedef struct [ConstraintEdgeStruct](#) [ConstraintEdgeStruct](#)

Models an edge between the two constraint nodes **start** and **stop**. These two fields are shallow copies of the nodes in the corresponding constraint net.

5.6.2.3 typedef [ConstraintNetStruct](#)* [ConstraintNet](#)

Pointer to the [ConstraintNetStruct](#) Definition at line 74 of file constraintnet.h.

Referenced by [cdgDeleteComputed\(\)](#), [cmdDistance\(\)](#), [cmdEdges\(\)](#), [cmdISearch\(\)](#), [cmdNet\(\)](#), [cmdNetdelete\(\)](#), [cmdNetsearch\(\)](#), [cmdNewnet\(\)](#), [cmdPrintParses\(\)](#), [cmdRenewnet\(\)](#), [cmdWriteAnno\(\)](#), [cmdWriteNet\(\)](#), [cmdWriteParses\(\)](#), [cnBuild\(\)](#), [cnBuildEdges\(\)](#), [cnBuildFinal\(\)](#), [cnBuildInit\(\)](#), [cnBuildIter\(\)](#), [cnBuildLevelValues\(\)](#), [cnBuildLv\(\)](#), [cnBuildNodes\(\)](#), [cnBuildTriple\(\)](#), [cnBuildUpdateArcs\(\)](#), [cnDelete\(\)](#), [cnDeleteAllLVs\(\)](#), [cnFindNet\(\)](#), [cnFindNode\(\)](#), [cnGetGraphemNodeFromArc\(\)](#), [cnGetLattice\(\)](#), [cnOptimizeNet\(\)](#), [cnOptimizeNode\(\)](#), [cnPrint\(\)](#), [cnPrintActiveLVs\(\)](#), [cnPrintInfo\(\)](#), [cnPrintParses\(\)](#), [cnRenew\(\)](#), [cnSortLVs\(\)](#), [cnSortNodes\(\)](#), [cnTag\(\)](#), [cnUndeleteAllLVs\(\)](#), [comApprove\(\)](#), [comCompareAllLvPairs\(\)](#), [comCompareAllLVs\(\)](#), [comCompareNets\(\)](#), [comCompareWithContext\(\)](#), [comFindComparableLv\(\)](#), [countValidValues\(\)](#), [evalBinary\(\)](#), [evalBinaryConstraint\(\)](#), [evalConstraint\(\)](#), [evalUnary\(\)](#), and [evalUnaryConstraint\(\)](#).

5.6.2.4 typedef [ConstraintNodeStruct](#)* [ConstraintNode](#)

Pointer to the [ConstraintNodeStruct](#) Definition at line 99 of file constraintnet.h.

Referenced by [cmdEdges\(\)](#), [cnBuildEdges\(\)](#), [cnBuildFinal\(\)](#), [cnBuildIter\(\)](#), [cnBuildLevelValues\(\)](#), [cnBuildLv\(\)](#), [cnBuildTriple\(\)](#), [cnBuildUpdateArcs\(\)](#), [cnConnectedByArc\(\)](#), [cnDelete\(\)](#), [cnDeleteNode\(\)](#), [cnFindNode\(\)](#), [cnIsEndNode\(\)](#), [cnIsStartNode\(\)](#), [cnNodeComparePrio\(\)](#), [cnNodeCompareSmallest\(\)](#), [cnOptimizeNet\(\)](#), [cnOptimizeNode\(\)](#), [cnPrint\(\)](#), [cnPrintActiveLVs\(\)](#), [cnPrintInfo\(\)](#), [cnPrintNode\(\)](#), [cnRenew\(\)](#), [cnSortLVs\(\)](#), [cnSortNodes\(\)](#), [cnUnaryPruning\(\)](#), and [comCompareNets\(\)](#).

5.6.2.5 typedef [ConstraintViolationStruct](#)* [ConstraintViolation](#)

Pointer to the [ConstraintViolationStruct](#) Definition at line 165 of file constraintnet.h.

Referenced by [cnCompareViolation\(\)](#), [comCompareAllLvPairs\(\)](#), [comCompareWithContext\(\)](#), [cvAnalyse\(\)](#), [cvClone\(\)](#), [cvCompare\(\)](#), [cvCompareNatural\(\)](#), [cvContains\(\)](#), [cvDelete\(\)](#), [cvNew\(\)](#), and [cvPrint\(\)](#).

5.6.2.6 typedef [NodeBindingStruct](#)* [NodeBinding](#)

Pointer to the [NodeBindingStruct](#) Definition at line 138 of file constraintnet.h.

Referenced by [cnDeleteBinding\(\)](#).

5.6.3 Enumeration Type Documentation

5.6.3.1 enum `CnEdgesType`

The types of a constraintnet edge Definition at line 33 of file constraintnet.h.

Referenced by `cmdIncrementalCompletion()`.

5.6.4 Function Documentation

5.6.4.1 `ConstraintNet` `cnBuild` (*Lattice* `lat`, *Boolean* `buildLVs`)

Build a constraint net from LAT.

This function returns a new constraint net for a word graph. This function uses `cnBuildInit()`, `cnBuildNodes()`, and `cnBuildFinal()`. The resulting net contains no edges. The function performs these initializations:

- `id` is set to `net<cnCounter>`
- the `lexemgraph` is built
- `state` is set to `NSCreated`

If `BUILDLVS` is set, the nodes in the constraint net will be filled with LVs, otherwise the net will contain only constraint nodes, no LVs. Definition at line 250 of file constraintnet.c.

References `CDG_WARNING`, `cdgCtrlCTrapped`, `cdgPrintf()`, `cnBuildFinal()`, `cnBuildInit()`, `cnBuildNodes()`, `cnDelete()`, `cnMostRecentlyCreatedNet`, `cnTag()`, `ConstraintNet`, and `NULL`.

Referenced by `cmdNewnet()`, and `comApprove()`.

5.6.4.2 `void` `cnBuildEdges` (*ConstraintNet* `net`)

builds edges in a constraint net

This function computes all edges in a constraint net:

- The Vector `edges` is allocated.
- A constraint edge is built for every pair of constraint nodes and inserted into `edges`.
- The fields `isMarked` and `scores` are allocated for each edge.
- All pairs of LVs from the two constraint nodes are evaluated jointly and the result is store in the matrix `scores`. The result will be ~ 0 if the LVs are no `lvCompatible()`, otherwise it is the product of the combined binary score and the two unary scores.
- A mapping from all edges to their reverses is computed and stored in the field `reverse`.

`C-c` interrupts this function, deletes all partial results, and displays the total time elapsed as a `CDG_PROFILE` message. Definition at line 1328 of file constraintnet.c.

References `CDG_DEBUG`, `CDG_PROFILE`, `cdgCtrlCTrapped`, `cdgFlush()`, `cdgPrintf()`, `cdgXCDG`, `cnConnectedByArc()`, `cnEdgesFlag`, `ConstraintEdge`, `ConstraintNet`, `ConstraintNode`, `ConstraintNetStruct::edges`, `evalBinary()`, `FALSE`, `ConstraintEdgeStruct::isMarked`, `ConstraintNodeStruct::level`,

ConstraintNetStruct::lexemgraph, ConstraintNetStruct::nodes, LexemGraphStruct::nodes, NULL, ConstraintEdgeStruct::reverse, ConstraintEdgeStruct::scores, smDelete(), smNew(), smSetFlag(), smSetScore(), ConstraintEdgeStruct::start, ConstraintEdgeStruct::stop, Timer, timerElapsed(), timerFree(), timerNew(), TRUE, and ConstraintNodeStruct::values.

Referenced by cnBuildFinal().

5.6.4.3 ConstraintNet cnBuildFinal (ConstraintNet net, Boolean buildLVs)

Final improvements: optimize, sort, unary pruning, score cache and edges

This function performs various clean-up operations on a net for which all constraint nodes have been built:

- It removes unnecessary structures from the net using [cnOptimizeNet\(\)](#).
- If **cnSortNodes** is set, it sorts the constraint nodes using [cnSortNodes\(\)](#).
- It applies [cnUnaryPruning\(\)](#) to each constraint node.
- If **scUseCache** is set, it initializes **net->cache** to a new cache returned by [scNew\(\)](#).
- If **cnEdgesFlag** is not **cnEdgesOff**, it applies [cnBuildEdges\(\)](#) to build the edges of the constraint net.

The function returns **FALSE** iff any of the subsidiary functions returned **FALSE**. Definition at line 163 of file constraintnet.c.

References ConstraintNetStruct::cache, CDG_WARNING, cdgCtrlCAllowed, cdgCtrlCTrapped, cdgPrintf(), cnBuildEdges(), cnDelete(), cnEdgesFlag, cnOptimizeNet(), cnSortLVs(), cnSortNodes(), cnSortNodesMethod, cnUnaryPruning(), cnUseNonSpec, ConstraintNet, ConstraintNode, ConstraintNetStruct::edges, FALSE, ConstraintNetStruct::isBuilt, ConstraintNetStruct::lexemgraph, LexemGraphStruct::max, max, LexemGraphStruct::min, ConstraintNetStruct::nodes, NULL, scNew(), scUseCache, and ConstraintNetStruct::values.

Referenced by cnBuild().

5.6.4.4 ConstraintNet cnBuildInit ()

This function returns a new ConstraintNet with a unique name, but with all fields set to meaningless default values. Building a complete constraint net is a multi-stage process that involves several of the other functions of this module. Definition at line 85 of file constraintnet.c.

References ConstraintNetStruct::cache, cnCounter, ConstraintNet, ConstraintNetStruct::edges, ConstraintNetStruct::evalBinary, ConstraintNetStruct::evalUnary, ConstraintNetStruct::id, ConstraintNetStruct::isBuilt, ConstraintNetStruct::lvTotals, ConstraintNetStruct::nodes, NULL, ConstraintNetStruct::parses, ConstraintNetStruct::searchagenda, ConstraintNetStruct::statUnary, ConstraintNetStruct::totalNumberOfValues, and ConstraintNetStruct::values.

Referenced by cnBuild().

5.6.4.5 Boolean cnBuildIter (ConstraintNet net, GraphemNode gn, Boolean buildLVs)

Builds the constraint node corresponding to GN in NET.

This function really performs most of the work that was documented under [cnBuildNodes\(\)](#) for simplicity. Definition at line 781 of file constraintnet.c.

References `cdgCtrlCTrapped`, `cdgFlush()`, `cnBuildLevelValues()`, `cnDeleteNode()`, `cnUseNonSpec`, `ConstraintNet`, `ConstraintNode`, `FALSE`, `ConstraintNodeStruct::gn`, `GraphemNode`, `LexemGraphStruct::graphemnodes`, `ConstraintNodeStruct::level`, `GraphemNodeStruct::lexemes`, `ConstraintNetStruct::lexemgraph`, `lgAreDeletedNodes()`, `ConstraintNodeStruct::net`, `ConstraintNetStruct::nodes`, `LexemGraphStruct::nodes`, `ConstraintNodeStruct::noValidValues`, `NULL`, `ConstraintNodeStruct::totalNumberOfValues`, `TRUE`, and `ConstraintNodeStruct::values`.

Referenced by `cnBuildNodes()`, and `cnBuildUpdateArcs()`.

5.6.4.6 void `cnBuildLevelValues` (`ConstraintNode node`, `Level level`, `GraphemNode modifier`, `GraphemNode modifiee`)

Builds all level values for `LEVEL`, `MODIFIER` and `MODIFIEE`.

This function builds all LVs that represent tuples composed from the parameters it receives. This function is used as the innermost loop by `cnBuildIter()`. Definition at line 652 of file `constraintnet.c`.

References `GraphemNodeStruct::arc`, `cnBuildLv()`, `ConstraintNet`, `ConstraintNode`, `GraphemNode`, `ConstraintNetStruct::lexemgraph`, `lgMayModify()`, `lgPartitions()`, `ConstraintNodeStruct::net`, and `NULL`.

Referenced by `cnBuildIter()`, `cnBuildTriple()`, and `cnBuildUpdateArcs()`.

5.6.4.7 void `cnBuildLv` (`ConstraintNode node`, `List modifiers`, `Level level`, `String label`, `List modifiees`)

Build a new levelvalue in `NODE`.

This function creates exactly one LV with the specified fields using `lvNew()` and stores it in `node`, incrementing all relevant counters properly. Definition at line 613 of file `constraintnet.c`.

References `CDG_DEBUG`, `cdgPrintf()`, `ConstraintNet`, `ConstraintNode`, `evalUnary()`, `FALSE`, `ConstraintNodeStruct::net`, `NULL`, `ConstraintNodeStruct::totalNumberOfValues`, `ConstraintNetStruct::totalNumberOfValues`, `ConstraintNetStruct::values`, and `ConstraintNodeStruct::values`.

Referenced by `cnBuildLevelValues()`.

5.6.4.8 Boolean `cnBuildNodes` (`ConstraintNet net`, `Boolean buildLVs`)

Builds the constraint nodes of `NET`. If `BUILDLVS` is set, the nodes are immediately filled with all possible LVs, otherwise they remain empty.

Returns `FALSE` if the `constraintnet` is invalid.

Basically it performs the following steps:

- allocates the Vectors `nodes` and `values`
- partitions the set of lexeme nodes created from each word hypothesis according to each level by using `lgPartitions()`
- allocates a `ConstraintNode` for each of the partitions k and inserts it into the Vector `nodes`
- checks whether the subordination (k, l, m) is possible for each triples of modifier set, label and modifiee set, and constructs the corresponding LV
- checks whether the subordination (k, l, NIL) is possible for each pair of modifier set and label, and constructs the corresponding LV

- inserts all new LVs into the respective constraint nodes and the Vector **values**
- sorts the LVs in each constraint node by their limit.

However, some complications apply:

- Under **CDG_DEBUG**, each word in the word graph is printed as it is used to build LVs.
- If a level has its **useflag** reset, it is ignored completely.
- Some of the LVs created may be destroyed again by **cnUnaryPruning()** if the variable **cnUnaryPruningFraction** is smaller than ~ 1 . Such LVs are not stored in the constraint node nor in the constraint net.
- Each iteration executes the hook **HOOK_CNBUILDNODES**.
- If **cnSortNodes** is set, **cnSortNodes()** is called after building all nodes.
- If **scUseCache** is set, a new cache is allocated for the constraint net.
- The time elapsed is printed as a **CDG_PROFILE** message.
- This function is interruptible by C-c in much the same way as **cnBuildEdges()**.

Definition at line 895 of file constraintnet.c.

References **CDG_DEBUG**, **CDG_HOOK**, **CDG_PROFILE**, **cdgExecHook()**, **cdgPrintf()**, **cnBuildIter()**, **ConstraintNet**, **GraphemNode**, **LexemGraphStruct::graphemnodes**, **hkVerbosity**, **HOOK_CNBUILDNODES**, **ConstraintNetStruct::isBuilt**, **GraphemNodeStruct::lexemes**, **ConstraintNetStruct::lexemgraph**, **LexemNode**, **LexemGraphStruct::max**, **ConstraintNetStruct::nodes**, **LexemGraphStruct::nodes**, **NULL**, **Timer**, **timerElapsed()**, **timerFree()**, **timerNew()**, **TRUE**, and **ConstraintNetStruct::values**.

Referenced by **cnBuild()**.

5.6.4.9 Boolean **cnBuildTriple** (**ConstraintNet** *net*, **int** *a*, **int** *b*, **int** *levelno*)

Build all LVs on LEVEL for the pair of time points (A,B) in NET.

All LVs that connect a word that starts at time point A as a modifier to a word that starts at time point B are built and inserted into the constraint net at the appropriate places. B == -1 means NIL, B == -2 means NONSPEC.

Returns FALSE if such LVs have already been built.

This function has nothing to do with incremental processing. It is intended for use by parsing algorithms that do not want the entire set of LVs built before they can start working, although the entire input is known. Definition at line 718 of file constraintnet.c.

References **GraphemNodeStruct::arc**, **cnBuildLevelValues()**, **ConstraintNet**, **ConstraintNode**, **FALSE**, **ConstraintNodeStruct::gn**, **GraphemNode**, **LexemGraphStruct::graphemnodes**, **ConstraintNetStruct::isBuilt**, **ConstraintNodeStruct::level**, **ConstraintNetStruct::lexemgraph**, **LexemGraphStruct::max**, **ConstraintNetStruct::nodes**, **ConstraintNodeStruct::noValidValues**, **NULL**, **ConstraintNodeStruct::totalNumberOfValues**, **TRUE**, and **ConstraintNodeStruct::values**.

5.6.4.10 Boolean `cnBuildUpdateArcs` ([ConstraintNet](#) *net*, [List](#) *listArcs*)

Update NET with the arcs in LISTARCS.

This function is used by **incrementalcompletion** to extend an existing constraint net by the structures corresponding to the specified **Arc** structures. Definition at line 2100 of file `constraintnet.c`.

References `CDG_DEBUG`, `cdgPrintf()`, `cnBuildIter()`, `cnBuildLevelValues()`, `cnGetGraphemNodeFromArc()`, `cnPrintNode()`, `ConstraintNet`, `ConstraintNode`, `FALSE`, `ConstraintNodeStruct::gn`, `GraphemNode`, `ConstraintNodeStruct::level`, `GraphemNodeStruct::lexemes`, `ConstraintNetStruct::nodes`, `ConstraintNodeStruct::noValidValues`, `NULL`, `ConstraintNodeStruct::totalNumberOfValues`, `TRUE`, and `ConstraintNodeStruct::values`.

5.6.4.11 void `cnCallback` ([String](#) *name*, [float](#) * *var*)

Callback function for `cnUnaryPruningFraction`.

This function is used as the callback function for the CDG variable **unaryFraction**. It prints a notification of the change to the CDG shell. Definition at line 2171 of file `constraintnet.c`.

References `CDG_INFO`, and `cdgPrintf()`.

Referenced by `cnInitialize()`.

5.6.4.12 Boolean `cnCompareViolation` ([ConstraintViolation](#) *a*, [ConstraintViolation](#) *b*)

compares two violations by penalty, then by domain index This function compares two structures of type **ConstraintViolation**, comparing first the penalty of the violation, after that the fields **nodeBindingIndex1** and **nodeBindingIndex2** and finally the names of the constraints.

TODO: this function is never used, see `cvCompare` for a similar function Definition at line 1830 of file `constraintnet.c`.

References `ConstraintViolationStruct::constraint`, `ConstraintViolation`, `FALSE`, `ConstraintViolationStruct::nodeBindingIndex1`, `ConstraintViolationStruct::nodeBindingIndex2`, `ConstraintViolationStruct::penalty`, and `TRUE`.

5.6.4.13 Boolean `cnConnectedByArc` ([ConstraintNode](#) *a*, [ConstraintNode](#) *b*)

Are two constraint nodes connected by an arc? This function checks whether two constraint nodes should be connected by an edge or not. This check is always performed before allocating a constraint edge. Two nodes fail this test if they bind the same lexeme and belong to the same level.

Think it over which nodes are connected by an arc. Do we need arcs in both directions?

`return ((a->lexemnode == b->lexemnode && a->level != b->level) || (a->level == b->level && lgDistanceOfNodes (a->lexemnode->lexemgraph, a->lexemnode, b->lexemnode) != 0));` Definition at line 1216 of file `constraintnet.c`.

References `ConstraintNode`, `ConstraintNodeStruct::gn`, and `ConstraintNodeStruct::level`.

Referenced by `cnBuildEdges()`.

5.6.4.14 void `cnDelete` ([ConstraintNet](#) *net*)

deletes constraint net

This function deallocates an entire constraint net. Note that while **cnBuild()** only allocates a few fields, this function allocates everything. The following functions are used for the purpose:

- **lgDelete()** for the **lexemgraph**
- **cnDeleteNode()** for the elements of **nodes**
- **cnDeleteEdge()** for the elements of **edges**
- **agDelete()** for the **searchagenda**
- **deleteParse()** for the elements of **parses**
- **scDelete()** for the **cache**

If **net** is identical to **cnMostRecentlyCreatedNet**, that variable is reset. Definition at line 1640 of file `constraintnet.c`.

References `ConstraintNetStruct::cache`, `CDG_WARNING`, `cdgFreeString()`, `cdgPrintf()`, `cnDeleteEdge()`, `cnDeleteNode()`, `cnMostRecentlyCreatedNet`, `ConstraintEdge`, `ConstraintNet`, `ConstraintNode`, `ConstraintNetStruct::edges`, `ConstraintNetStruct::id`, `ConstraintNetStruct::lexemgraph`, `lgDelete()`, `ConstraintNetStruct::lvTotals`, `ConstraintNetStruct::nodes`, `NULL`, `ConstraintNetStruct::parses`, `scDelete()`, `ConstraintNetStruct::searchagenda`, and `ConstraintNetStruct::values`.

Referenced by `cdgDeleteComputed()`, `cmdNetdelete()`, `cnBuild()`, `cnBuildFinal()`, `cnTag()`, and `comApprove()`.

5.6.4.15 void **cnDeleteAllLVs** (**ConstraintNet** *net*)

delete all levelvalues in a constraint net

This function sets the field **isDeleted** for all LVs in the specified constraint net. Definition at line 2083 of file `constraintnet.c`.

References `ConstraintNet`, `TRUE`, and `ConstraintNetStruct::values`.

5.6.4.16 void **cnDeleteBinding** (**NodeBinding** *nb*)

`cnDeleteBinding`

This function deallocates **NodeBinding** structure. Both components of the pair are shallow copies and cannot be deallocated. Definition at line 1484 of file `constraintnet.c`.

References `NodeBinding`.

5.6.4.17 void **cnDeleteEdge** (**ConstraintEdge** *cn*)

deletes a constraint-edge

This function deallocates a constraint edge. Apart from the **ConstraintEdgeStruct** itself, only the field **scores** is deallocated by calling **smDelete()**. Definition at line 1615 of file `constraintnet.c`.

References `ConstraintEdge`, `ConstraintEdgeStruct::isMarked`, `ConstraintEdgeStruct::scores`, and `smDelete()`.

Referenced by `cnDelete()`.

5.6.4.18 void `cnDeleteNode` (`ConstraintNode cn`)

deletes a constraint-node

This function deallocates a single constraint node and all LVs contained in it. Note that pointers to these LVs may remain in the field **values** of the enclosing constraint net, so `cnDeleteNode()` should only be called immediately before deleting the net itself. Definition at line 1593 of file `constraintnet.c`.

References `ConstraintNode`, and `ConstraintNodeStruct::values`.

Referenced by `cnBuildIter()`, and `cnDelete()`.

5.6.4.19 `ConstraintNet` `cnFindNet` (`String id`)

Looks for a constraint net with id ID and return it or NULL if it can't find the net in `inputCurrentGrammar`.

Returns:

the net with the specified **id** in the `inputCurrentGrammar` structure, or **NULL**.

Definition at line 2031 of file `constraintnet.c`.

References `cdgNets`, and `ConstraintNet`.

Referenced by `cmdDistance()`, `cmdPrintParses()`, `cmdRenewnet()`, and `cmdWriteParses()`.

5.6.4.20 `ConstraintNode` `cnFindNode` (`ConstraintNet net`, `LevelValue lv`)

Find `ConstraintNode` of an LV. This finds LVs by comparing their IWRT numbers, so cannot give meaningful results if `lv` was not originally built for net.

Returns:

the constraint node in **net** that holds **lv** (or **NULL**).

Definition at line 1881 of file `constraintnet.c`.

References `CDG_WARNING`, `cdgPrintf()`, `ConstraintNet`, `ConstraintNode`, `ConstraintNetStruct::nodes`, `NULL`, and `ConstraintNodeStruct::values`.

5.6.4.21 `GraphemNode` `cnGetGraphemNodeFromArc` (`ConstraintNet net`, `Arc arc`)

Return the graphem node in NET that points to ARC. Definition at line 2149 of file `constraintnet.c`.

References `GraphemNodeStruct::arc`, `ConstraintNet`, `GraphemNode`, `LexemGraphStruct::graphemnodes`, `ConstraintNetStruct::lexemgraph`, and `NULL`.

Referenced by `cnBuildUpdateArcs()`.

5.6.4.22 `Lattice` `cnGetLattice` (`ConstraintNet cn`)

Needed by XCDG because it can't talk to `LexemGraphs` directly.

This function simply returns **cn->lexemgraph->lattice**. It exists because XCDG needs that field, and XCDG cannot talk directly to lexeme graphs (they contain arrays of long long ints, which SWIG doesn't handle correctly). Definition at line 2204 of file `constraintnet.c`.

References `ConstraintNet`, `LexemGraphStruct::lattice`, and `ConstraintNetStruct::lexemgraph`.

5.6.4.23 void cnInitialize ()

Initialize the module constraintnet.

This function initializes the module [Constraintnet - maintenance of constraint nets](#). It merely registers the module's CDG variables. Definition at line 2185 of file constraintnet.c.

References [cnCallback\(\)](#), [cnEdgesFlag](#), [cnShowDeletedFlag](#), [cnSortNodesMethod](#), [cnUnaryPruning-Fraction](#), [cnUseNonSpec](#), and [NULL](#).

Referenced by [cdgInitialize\(\)](#).

5.6.4.24 Boolean cnIsEndNode ([ConstraintNode n](#))

Does this node bind the latest lexeme in the lattice? The functions [lgIsStartNode\(\)](#) and [lgIsEndNode\(\)](#) are used for this purpose. Definition at line 1471 of file constraintnet.c.

References [ConstraintNode](#), [ConstraintNodeStruct::gn](#), [ConstraintNodeStruct::level](#), and [lgIsEndNode\(\)](#).

5.6.4.25 Boolean cnIsStartNode ([ConstraintNode n](#))

Does this node bind the earliest lexeme in the lattice? The functions [lgIsStartNode\(\)](#) and [lgIsEndNode\(\)](#) are used for this purpose. Definition at line 1459 of file constraintnet.c.

References [ConstraintNode](#), [ConstraintNodeStruct::gn](#), [ConstraintNodeStruct::level](#), and [lgIsStartNode\(\)](#).

5.6.4.26 Boolean cnNodeComparePrio ([ConstraintNode a](#), [ConstraintNode b](#))

Compare constraint nodes by the `->no` of their levels. Definition at line 549 of file constraintnet.c.

References [ConstraintNode](#), and [ConstraintNodeStruct::level](#).

Referenced by [cnSortNodes\(\)](#).

5.6.4.27 Boolean cnNodeCompareSmallest ([ConstraintNode a](#), [ConstraintNode b](#))

Comparison function for [cnSortNodes\(\)](#) based on domain size.

This function is used by [cnSortNodes\(\)](#) to compare two constraint nodes according to the value of their `totalNumberOfValues=` fields. It returns `TRUE` iff `a` has a lower value or the same values as `b`. Definition at line 562 of file constraintnet.c.

References [ConstraintNode](#), and [ConstraintNodeStruct::noValidValues](#).

Referenced by [cnSortNodes\(\)](#).

5.6.4.28 int cnOptimizeNet ([ConstraintNet net](#))

optimize constraint net by deleting lexemes and values

This function deletes lexeme nodes and LVs from a constraint net that cannot possibly appear in any solution. It uses several rules for deleting structures:

- An LV can be deleted if it binds a set of lexeme nodes that are all deleted.
- An LV can be deleted if its modifier and modifiee are incompatible.

- A lexeme node can be deleted if it cannot be bound by any LV on one level.
- A lexeme can be deleted if there is no path through the lexeme graph in which it appears, and which is totally undeleted.

Since these conditions can trigger each other, the function `cnOptimizeNode()` is called repeatedly on each node in the constraint net until no progress has been made. The function returns the total number of changes returned by the calls to `cnOptimizeNode()`, or -1 if none of them did so.

Returns:

- > 0 : we have changed the constraintnet
- 0 : no changes have been made
- < 0 : the net is invalid

Definition at line 1168 of file constraintnet.c.

References `CDG_DEBUG`, `cdgPrintf()`, `cnOptimizeNode()`, `ConstraintNet`, `ConstraintNode`, `countValidValues()`, `FALSE`, `ConstraintNetStruct::nodes`, and `TRUE`.

Referenced by `cnBuildFinal()`, and `cnRenew()`.

5.6.4.29 int cnOptimizeNode (ConstraintNet net, ConstraintNode node)

Optimize constraint node by deleting lexemes and values.

Returns:

- > 0 : we have changed the constraintnode
- 0 : no changes have been made
- < 0 : the node is invalid

This function serves to discard structures in a constraint node that cannot appear in any solution. It returns the number of deletion operations that it has performed, or -1 if an inconsistency was found. This function is repeatedly called by `cnOptimizeNet()`. Definition at line 943 of file constraintnet.c.

References `GraphemNodeStruct::arc`, `CDG_DEBUG`, `CDG_WARNING`, `cdgPrintf()`, `ConstraintNet`, `ConstraintNode`, `FALSE`, `ConstraintNodeStruct::gn`, `GraphemNode`, `ConstraintNodeStruct::level`, `GraphemNodeStruct::lexemes`, `ConstraintNetStruct::lexemgraph`, `LexemNode`, `IgAreDeletableNodes()`, `IgAreDeletedNodes()`, `IgCompatibleNodes()`, `IgDeleteNode()`, `IgDeleteNodes()`, `IgIsDeletedNode()`, `IgPrintNode()`, `GraphemNodeStruct::no`, `LexemNodeStruct::no`, `LexemGraphStruct::nodes`, `LexemGraphStruct::noOfPathsFromStart`, `LexemGraphStruct::noOfPathsToEnd`, `ConstraintNodeStruct::noValidValues`, `NULL`, `TRUE`, and `ConstraintNodeStruct::values`.

Referenced by `cnOptimizeNet()`.

5.6.4.30 void cnPrint (long unsigned int mode, ConstraintNet net)

Print a constraint net.

This function displays a constraint net in textual form. Output is suppressed if `hkVerbosity` does not have the bit `mode` set. The function uses `cnPrintNode()` and `lvPrint()`. Constraint nodes from levels that have their `showflag` reset are skipped. Deleted LVs are only shown if `cnShowDeletedFlag` is set. Definition at line 291 of file constraintnet.c.

References `cdgCtrlCTrapped`, `cdgPrintf()`, `chunkerPrintChunks()`, `LexemGraphStruct::chunks`, `cnPrintNode()`, `cnShowDeletedFlag`, `ConstraintNet`, `ConstraintNode`, `ConstraintNetStruct::edges`, `ConstraintNodeStruct::gn`, `ConstraintNetStruct::id`, `LexemGraphStruct::isDeletedNode`, `ConstraintNodeStruct::level`,

GraphemNodeStruct::lexemes, ConstraintNetStruct::lexemgraph, lgAreDeletedNodes(), max, min, ConstraintNetStruct::nodes, LexemGraphStruct::noOfPaths, ConstraintNodeStruct::noValidValues, NULL, ConstraintNetStruct::parses, and ConstraintNodeStruct::values.

Referenced by cmdNet().

5.6.4.31 void cnPrintActiveLVs (ConstraintNet net)

prints the active levelvalues

This function applies **lvPrint()** to all LVs in the specified constraint net that are not deleted. Definition at line 2041 of file constraintnet.c.

References CDG_DEBUG, CDG_INFO, cdgPrintf(), ConstraintNet, ConstraintNode, LexemGraphStruct::isDeletedNode, ConstraintNetStruct::lexemgraph, ConstraintNetStruct::nodes, ConstraintNetStruct::values, and ConstraintNodeStruct::values.

5.6.4.32 void cnPrintEdge (long unsigned int mode, ConstraintEdge e)

Print a constraint edge.

This function displays a constraint edge and the associated matrix **scores**. It reacts to the display flag **mode** in the same way as **cnPrint()** and all other display functions. Definition at line 384 of file constraintnet.c.

References GraphemNodeStruct::arc, cdgPrintf(), ConstraintEdge, ConstraintNodeStruct::gn, ConstraintNodeStruct::level, NULL, ConstraintEdgeStruct::scores, smGetScore(), ConstraintEdgeStruct::start, ConstraintEdgeStruct::stop, and ConstraintNodeStruct::values.

Referenced by cmdEdges().

5.6.4.33 void cnPrintInfo (ConstraintNet net)

print out some information about this net

This function displays some general information about **net**. Apart from the fields of the structure themselves, the minimal, average, and maximal number of LVs per node is calculated and displayed. This function is called in various places to summarize briefly the state of a constraint net. Definition at line 1696 of file constraintnet.c.

References ConstraintNetStruct::cache, ScoreCacheStruct::capacity, CDG_INFO, CDG_WARNING, cdgPrintf(), ConstraintNet, ConstraintNode, ConstraintNetStruct::edges, ConstraintNetStruct::evalBinary, ConstraintNetStruct::evalUnary, ScoreCacheStruct::hits, hkVerbosity, ConstraintNetStruct::id, LexemGraphStruct::lattice, ConstraintNetStruct::lexemgraph, max, min, ConstraintNetStruct::nodes, ConstraintNodeStruct::noValidValues, NULL, ScoreCacheStruct::size, and ConstraintNetStruct::statUnary.

Referenced by cmdNetsearch(), cmdNewnet(), cmdPrintParses(), and cmdRenewnet().

5.6.4.34 void cnPrintNode (long unsigned int mode, ConstraintNode cn)

Print a constraint node.

This function displays a constraint node. A node of the level **LEVEL** binding the lexemes **das_1** and **das_2** from time span (*l,2*) would be displayed like this:

das_1/das_2(1-2)/LEVEL Definition at line 458 of file constraintnet.c.

References GraphemNodeStruct::arc, cdgPrintf(), ConstraintNode, ConstraintNodeStruct::gn, ConstraintNodeStruct::level, GraphemNodeStruct::lexemes, LexemNode, and NULL.

Referenced by cnBuildUpdateArcs(), cnPrint(), and cnSortNodes().

5.6.4.35 void cnPrintParses (ConstraintNet net)

prints all Parses of a net

This function applies **parsePrint()** to all entries of **net->parses**. Definition at line 1864 of file constraint-net.c.

References CDG_INFO, cdgPrintf(), ConstraintNet, NULL, and ConstraintNetStruct::parses.

Referenced by cmdNetsearch(), and cmdPrintParses().

5.6.4.36 Boolean cnRenew (ConstraintNet net)

Return a constraint net to untouched state

This function undoes all changes that have been made to a constraint net since it was built. In particular, it restores all deleted lexeme nodes and LVs to a constraint net. All parses of **net** are deallocated. This returns the net to the state that it was in immediately after being optimized. **TRUE** is returned only if at least one value remains for each pair of word and level. Definition at line 1247 of file constraintnet.c.

References ConstraintNetStruct::cache, cnOptimizeNet(), cnSortLVs(), cnUseNonSpec, ConstraintNet, ConstraintNode, FALSE, LexemGraphStruct::isDeletedNode, ConstraintNetStruct::lexemgraph, LexemNode, lgComputeDistances(), lgComputeNoOfPaths(), LexemNodeStruct::limit, LexemGraphStruct::nodes, ConstraintNetStruct::nodes, ConstraintNodeStruct::noValidValues, NULL, ConstraintNetStruct::parses, scDelete(), scNew(), scUseCache, TRUE, ConstraintNetStruct::values, and ConstraintNodeStruct::values.

Referenced by cmdRenewnet().

5.6.4.37 void cnSortLVs (ConstraintNet net)

re-sorts the LVs in each constraint node by limit rather than by score

This function sorts the Vectors **values=of** all constraint nodes of **net** using **lvCompare()**. Definition at line 1808 of file constraintnet.c.

References ConstraintNet, ConstraintNode, ConstraintNetStruct::nodes, and ConstraintNodeStruct::values.

Referenced by cnBuildFinal(), and cnRenew().

5.6.4.38 void cnSortNodes (ConstraintNet net)

Sort constraint nodes.

This function sorts the constraint nodes of **net** using the function **cnNodeCompare()**. Definition at line 573 of file constraintnet.c.

References CDG_DEBUG, cdgPrintf(), cnNodeComparePrio(), cnNodeCompareSmallest(), cnPrintNode(), cnSortNodesMethod, ConstraintNet, ConstraintNode, hkVerbosity, ConstraintNetStruct::nodes, and ConstraintNodeStruct::noValidValues.

Referenced by cnBuildFinal().

5.6.4.39 Boolean `cnTag` (`ConstraintNet net`, `Lattice lat`)

Build lexeme graph and add tagger and chunker information to NET. Definition at line 109 of file `constraintnet.c`.

References `CDG_DEBUG`, `CDG_INFO`, `cdgPrintf()`, `Chunker`, `chunkerChunk()`, `chunkerDelete()`, `chunkerNew()`, `chunkerPrintChunks()`, `cnDelete()`, `ConstraintNet`, `DefaultChunker`, `FALSE`, `LexemGraphStruct::graphemnodes`, `hkVerbosity`, `ConstraintNetStruct::lexemgraph`, `IgNew()`, `LexemGraphStruct::max`, `LexemGraphStruct::min`, `LexemGraphStruct::nodes`, `LexemGraphStruct::noOfPaths`, `NULL`, and `TRUE`.

Referenced by `cnBuild()`.

5.6.4.40 void `cnUnaryPruning` (`ConstraintNode node`)

Remove a given percentage of values from a node.

This function finds the values with the worst scores from a constraint node and deletes them. Let W be the set of LVs in `node`. All elements of W are deleted whose score is smaller than the score of the n th element, where

$$n = (\text{int}) = |W| * (1 - \text{cnUnaryPruningFraction})$$

This function temporarily sorts the set W using the function `cnUnaryPruningCompare()`. Definition at line 500 of file `constraintnet.c`.

References `CDG_INFO`, `cdgPrintf()`, `cnUnaryPruningCompare()`, `cnUnaryPruningFraction`, `ConstraintNode`, `ConstraintNodeStruct::noValidValues`, `NULL`, `TRUE`, and `ConstraintNodeStruct::values`.

Referenced by `cnBuildFinal()`.

5.6.4.41 Boolean `cnUnaryPruningCompare` (`LevelValue a`, `LevelValue b`)

comparison function for `cnUnaryPruning()`

This function compares two LVs by their scores. It returns `TRUE` iff $a->\text{score} < b->\text{score}$. Definition at line 482 of file `constraintnet.c`.

Referenced by `cnUnaryPruning()`.

5.6.4.42 void `cnUndeleteAllLVs` (`ConstraintNet net`)

undelete all levelvalues in a constraint net

This function resets the field `isDeleted` for all LVs in the specified constraint net. Definition at line 2066 of file `constraintnet.c`.

References `ConstraintNet`, `FALSE`, and `ConstraintNetStruct::values`.

5.6.4.43 int `countValidValues` (`ConstraintNet net`)

This function returns the number of undeleted LVs in `net`. It is a helper fo `cnOptimizeNet` Definition at line 1126 of file `constraintnet.c`.

References `ConstraintNet`, and `ConstraintNetStruct::values`.

Referenced by `cnOptimizeNet()`.

5.6.4.44 void cvAnalyse ([ConstraintViolation](#) *cv*, Vector *context*)

Show what the conflict means. This function prints out the conflict itself, the LVs that actually cause it, the constraint that is violated, and the participating lexemes. Definition at line 1494 of file constraintnet.c.

References CDG_INFO, cdgPrintf(), ConstraintViolationStruct::constraint, ConstraintViolation, cvPrint(), evalBinaryConstraint(), LexemNodeStruct::lexem, LexemNode, ConstraintViolationStruct::lv1, ConstraintViolationStruct::lv2, and NULL.

5.6.4.45 [ConstraintViolation](#) cvClone ([ConstraintViolation](#) *cv*)

clone a constraint violation

Returns:

a clone of *cv*.

Definition at line 1793 of file constraintnet.c.

References ConstraintViolationStruct::constraint, ConstraintViolation, cvNew(), ConstraintViolationStruct::lv1, ConstraintViolationStruct::lv2, and ConstraintViolationStruct::penalty.

5.6.4.46 Boolean cvCompare ([ConstraintViolation](#) *a*, [ConstraintViolation](#) *b*)

Constraint violations should be sorted first by penalty, then by arity, finally by natural order.

This function compares two conflicts. It returns **TRUE**

- if **a** has the lower penalty
- otherwise, if **a** is unary and **b** is binary
- otherwise, if the first LV in **a** precedes the first lv in **b** according to natural order.

Definition at line 1992 of file constraintnet.c.

References ConstraintViolationStruct::constraint, ConstraintViolation, FALSE, ConstraintViolationStruct::lv1, ConstraintViolationStruct::lv2, ConstraintViolationStruct::penalty, and TRUE.

Referenced by comCompareAllLvPairs(), and comCompareWithContext().

5.6.4.47 Boolean cvCompareNatural ([ConstraintViolation](#) *a*, [ConstraintViolation](#) *b*)

compares two violations by natural order

This function compares two **ConstraintViolation**= structures by the position of the affected LVs. If both affect the same LVs, it compares them by the constraint name. Definition at line 1960 of file constraintnet.c.

References ConstraintViolationStruct::constraint, ConstraintViolation, FALSE, ConstraintViolationStruct::nodeBindingIndex1, ConstraintViolationStruct::nodeBindingIndex2, and TRUE.

5.6.4.48 Boolean cvContains (List *conflicts*, [ConstraintViolation](#) *cv*)

Does a conflict occur in a list?

This function checks whether a conflict equivalent to **cv**= appears in the List. **TRUE**= is returned if both the constraint and the fields **nodeBindingIndex1**= and **nodeBindingIndex2**= match (the penalty need not match). Definition at line 1933 of file constraintnet.c.

References `ConstraintViolationStruct::constraint`, `ConstraintViolation`, `FALSE`, `ConstraintViolationStruct::nodeBindingIndex1`, `ConstraintViolationStruct::nodeBindingIndex2`, and `TRUE`.

5.6.4.49 void cvDelete ([ConstraintViolation](#) cv)

cvDelete

This function deallocates a structure of type **ConstraintViolation**. Since all components of this type are shallow copies, only the structure itself is deallocated. Definition at line 1567 of file `constraintnet.c`.

References `ConstraintViolation`, `ConstraintViolationStruct::lv1`, and `ConstraintViolationStruct::lv2`.

Referenced by `comCompareAllLvPairs()`, and `comCompareWithContext()`.

5.6.4.50 [ConstraintViolation](#) cvNew ([Constraint](#) c, [LevelValue](#) lva, [LevelValue](#) lvb)

Allocate a new constraint violation. This function allocates a new **ConstraintViolation**. The parameters are installed in the corresponding fields of the new structure. All fields of a **ConstraintViolation** are shallow copies. Definition at line 1756 of file `constraintnet.c`.

References `ConstraintViolationStruct::constraint`, `ConstraintViolation`, `ConstraintViolationStruct::lv1`, `ConstraintViolationStruct::lv2`, `ConstraintViolationStruct::nodeBindingIndex1`, `ConstraintViolationStruct::nodeBindingIndex2`, and `ConstraintViolationStruct::penalty`.

Referenced by `cvClone()`, `evalBinary()`, and `evalUnary()`.

5.6.4.51 void cvPrint (unsigned long mode, [ConstraintViolation](#) cv)

Print a conflict.

This function displays a conflict using `cdgPrintf()`. If `width` is ~ 0 , the constraint identifier is printed in full, otherwise it is truncated to `width` characters. Definition at line 1912 of file `constraintnet.c`.

References `CDG_INFO`, `cdgPrintf()`, `ConstraintViolationStruct::constraint`, `ConstraintViolation`, `ConstraintViolationStruct::nodeBindingIndex1`, `ConstraintViolationStruct::nodeBindingIndex2`, and `ConstraintViolationStruct::penalty`.

Referenced by `cvAnalyse()`.

5.6.5 Variable Documentation

5.6.5.1 int cnCounter

Counts the constraint nets created so far and appears in the name of every constraint net. Definition at line 50 of file `constraintnet.c`.

Referenced by `cnBuildInit()`.

5.6.5.2 int cnCounter = 0

Counts the constraint nets created so far and appears in the name of every constraint net. Definition at line 50 of file `constraintnet.c`.

Referenced by `cnBuildInit()`.

5.6.5.3 CnEdgesType cnEdgesFlag

This variable implements the CDG variable edges. Definition at line 58 of file constraintnet.c.

Referenced by cmdIncrementalCompletion(), cmdStatus(), cnBuildEdges(), cnBuildFinal(), and cnInitialize().

5.6.5.4 CnEdgesType cnEdgesFlag = cnEdgesOff

This variable implements the CDG variable edges. Definition at line 58 of file constraintnet.c.

Referenced by cmdIncrementalCompletion(), cmdStatus(), cnBuildEdges(), cnBuildFinal(), and cnInitialize().

5.6.5.5 ConstraintNet cnMostRecentlyCreatedNet

cnMostRecentlyCreatedNet points to last constraint net created (or to NULL). This variable is used as an implicit argument to commands that expect the name of a constraint net. Definition at line 55 of file constraintnet.c.

Referenced by cdgDeleteComputed(), cmdISearch(), cmdNet(), cmdNetdelete(), cmdNetsearch(), cmdPrintParses(), cmdRenewnet(), cmdWriteAnno(), cmdWriteNet(), cmdWriteParses(), cnBuild(), and cnDelete().

5.6.5.6 ConstraintNet cnMostRecentlyCreatedNet = NULL

cnMostRecentlyCreatedNet points to last constraint net created (or to NULL). This variable is used as an implicit argument to commands that expect the name of a constraint net. Definition at line 55 of file constraintnet.c.

Referenced by cdgDeleteComputed(), cmdISearch(), cmdNet(), cmdNetdelete(), cmdNetsearch(), cmdPrintParses(), cmdRenewnet(), cmdWriteAnno(), cmdWriteNet(), cmdWriteParses(), cnBuild(), and cnDelete().

5.6.5.7 Boolean cnShowDeletedFlag

If this flag is set, deleted LVs will be bracketed with [] in all output. If it is not set, they will not be shown at all. Definition at line 62 of file constraintnet.c.

Referenced by cmdStatus(), cnInitialize(), and cnPrint().

5.6.5.8 Boolean cnShowDeletedFlag = FALSE

If this flag is set, deleted LVs will be bracketed with [] in all output. If it is not set, they will not be shown at all. Definition at line 62 of file constraintnet.c.

Referenced by cmdStatus(), cnInitialize(), and cnPrint().

5.6.5.9 int cnSortNodesMethod

This variable implements the CDG variable sortnodes. Definition at line 76 of file constraintnet.c.

Referenced by cmdStatus(), cnBuildFinal(), cnInitialize(), and cnSortNodes().

5.6.5.10 int `cnSortNodesMethod` = 0

This variable implements the CDG variable `sortnodes`. Definition at line 76 of file `constraintnet.c`.

Referenced by `cmdStatus()`, `cnBuildFinal()`, `cnInitialize()`, and `cnSortNodes()`.

5.6.5.11 Number `cnUnaryPruningFraction`

This variable gives the approximate ratio of LVs retained while building a constraint net. It defaults to ~ 1 , so that no LVs are deleted upon building. If it is smaller, then from any set W of LVs the worst $\text{floor}(|W| \cdot x)$ LVs will be removed. Definition at line 73 of file `constraintnet.c`.

Referenced by `cmdStatus()`, `cnInitialize()`, and `cnUnaryPruning()`.

5.6.5.12 Number `cnUnaryPruningFraction` = 1.0

This variable gives the approximate ratio of LVs retained while building a constraint net. It defaults to ~ 1 , so that no LVs are deleted upon building. If it is smaller, then from any set W of LVs the worst $\text{floor}(|W| \cdot x)$ LVs will be removed. Definition at line 73 of file `constraintnet.c`.

Referenced by `cmdStatus()`, `cnInitialize()`, and `cnUnaryPruning()`.

5.6.5.13 Boolean `cnUseNonSpec`

if this Flag is set, newly created constraint networks will contain non-specific dependency edges aswell
Definition at line 66 of file `constraintnet.c`.

Referenced by `cnBuildFinal()`, `cnBuildIter()`, `cnInitialize()`, and `cnRenew()`.

5.6.5.14 Boolean `cnUseNonSpec` = FALSE

if this Flag is set, newly created constraint networks will contain non-specific dependency edges aswell
Definition at line 66 of file `constraintnet.c`.

Referenced by `cnBuildFinal()`, `cnBuildIter()`, `cnInitialize()`, and `cnRenew()`.

5.7 Eval - routines to evaluate constraint formulas

5.7.1 Detailed Description

Author:

Ingo Schroeder

Date:

7/3/97

This module provides functions to evaluate constraints (structures of type `Constraint`) on LVs (structures of type `LevelValue`). The result of evaluating a constraint is always a Boolean, but since constraint formulas are defined compositionally, there are also functions for evaluating terms of other types.

All evaluation functions use output parameters of type `Value` to compute and return their results. `ErrorValue` is returned if any error occurred during evaluation, such as an access to a feature of the root node. This typically does not cause a run-time error, but simply makes the containing formula evaluate to `FALSE`. Note that unpredictable results can be produced if an error occurs in a nested term. Therefore an evaluation error always produces a `CDG_WARNING` message.

Data Structures

- struct [BadnessStruct](#)

Static Strings

These strings are used so often that we want to pre-compute their addresses. This is the only location where it's really important to have full speed. They are initialized in [evalInitialize\(\)](#)

- String [static_string_chunk_end](#)
- String [static_string_chunk_start](#)
- String [static_string_chunk_type](#)
- String [static_string_from](#)
- String [static_string_id](#)
- String [static_string_info](#)
- String [static_string_to](#)
- String [static_string_word](#)

Typedefs

- typedef [BadnessStruct](#) * [Badness](#)

Enumerations

- enum [EvalMethodType](#) { [EMTInterpreted](#), [EMTCompiled](#) }

Functions

- [Badness bAdd](#) ([Badness b](#), [Number score](#))
- [Badness bAddBadness](#) ([Badness a](#), [Badness b](#))
- [Badness bClone](#) ([Badness b](#))
- Boolean [bCompare](#) ([Badness a](#), [Badness b](#))
- void [bCopy](#) ([Badness dest](#), [Badness src](#))
- void [bDelete](#) ([Badness b](#))
- Boolean [bEqual](#) ([Badness a](#), [Badness b](#))
- [Badness bestBadness](#) (void)
- [Badness bNew](#) (int no, int hard, [Number soft](#))
- void [bPrint](#) (unsigned long mode, [Badness b](#))
- [Badness bSubtract](#) ([Badness b](#), [Number score](#))
- [Badness bSubtractBadness](#) ([Badness a](#), [Badness b](#))
- [Number evalBinary](#) ([LevelValue lva](#), [LevelValue lvb](#), [ConstraintNet net](#), [Vector context](#), Boolean [use_cs_only](#), [Badness b](#), List *conflicts)
- Boolean [evalBinaryConstraint](#) ([Constraint c](#), [ConstraintNet net](#), [Vector context](#), [LevelValue lva](#), [LevelValue lvb](#))
- Boolean [evalConstraint](#) ([Constraint c](#), [ConstraintNet net](#), [Vector context](#),...)
- void [evalFinalize](#) (void)
- Boolean [evalFormula](#) ([Formula f](#), [LexemGraph lg](#), [Vector context](#))
- [Number evalInContext](#) ([Vector LVs](#), [Vector context](#), [Badness b](#), List *conflicts)
- void [evalInitialize](#) (void)
- [Value evalTerm](#) ([Term t](#), [Value val](#), [LexemGraph lg](#), [Vector context](#))
- [Number evalUnary](#) ([LevelValue lv](#), [ConstraintNet net](#), [Vector context](#), Boolean [use_cs_only](#), [Badness b](#), List *conflicts)
- Boolean [evalUnaryConstraint](#) ([Constraint c](#), [ConstraintNet net](#), [Vector context](#), [LevelValue lv](#))
- Boolean [evalValidateEvalMethod](#) ([String name](#), [String value](#), int *var)
- void [lock_tree](#) (int width)
- [Value peekValue](#) ([Value val](#), List path)
- Boolean [significantlyGreater](#) ([Number a](#), [Number b](#))
- void [unlock_tree](#) (void)
- [Badness worstBadness](#) (void)

Variables

- [Constraint evalCurrentConstraint](#)
- [Constraint evalCurrentConstraint](#) = NULL
- [Formula evalCurrentFormula](#)
- [Formula evalCurrentFormula](#) = NULL
- [EvalMethodType evalEvaluationMethod](#)
- [EvalMethodType evalEvaluationMethod](#) = EMTInterpreted
- [EvalMethodType evalPeekValueMethod](#)
- [EvalMethodType evalPeekValueMethod](#) = EMTCompiled
- Boolean [evalSloppySubsumesWarnings](#)
- Boolean [evalSloppySubsumesWarnings](#) = FALSE
- int * [has_cache](#)
- int * [has_cache](#) = NULL
- int [lock_counter](#)
- int [lock_counter](#) = 0

- int `lock_width`
- int `lock_width = 0`
- String `static_string_chunk_end`
- String `static_string_chunk_start`
- String `static_string_chunk_type`
- String `static_string_from`
- String `static_string_id`
- String `static_string_info`
- String `static_string_to`
- String `static_string_word`

5.7.2 Typedef Documentation

5.7.2.1 typedef `BadnessStruct*` `Badness`

A pointer to a `BadnessStruct` Definition at line 60 of file eval.h.

Referenced by `bAdd()`, `bAddBadness()`, `bClone()`, `bCompare()`, `bCopy()`, `bDelete()`, `bEqual()`, `bestBadness()`, `bNew()`, `bPrint()`, `bSubtract()`, `bSubtractBadness()`, `evalBinary()`, `evalInContext()`, `evalUnary()`, and `worstBadness()`.

5.7.3 Enumeration Type Documentation

5.7.3.1 enum `EvalMethodType`

Evaluation method

This is the Type of the evaluation method which could be:

- interpreted
- compiled

Definition at line 38 of file eval.h.

Referenced by `comApprove()`.

5.7.4 Function Documentation

5.7.4.1 `Badness` `bAdd` (`Badness b`, `Number score`)

Add score to a `Badness`.

This function adds a classical score to a generalized score, i.e. it either increments `b->hard` or multiplies `b->soft` by score. Definition at line 258 of file eval.c.

References `Badness`, `BadnessStruct::hard`, `BadnessStruct::no`, and `BadnessStruct::soft`.

Referenced by `evalBinary()`, and `evalUnary()`.

5.7.4.2 `Badness` `bAddBadness` (`Badness a`, `Badness b`)

Add `Badness b` to `a`. Definition at line 290 of file eval.c.

References `Badness`, `BadnessStruct::hard`, `BadnessStruct::no`, and `BadnessStruct::soft`.

5.7.4.3 **Badness** bClone (**Badness** *b*)

Create deep copy of a Badness.

The result is a new Badness owned by the caller. Definition at line 246 of file eval.c.

References Badness, bNew(), BadnessStruct::hard, BadnessStruct::no, and BadnessStruct::soft.

5.7.4.4 **Boolean** bCompare (**Badness** *a*, **Badness** *b*)

Compare two Badnesses.

Returns TRUE if *a* is properly better than *b*, FALSE if they are equal. Definition at line 348 of file eval.c.

References Badness, FALSE, BadnessStruct::hard, significantlyGreater(), BadnessStruct::soft, and TRUE.

Referenced by cmdWriteAnno().

5.7.4.5 **void** bCopy (**Badness** *dest*, **Badness** *src*)

Copy a Badness.

src must have been allocated by the caller. Definition at line 236 of file eval.c.

References Badness.

5.7.4.6 **void** bDelete (**Badness** *b*)

De-allocate a Badness. Definition at line 317 of file eval.c.

References Badness.

5.7.4.7 **Boolean** bEqual (**Badness** *a*, **Badness** *b*)

Test if two badnesses are equal. Definition at line 338 of file eval.c.

References Badness, BadnessStruct::hard, and BadnessStruct::soft.

5.7.4.8 **Badness** bestBadness (**void**)

Return new Badness better than any other.

The result is owned by the caller. Definition at line 370 of file eval.c.

References Badness, and bNew().

5.7.4.9 **Badness** bNew (**int** *no*, **int** *hard*, **Number** *soft*)

Allocate a Badness. Definition at line 219 of file eval.c.

References Badness, BadnessStruct::hard, BadnessStruct::no, and BadnessStruct::soft.

Referenced by bClone(), bestBadness(), and worstBadness().

5.7.4.10 void bPrint (unsigned long *mode*, **Badness** *b*)

Print a badness in canonical form. Definition at line 325 of file eval.c.

References **Badness**, `cdgPrintf()`, `BadnessStruct::hard`, `BadnessStruct::no`, and `BadnessStruct::soft`.

5.7.4.11 **Badness** bSubtract (**Badness** *b*, **Number** *score*)

Subtract score from a **Badness**. Definition at line 274 of file eval.c.

References **Badness**, `BadnessStruct::hard`, `BadnessStruct::no`, and `BadnessStruct::soft`.

5.7.4.12 **Badness** bSubtractBadness (**Badness** *a*, **Badness** *b*)

Subtract **Badness** *b* from *a*. Definition at line 302 of file eval.c.

References **Badness**, `BadnessStruct::hard`, `BadnessStruct::no`, and `BadnessStruct::soft`.

5.7.4.13 **Number** evalBinary (**LevelValue** *lva*, **LevelValue** *lvb*, **ConstraintNet** *net*, **Vector** *context*, **Boolean** *use_cs_only*, **Badness** *b*, **List** * *conflicts*)

Evaluate all constraints on two LVs.

This function performs a conceptually simple task: it computes the combined score of two LVs in a constraint net and returns the computed score. This is done by applying `evalConstraint()` to all known constraints and tallying the scores of all violated constraints.

For efficiency reasons, the actual implementation of this function is much more complicated:

- The entire function can be run in fast mode or in thorough mode. In fast mode, the function will stop applying constraints if a constraint with score 0 is violated, and return 0 immediately. In thorough mode, it will keep applying all possible constraints and `bAdd()` the results to the **Badness** *b*. Also, `ConstraintViolation` structures will be allocated and inserted into the **List** *conflicts*. Thorough mode is requested by passing a non-null value for *b*. For fast mode, both *b* and *conflicts* should be `NULL`.
- In fast mode, if the LVs *lva* and *lvb* have already been evaluated by this function and `scUseCache` was set during both invocations of `evalBinary()`, the result is read from `net->cache` and not computed again.
- To make this case more probable, the LVs *lva* and *lvb* are always sorted by the number of their respective levels. (That is, only half of the entire cache is ever used.)
- Even if no precomputed result exists, not all constraints may be evaluated:

Unary constraints are ignored completely.

Inactive constraints and constraints from inactive sections are skipped likewise.

Of the remaining constraints, only those that may actually fail are evaluated. Hence not all constraints are evaluated, but only those whose signature matches the actual configuration existing between *lva* and *lvb*. The **List** of these constraints is found in the matrix `cdgConstraintMatrix[]`.

In fast mode, if a hard constraint is violated, the result 0.0 is returned immediately.

If the parameter *context* is `NULL`, context-sensitive constraints will not be evaluated.

If the parameter *use_cs_only* is set, ONLY context-sensitive constraints will be evaluated. See `evalInContext()` for why this is occasionally necessary.

- Theoretically, a binary constraint may be violated twice by two LVs. Think of a constraint that forbids the label X to appear more than once in an analysis. If two LVs with label X are evaluated jointly, both (a,b) and (b,a) violate this condition. But since the two violations do not really indicate different errors, this kind of double fault is usually not desired.

A double fault can only occur if the LVs match the signature of a constraint in both variants of instantiation. Since every binary constraint signature must explicitly state the two levels it refers to, this is only possible if both LVs belong to the same level and either their configuration is symmetrical, or it is asymmetrical, but the signature of the constraint subsumes both configurations.

To eliminate all double faults, `evalBinary()` checks for all of these possibilities explicitly.

- If a value was computed rather than looked up, and `scUseCache` is set, it is registered in `net->cache`.

The parameter `net` can be NULL, so that LVs can be evaluated even in the absence of a constraint net. In this case no caching is possible. Definition at line 1395 of file `eval.c`.

References `bAdd()`, `Badness`, `ConstraintNetStruct::cache`, `CDG_DEBUG`, `cdgPrintf()`, `ConstraintNet`, `cvNew()`, `evalBinaryConstraint()`, `FALSE`, `lock_tree()`, `NULL`, `scGetScore()`, `scSetScore()`, `scUseCache`, `TRUE`, and `unlock_tree()`.

Referenced by `cnBuildEdges()`, `comCompareAllLvPairs()`, `comCompareWithContext()`, and `evalInContext()`.

5.7.4.14 Boolean `evalBinaryConstraint` (Constraint `c`, `ConstraintNet net`, Vector `context`, LevelValue `lva`, LevelValue `lwb`)

Evaluate one binary constraint on two LVs.

This function evaluates a binary constraint. It has the same effect as `evalConstraint()` used with five arguments, but saves the overhead of accessing the variable argument list. Also, it will call a compiled constraint function rather than `evalConstraint()` if possible. Definition at line 1188 of file `eval.c`.

References `CDG_HOOK`, `CDG_WARNING`, `cdgExecHook()`, `cdgPrintf()`, `ConstraintNet`, `ConstraintNetStruct::evalBinary`, `evalCurrentConstraint`, `evalEvaluationMethod`, `evalFormula()`, `evalTerm()`, `hkVerbosity`, `HOOK_EVAL`, `NULL`, and `TRUE`.

Referenced by `cvAnalyse()`, and `evalBinary()`.

5.7.4.15 Boolean `evalConstraint` (Constraint `c`, `ConstraintNet net`, Vector `context`, ...)

Evaluate a constraint w.r.t. the specified LVs.

Here's the detailed algorithm:

- check the number of arguments and returns TRUE (with a warning) if it does not match the arity of `c`
- instantiate the variables in `c->vars` with the LVs passed as arguments
- do *not* check whether `c` is applicable to these LVs. Applying constraints to arbitrary LVs can produce spurious constraint violations without warning! Hence `evalConstraint()` should only be called on constraints whose applicability was previously checked by `match{Unary,Binary}Signature()`.
- set `evalCurrentConstraint` and evaluates `c->formula` using `evalFormula()`
- reset `evalCurrentConstraint`

- if the result was FALSE and *c* has a variable penalty (if *c*->penaltyTerm is non-NULL), set *c*->penalty to evalTerm(*c*->penaltyTerm)
- execute the hook HOOK_EVAL
- de-instantiate *c*->vars
- increase either evalUnary or evalBinary
- return the result of evalFormula()

The net is actually only used to record statistics (the fields evalUnary and evalBinary) and can be NULL.

The context may be NULL. If it is not NULL it should be a Vector of LVs, each of which must be located at its index as calculated by lvIndex(). This is then passed on to predicates and functions that need to examine the context of an LV to operate.

If the constraint *c* uses such a function or predicate (that is, if it has its is_context_sensitive flag set) and there is no context, then evalConstraint() returns TRUE. This is obviously not always correct, so if you do not provide context to your evaluations and still use context-sensitive constraints, you can miss some conflicts.

This function can evaluate constraints of arbitrary arity. The number of LVs passed must equal the arity. Unary and binary constraints can also be evaluated by evalUnaryConstraint() and evalBinaryConstraint(). Those functions are slightly faster than using the general vararg mechanism. Definition at line 1040 of file eval.c.

References CDG_HOOK, CDG_WARNING, cdgExecHook(), cdgPrintf(), ConstraintNet, ConstraintNetStruct::evalBinary, evalCurrentConstraint, evalFormula(), evalTerm(), ConstraintNetStruct::evalUnary, FALSE, hkVerbosity, HOOK_EVAL, NULL, and TRUE.

5.7.4.16 void evalFinalize (void)

finalize the eval module Definition at line 1666 of file eval.c.

References cdgFreeString(), static_string_chunk_end, static_string_chunk_start, static_string_chunk_type, static_string_from, static_string_id, static_string_info, static_string_to, and static_string_word.

Referenced by cdgFinalize().

5.7.4.17 Boolean evalFormula (Formula *f*, LexemGraph *lg*, Vector *context*)

Evaluate a formula.

This function interprets the Boolean formulas defined in the cdg input language. It is basically a comprehensive listing of actions for each possible value of *f*->type.

Each of the different types of formula is evaluated in the expected way. In particular, conjunctions, implications and disjunction are guaranteed to short-circuit in the same way as the corresponding C operators.

Binary relations (FTEqual etc.) are evaluated by calling evalTerm() on both halves of the formula. If the two resulting types are not compatible, FALSE is returned with a warning. In general, every type is only compatible with itself. The ErrorValue is not compatible with any type, not even with itself.

The various relations are defined as follows:

- Two strings are equal if they are identical (since no strings are duplicated in cdg, this is equivalent to checking whether they have the same reading).

- Two numbers are equal if == holds between them.
- Two lexeme nodes are equal if they were created from the same grapheme node or they are both NULL. Note that this may judge two simultaneous lexeme nodes as equal even though they are acoustically incompatible. This case should not arise since no constraint should ever be evaluated on two incompatible LVs; however, `evalFormula()` does not check this.
- Two underspecified values or an underspecified and a specified value are never equal.
- FTGreater holds between two numbers that satisfy C's > operator.
- FTLess holds between two numbers that satisfy C's < operator.

If `f->type` is `FTConnexion`, `subsumesConnexion()` is called to check whether the specification `f->data.connexion.c` subsumes the actual configuration of the LVs found in the fields `f->data.connexion.var1->levelvalue` and `f->data.connexion.var2->levelvalue`.

If `f->type` is `FTDirection`, `subsumesDirection()` is called to check whether the specification `f->data.direction.d` subsumes the `Direction` of the `LevelValue` structure found in the field `f->data.direction.var->levelvalue`. Definition at line 434 of file `eval.c`.

References `CDG_WARNING`, `cdgPrintf()`, `evalCurrentFormula`, `evalTerm()`, `FALSE`, `LexemNode`, `NULL`, and `TRUE`.

Referenced by `evalBinaryConstraint()`, `evalConstraint()`, and `evalUnaryConstraint()`.

5.7.4.18 Number `evalInContext` (Vector LVs, Vector context, Badness b, List * conflicts)

Eval context-sensitive constraints only.

This function evaluates all LVs in the Vector LVs against each other, using only context-sensitive constraints, and assuming that the context is context.

This is useful for methods that cannot normally provide context, e.g. because they operate incrementally. Such methods should use normal `evalUnary` and `evalBinary` calls with no context while they are building a structure, and then this function once the structure is complete.

Note that LVs and context can be different; one frobbing method actually needs to eval a subset of a structure, but in the global context. Definition at line 1614 of file `eval.c`.

References `Badness`, `evalBinary()`, `evalUnary()`, `NULL`, and `TRUE`.

5.7.4.19 void `evalInitialize` (void)

Initialize the eval module Definition at line 1642 of file `eval.c`.

References `evalEvaluationMethod`, `evalPeekValueMethod`, `evalSloppySubsumesWarnings`, `evalValidateEvalMethod()`, `FALSE`, `NULL`, `static_string_chunk_end`, `static_string_chunk_start`, `static_string_chunk_type`, `static_string_from`, `static_string_id`, `static_string_info`, `static_string_to`, `static_string_word`, and `TRUE`.

Referenced by `cdgInitialize()`.

5.7.4.20 Value `evalTerm` (Term t, Value val, LexemGraph lg, Vector context)

evaluate a CDG Term.

Parameters:

- t* the term to be evaluated
- val* temporary storage where the result is stored in
- lg* the current lexemgraph
- context* the current parse tree (or NULL)

Returns:

- a pointer to *val* or the `ErrorValue`

This function evaluates a term on a value of the cdg formula language, computing both its type and its value. Both are returned as the result of `evalTerm()`. The function performs an exhaustive case distinction over the possible values of `t->type`:

- **TTTopPeek:** The field `t->data.peek.varinfo->levelvalue->modifiee` is checked. If the modifiee is underspecified, `ErrorValue` is returned, otherwise `peekValue()` is applied to the path `t->data.peek.path` in the value of the modifiee.

Exceptions occur if `t->data.peek.path` is `id`, `word`, `from` or `to`. These special attributes are notated like features of a lexeme node, but are actually attributes of the lexeme node, and not the lexeme itself. These attributes are computed as follows:

 - the attribute `id` is computed by setting `val->lexemnode` to the modifiee itself. If the modifiee is underspecified, `ErrorValue` is returned, `VTLexemNode` otherwise.
 - the attribute `word` is computed by setting `val->data.string` to the field `word` of the lexical entry of the modifier. (Strictly speaking, this *is* an attribute of the lexeme rather than the lexeme node; but as it does not appear in the feature matrix, it cannot be computed by `peekValue()`.) If the modifiee is underspecified, `ErrorValue` is returned, `val` otherwise.
 - the attribute `from` is computed by setting `val->data.number` to the field `from` of the modifiee. If the modifiee is underspecified, `lg->max+1` is assigned instead. In any case, `val` is returned.
 - the attribute `to` is computed by setting `val->data.number` to the field `to` of the modifiee. If the modifiee is underspecified, the number `lg->max+2` is assigned instead. In any case, `val` is returned.
- **TTBottomPeek:** The same algorithm is applied to the modifier of the `LevelValue` – simplified by the fact that modifiers cannot be underspecified.
- **TTLabel:** `val->data.string` is set to the field `label` of the `LV` and `val` is returned.
- **TTLevel:** `val->data.string` is set to the identifier of the `Level` of `LV`, and `val` is returned.
- **TTAdd, TTSubtract, TTMultiply, TTDivide:** The two fields `t->data.operation.op1` and `t->data.operation.op2` are evaluated by `evalTerm()`. `ErrorValue` is returned unless both results are numbers. Dividing by 0 also returns `ErrorValue`. Otherwise the results are conjoined by the corresponding C operator (+, -, * oder /), `val->data.number` is set to the result of this computation, and `VTNumber` is returned. applied to `t->data.function.args` (and `lg`) and both the resulting type and the computed are passed to the caller of `evalTerm()`.
- **TTString:** `val->data.string` is set to `t->data.string`, and `val` is returned.
- **TTNumber:** `val->data.number` is set to `t->data.number`, and `val` is returned.

Definition at line 710 of file `eval.c`.

References GraphemNodeStruct::arc, LexemNodeStruct::arc, CDG_WARNING, cdgPrintf(), GraphemNodeStruct::chunk, Chunk, chunkerStringOfChunkType(), evalPeekValueMethod, FALSE, ChunkStruct::from, LexemNodeStruct::grapheme, LexemNodeStruct::lexem, LexemNode, LexemGraphStruct::max, peekValue(), static_string_chunk_end, static_string_chunk_start, static_string_chunk_type, static_string_from, static_string_id, static_string_info, static_string_to, static_string_word, and ChunkStruct::to.

Referenced by evalBinaryConstraint(), evalConstraint(), evalFormula(), and evalUnaryConstraint().

5.7.4.21 Number evalUnary (LevelValue *lv*, ConstraintNet *net*, Vector *context*, Boolean *use_cs_only*, Badness *b*, List * *conflicts*)

Eval unary constraints on LV.

This function applies all unary constraints to *lv*. It uses the Vector *cdgConstraintVector* in much the same way as [evalBinary\(\)](#) uses *cdgConstraintMatrix*. The same exceptions for deactivated constraints apply as in that function. In addition, [evalUnary\(\)](#) stores all violated local constraints in the List *lv->constraints*. Definition at line 1259 of file *eval.c*.

References *bAdd()*, *Badness*, CDG_DEBUG, *cdgPrintf()*, *ConstraintNet*, *cvNew()*, *evalUnaryConstraint()*, and NULL.

Referenced by *cnBuildLv()*, and *evalInContext()*.

5.7.4.22 Boolean evalUnaryConstraint (Constraint *c*, ConstraintNet *net*, Vector *context*, LevelValue *lv*)

Evaluate a unary constraint on LV.

This function evaluates a unary constraint. It has the same effect as [evalConstraint\(\)](#) used with four arguments, but saves the overhead of accessing the variable argument list. Also, it will call a compiled constraint function rather than [evalConstraint\(\)](#) if possible. Definition at line 1122 of file *eval.c*.

References CDG_HOOK, CDG_WARNING, *cdgExecHook()*, *cdgPrintf()*, *ConstraintNet*, *evalCurrentConstraint*, *evalEvaluationMethod*, *evalFormula()*, *evalTerm()*, *ConstraintNetStruct::evalUnary*, *hkVerbosity*, HOOK_EVAL, NULL, and TRUE.

Referenced by *evalUnary()*.

5.7.4.23 Boolean evalValidateEvalMethod (String *name*, String *value*, int * *var*)

This is the validation function for the CDG variable *evalmethod*. Definition at line 1681 of file *eval.c*.

References CDG_WARNING, *cdgPrintf()*, FALSE, and TRUE.

Referenced by *evalInitialize()*.

5.7.4.24 void lock_tree (int *width*)

Declare evaluation lock.

When many constraints are evaluated on the same dependency tree, it can be worthwhile to cache the result of common subformulas. Therefore we offer the caller the possibility to declare that the dependency tree will not change during the next evaluations. The module *eval* is then free to reuse the results of previous evaluations until the tree is unlocked again.

WIDTH specifies the number of time points in the lexeme graph that will be used in the evaluations. Definition at line 157 of file eval.c.

References has_cache, lock_counter, and lock_width.

Referenced by evalBinary().

5.7.4.25 Value peekValue (Value val, List path)

returns value for a path in a feature matrix Definition at line 611 of file eval.c.

References CDG_WARNING, cdgPrintf(), and NULL.

Referenced by evalTerm().

5.7.4.26 Boolean significantlyGreater (Number a, Number b)

Compare Numbers with some margin for rounding error.

Upon removing conflicts and introducing new ones, the score of a structure must frequently be multiplied and divided with constants. Repeated use of this practice may introduce rounding errors. Therefore we use a special function for comparison with some margin for error. The problem of distinguishing rounding errors from very subtle penalties remains unsolved. Definition at line 206 of file eval.c.

Referenced by bCompare().

5.7.4.27 void unlock_tree (void)

Release evaluation lock.

If `lock_tree()` was called multiple times, evaluation remains locked onto the current tree until each call been cancelled individually. Definition at line 178 of file eval.c.

References CDG_ERROR, cdgPrintf(), has_cache, lock_counter, lock_width, and NULL.

Referenced by evalBinary().

5.7.4.28 Badness worstBadness (void)

Return new Badness worse than any other.

Strictly speaking, this is not the worst Badness that any structure could ever have, but I guess it will do.

The result is owned by the caller. Definition at line 383 of file eval.c.

References Badness, and bNew().

5.7.5 Variable Documentation

5.7.5.1 Constraint evalCurrentConstraint

This variable is set to the current Constraint during an evaluation, NULL otherwise. Definition at line 80 of file eval.c.

Referenced by evalBinaryConstraint(), evalConstraint(), and evalUnaryConstraint().

5.7.5.2 Constraint `evalCurrentConstraint` = NULL

This variable is set to the current Constraint during an evaluation, NULL otherwise. Definition at line 80 of file eval.c.

Referenced by `evalBinaryConstraint()`, `evalConstraint()`, and `evalUnaryConstraint()`.

5.7.5.3 Formula `evalCurrentFormula`

Set to the current Formula during an evaluation, NULL otherwise.

Actually this is not strictly true. During evaluation, it is set to the formula whose evaluation was most recently **begun**, which is not at all the same; if we evaluate the formula

```
X@cat = N & X^cat != V
```

then the first formula (&) will be evaluated first, then the second formula (=) and then the third one (!=), but after that, `evalCurrentFormula` is not reset to the first formula. To trace the progress of evaluation exactly, we would need either LISP-style dynamic binding, or an extra Formula argument across the entire chain of evaluation. But since `evalCurrentFormula` is only used within `predHas()` to distinguish different `has()` invocations from each other, and those do not nest, the current behaviour is sufficient. Definition at line 100 of file eval.c.

Referenced by `evalFormula()`.

5.7.5.4 Formula `evalCurrentFormula` = NULL

Set to the current Formula during an evaluation, NULL otherwise.

Actually this is not strictly true. During evaluation, it is set to the formula whose evaluation was most recently **begun**, which is not at all the same; if we evaluate the formula

```
X@cat = N & X^cat != V
```

then the first formula (&) will be evaluated first, then the second formula (=) and then the third one (!=), but after that, `evalCurrentFormula` is not reset to the first formula. To trace the progress of evaluation exactly, we would need either LISP-style dynamic binding, or an extra Formula argument across the entire chain of evaluation. But since `evalCurrentFormula` is only used within `predHas()` to distinguish different `has()` invocations from each other, and those do not nest, the current behaviour is sufficient. Definition at line 100 of file eval.c.

Referenced by `evalFormula()`.

5.7.5.5 EvalMethodType `evalEvaluationMethod`

Implements `evalmethod`. Definition at line 116 of file eval.c.

Referenced by `cmdCompile()`, `cmdLoad()`, `cmdStatus()`, `comApprove()`, `comCompareAllLvPairs()`, `comCompareWithContext()`, `evalBinaryConstraint()`, `evalInitialize()`, and `evalUnaryConstraint()`.

5.7.5.6 EvalMethodType `evalEvaluationMethod` = `EMTInterpreted`

Implements `evalmethod`. Definition at line 116 of file eval.c.

Referenced by `cmdCompile()`, `cmdLoad()`, `cmdStatus()`, `comApprove()`, `comCompareAllLvPairs()`, `comCompareWithContext()`, `evalBinaryConstraint()`, `evalInitialize()`, and `evalUnaryConstraint()`.

5.7.5.7 **EvalMethodType evalPeekValueMethod**

Implements `peekvaluemethod`. Definition at line 119 of file `eval.c`.

Referenced by `cmdStatus()`, `evalInitialize()`, and `evalTerm()`.

5.7.5.8 **EvalMethodType evalPeekValueMethod = EMTCCompiled**

Implements `peekvaluemethod`. Definition at line 119 of file `eval.c`.

Referenced by `cmdStatus()`, `evalInitialize()`, and `evalTerm()`.

5.7.5.9 **Boolean evalSloppySubsumesWarnings**

Implements `subsumesWarnings`.

If a constraint evaluation tries to access lexical information of the root node, the corresponding subexpression unconditionally evaluates to `FALSE`, and a warning is printed. This flag can be used to turn off the warnings; however the interpretation of the erroneous expression as `FALSE` remains. It is therefore recommended that you write all constraints so that they will not access features of the root node in the first place. Definition at line 113 of file `eval.c`.

Referenced by `cmdStatus()`, and `evalInitialize()`.

5.7.5.10 **Boolean evalSloppySubsumesWarnings = FALSE**

Implements `subsumesWarnings`.

If a constraint evaluation tries to access lexical information of the root node, the corresponding subexpression unconditionally evaluates to `FALSE`, and a warning is printed. This flag can be used to turn off the warnings; however the interpretation of the erroneous expression as `FALSE` remains. It is therefore recommended that you write all constraints so that they will not access features of the root node in the first place. Definition at line 113 of file `eval.c`.

Referenced by `cmdStatus()`, and `evalInitialize()`.

5.7.5.11 **int* has_cache**

At the moment, only the results of identical `has()` applications are cached because those are expected to be particularly expensive. In principle, this method could be extended to arbitrary identical subformulas.

The result of the application of `has()` number `X` at time point `T` is stored in cell `X * lock_width + T`. 2 means `TRUE`, 1 means `FALSE`, and 0 'unknown'. Definition at line 142 of file `eval.c`.

Referenced by `lock_tree()`, and `unlock_tree()`.

5.7.5.12 **int* has_cache = NULL**

At the moment, only the results of identical `has()` applications are cached because those are expected to be particularly expensive. In principle, this method could be extended to arbitrary identical subformulas.

The result of the application of has() number X at time point T is stored in cell $X * \text{lock_width} + T$. 2 means TRUE, 1 means FALSE, and 0 'unknown'. Definition at line 142 of file eval.c.

Referenced by lock_tree(), and unlock_tree().

5.7.5.13 int lock_counter

Notes whether evaluation is locked.

Since locks can be nested, this counter can rise above 1; any value higher than 0 means that cached results are still valid. Definition at line 126 of file eval.c.

Referenced by lock_tree(), and unlock_tree().

5.7.5.14 int lock_counter = 0

Notes whether evaluation is locked.

Since locks can be nested, this counter can rise above 1; any value higher than 0 means that cached results are still valid. Definition at line 126 of file eval.c.

Referenced by lock_tree(), and unlock_tree().

5.7.5.15 int lock_width

The number of time points in the lexeme graph that will be used in the evaluations Definition at line 131 of file eval.c.

Referenced by lock_tree(), and unlock_tree().

5.7.5.16 int lock_width = 0

The number of time points in the lexeme graph that will be used in the evaluations Definition at line 131 of file eval.c.

Referenced by lock_tree(), and unlock_tree().

5.7.5.17 String static_string_chunk_end

static string "chunk_end" Definition at line 74 of file eval.c.

Referenced by evalFinalize(), evalInitialize(), and evalTerm().

5.7.5.18 String static_string_chunk_end

static string "chunk_end" Definition at line 74 of file eval.c.

Referenced by evalFinalize(), evalInitialize(), and evalTerm().

5.7.5.19 String static_string_chunk_start

static string "chunk_start" Definition at line 73 of file eval.c.

Referenced by evalFinalize(), evalInitialize(), and evalTerm().

5.7.5.20 String [static_string_chunk_start](#)

static string "*chunk_start*" Definition at line 73 of file eval.c.
Referenced by evalFinalize(), evalInitialize(), and evalTerm().

5.7.5.21 String [static_string_chunk_type](#)

static string "*chunk_type*" Definition at line 75 of file eval.c.
Referenced by evalFinalize(), evalInitialize(), and evalTerm().

5.7.5.22 String [static_string_chunk_type](#)

static string "*chunk_type*" Definition at line 75 of file eval.c.
Referenced by evalFinalize(), evalInitialize(), and evalTerm().

5.7.5.23 String [static_string_from](#)

static string "*from*" Definition at line 70 of file eval.c.
Referenced by evalFinalize(), evalInitialize(), and evalTerm().

5.7.5.24 String [static_string_from](#)

static string "*from*" Definition at line 70 of file eval.c.
Referenced by evalFinalize(), evalInitialize(), and evalTerm().

5.7.5.25 String [static_string_id](#)

static string "*id*" Definition at line 68 of file eval.c.
Referenced by evalFinalize(), evalInitialize(), and evalTerm().

5.7.5.26 String [static_string_id](#)

static string "*id*" Definition at line 68 of file eval.c.
Referenced by evalFinalize(), evalInitialize(), and evalTerm().

5.7.5.27 String [static_string_info](#)

static string "*info*" Definition at line 72 of file eval.c.
Referenced by evalFinalize(), evalInitialize(), and evalTerm().

5.7.5.28 String [static_string_info](#)

static string "*info*" Definition at line 72 of file eval.c.
Referenced by evalFinalize(), evalInitialize(), and evalTerm().

5.7.5.29 String [static_string_to](#)

static string *"to"* Definition at line 71 of file eval.c.

Referenced by evalFinalize(), evalInitialize(), and evalTerm().

5.7.5.30 String [static_string_to](#)

static string *"to"* Definition at line 71 of file eval.c.

Referenced by evalFinalize(), evalInitialize(), and evalTerm().

5.7.5.31 String [static_string_word](#)

static string *"word"* Definition at line 69 of file eval.c.

Referenced by evalFinalize(), evalInitialize(), and evalTerm().

5.7.5.32 String [static_string_word](#)

static string *"word"* Definition at line 69 of file eval.c.

Referenced by evalFinalize(), evalInitialize(), and evalTerm().

5.8 HookCore - C Part in the core system

5.8.1 Detailed Description

The Hook module connects the core of the library to its outside world by a callback mechanism. Hook functions are to be supplied by the application programmer and will be called at strategic points in the application. For instance, a commonly used hook is [HOOK_PRINTF](#), that handles the output channels (see below).

So hooking the [HOOK_PRINTF](#) callback allows you to redefine and redirect the channels. The following callbacks are defined:

- [HOOK_CNBUILDNODES](#)
- [HOOK_EVAL](#)
- [HOOK_NSSEARCH](#)
- [HOOK_PRINTF](#)
- [HOOK_FLUSH](#)
- [HOOK_GLSINTERACTION](#)
- [HOOK_GETS](#)
- [HOOK_PROGRESS](#)
- [HOOK_PARTIALRESULT](#)
- [HOOK_ICINTERACTION](#)
- [HOOK_RESET](#)

As a part of that the output channels are handled here. The following output channels exist:

- [CDG_HINT](#)
- [CDG_INFO](#)
- [CDG_WARNING](#)
- [CDG_PROLOG](#)
- [CDG_EVAL](#)
- [CDG_SEARCHRESULT](#)
- [CDG_PROFILE](#)
- [CDG_HOOK](#)
- [CDG_ERROR](#)
- [CDG_DEBUG](#)
- [CDG_DEFAULT](#)
- [CDG_PROGRESS](#)
- [CDG_XML](#)

Note, that the term *hook* and *callback* are used mostly as synonyms. (Actually the term *hook* should be replaced by *callback* for clarity.)

As oposed to other modules some of the functions in this module are not prefixed by `hk` as the [Coding style](#) dictates. The reason for that lies in the relative importance of this module and some of its functions to the overall system, i.e. `cdgPrintf()`.

Data Structures

- struct [HookStruct](#)

Defines

- #define [CDG_DEBUG](#) (1L << 9)
- #define [CDG_DEFAULT](#) (1L << 10)
- #define [CDG_ERROR](#) (1L << 8)
- #define [CDG_EVAL](#) (1L << 4)
- #define [CDG_HINT](#) (1L << 0)
- #define [CDG_HOOK](#) (1L << 7)
- #define [CDG_INFO](#) (1L << 1)
- #define [CDG_PROFILE](#) (1L << 6)
- #define [CDG_PROGRESS](#) (1L << 11)
- #define [CDG_PROLOG](#) (1L << 3)
- #define [CDG_SEARCHRESULT](#) (1L << 5)
- #define [CDG_WARNING](#) (1L << 2)
- #define [CDG_XML](#) (1L << 12)
- #define [HOOK_CNBUILDNODES](#) 0
- #define [HOOK_EVAL](#) 1
- #define [HOOK_FLUSH](#) 4
- #define [HOOK_GETS](#) 7
- #define [HOOK_GLSINTERACTION](#) 6
- #define [HOOK_ICINTERACTION](#) 10
- #define [HOOK_NSSEARCH](#) 2
- #define [HOOK_PARTIALRESULT](#) 9
- #define [HOOK_PRINTF](#) 3
- #define [HOOK_PROGRESS](#) 8
- #define [HOOK_RESET](#) 5

Typedefs

- typedef [HookStruct](#) * [Hook](#)
- typedef void [HookFunction](#) ()

Functions

- void [cdgExecHook](#) (int hookNo,...)
- void [cdgFlush](#) (void)
- void [cdgGetString](#) (String buffer, int size)
- void [cdgPrintf](#) (int mode, String format,...)

- void `hkCallback` (String name, Boolean *var)
- void `hkFinalize` (void)
- int `hkFindNoOfHook` (String name)
- void `hkInitialize` (void)
- Boolean `hkValidate` (String name, String value, Boolean *var)

Variables

- Vector `hkHooks`
- Vector `hkHooks` = NULL
- unsigned long int `hkVerbosity`
- unsigned long int `hkVerbosity` = CDG_DEFAULT | CDG_ERROR | CDG_INFO | CDG_WARNING | CDG_SEARCHRESULT

5.8.2 Define Documentation

5.8.2.1 #define CDG_DEBUG (1L << 9)

identification of the `DEBUG` channel. This message increases the normal output verbosity by offering debug information. Definition at line 167 of file `hook.h`.

Referenced by `cmdStatus()`, `cnBuildEdges()`, `cnBuildLv()`, `cnBuildNodes()`, `cnBuildUpdateArcs()`, `cnOptimizeNet()`, `cnOptimizeNode()`, `cnPrintActiveLVs()`, `cnSortNodes()`, `cnTag()`, `comMake()`, `embedChunk()`, `evalBinary()`, `evalUnary()`, `findChunk()`, `getCategories()`, `getChunks()`, `getFakeChunksAt()`, `hkInitialize()`, `initRealChunker()`, `lgPartitions()`, `mergeChunk()`, `postProcessChunks()`, and `scSetScore()`.

5.8.2.2 #define CDG_DEFAULT (1L << 10)

identification of the `DEFAULT` channel. Unclassified messages of messages that should only be suppressed in very rare cases are emitted on the `DEFAULT` channel. Definition at line 174 of file `hook.h`.

Referenced by `cmdAnnotation()`, `cmdConstraint()`, `cmdDistance()`, `cmdEdges()`, `cmdHelp()`, `cmdHierarchy()`, `cmdHook()`, `cmdLevel()`, `cmdLexicon()`, `cmdLicense()`, `cmdNet()`, `cmdNonSpecCompatible()`, `cmdSection()`, `cmdStatus()`, `cmdVersion()`, `cmdWordgraph()`, `comApprove()`, `comCompile()`, `commandLoop()`, and `hkInitialize()`.

5.8.2.3 #define CDG_ERROR (1L << 8)

identification of the `ERROR` channel. This channel transmits messages about unexpected events that lead to an abortion of further computation. Compare to `WARNING`. Definition at line 160 of file `hook.h`.

Referenced by `cdgExecHook()`, `cdgPrintf()`, `chunkerChunk()`, `chunkerCommandValidate()`, `chunkerNew()`, `cmdAnno2Parse()`, `cmdAnnos2prolog()`, `cmdAnnotation()`, `cmdChunk()`, `cmdCompareParses()`, `cmdCompile()`, `cmdConstraint()`, `cmdDistance()`, `cmdEdges()`, `cmdHelp()`, `cmdHierarchy()`, `cmdHook()`, `cmdIncrementalCompletion()`, `cmdInputwordgraph()`, `cmdISearch()`, `cmdLevel()`, `cmdLevelsort()`, `cmdLoad()`, `cmdLs()`, `cmdNet()`, `cmdNetdelete()`, `cmdNetsearch()`, `cmdNewnet()`, `cmdNonSpecCompatible()`, `cmdParsedelete()`, `cmdParses2prolog()`, `cmdPrintParse()`, `cmdPrintParses()`, `cmdRenewnet()`, `cmdReset()`, `cmdSection()`, `cmdSet()`, `cmdShowlevel()`, `cmdStatus()`, `cmdTagger()`, `cmdTesting()`, `cmdUseconstraint()`, `cmdUselevel()`, `cmdUseLexicon()`, `cmdVerify()`, `cmdWeight()`, `cmdWordgraph()`, `cmdWriteAnno()`, `cmdWriteNet()`, `cmdWriteParses()`, `cmdWriteWordgraph()`, `comApprove()`, `comCompareAllLvPairs()`, `comCompareAllLvs()`, `comCompareLvs()`, `comCompareNets()`, `comCompareWithContext()`, `comCompile()`,

comFinitGrammar(), comInitGrammar(), comLoad(), comMake(), commandEval(), computeNoOfPathsFromStart(), computeNoOfPathsToEnd(), dbClose(), dbGetEntries(), dbLoadEntries(), dbOpen(), dbOpenIndexFile(), getChunks(), hkInitialize(), initChunker(), initFakeChunker(), initRealChunker(), lgAreDeletableNodes(), lgCompatibleNodes(), lgDistanceOfNodes(), parseGetGrapheme(), scSetScore(), smDelete(), smGetFlag(), smGetScore(), smSetAllFlags(), smSetFlag(), smSetScore(), terminateChild(), timerElapsed(), and unlock_tree().

5.8.2.4 #define CDG_EVAL (1L << 4)

identification of the EVAL channel.

Todo

The EVAL channel is bogus and never used.

Definition at line 130 of file hook.h.

Referenced by cmdStatus(), and hkInitialize().

5.8.2.5 #define CDG_HINT (1L << 0)

identification of the HINT channel.

Todo

The HINT channel is bogus and never used.

Definition at line 103 of file hook.h.

Referenced by cmdStatus(), and hkInitialize().

5.8.2.6 #define CDG_HOOK (1L << 7)

global hook flag. This flag switches on/off the complete hook system.

Todo

By no means this is an output channel. This define should be removed and replaced with a proper variable that is altered by the hook command.

Definition at line 153 of file hook.h.

Referenced by cmdHook(), cmdStatus(), cnBuildNodes(), evalBinaryConstraint(), evalConstraint(), and evalUnaryConstraint().

5.8.2.7 #define CDG_INFO (1L << 1)

identification of the INFO channel. Most of the output of the CDG library is tagged as INFO. A message to the INFO channel is *informal*. Definition at line 110 of file hook.h.

Referenced by cdgDeleteComputed(), chunkerChunk(), cmdAnno2Parse(), cmdAnnos2prolog(), cmdChunk(), cmdCompile(), cmdHook(), cmdIncrementalCompletion(), cmdInputwordgraph(), cmdISearch(), cmdLevelsort(), cmdLexicon(), cmdLoad(), cmdNetsearch(), cmdNewnet(), cmdNonSpecCompatible(), cmdParsedelete(), cmdParses2prolog(), cmdRenewnet(), cmdSet(), cmdShowlevel(), cmdStatus(), cmdUseconstraint(), cmdUselevel(), cmdVerify(), cmdWeight(), cnCallback(), cnPrintActiveLVs(), cnPrintInfo(), cnPrintParses(), cnTag(), cnUnaryPruning(), comLoad(), comMake(), comTranslate(), cvAnalyse(), cvPrint(), dbLoad(), dbLoadAll(), dbLoadEntries(), evalChunker(), hkInitialize(), and lgNewFinal().

5.8.2.8 #define CDG_PROFILE (1L << 6)

identification of the PROFILE channel. This channel is used to emit profiling information, that is time statistics on computations. Definition at line 144 of file hook.h.

Referenced by cmdChunk(), cmdIncrementalCompletion(), cmdISearch(), cmdNewnet(), cmdStatus(), cnBuildEdges(), cnBuildNodes(), and hkInitialize().

5.8.2.9 #define CDG_PROGRESS (1L << 11)

identification of the PROGRESS channel. Messages about progress are emitted on this channel and might be redirected over the HOOK_PROGRESS callback. Definition at line 181 of file hook.h.

Referenced by cdgPrintf(), cmdAnno2Parse(), cmdStatus(), dbLoadAll(), and hkInitialize().

5.8.2.10 #define CDG_PROLOG (1L << 3)

identification of the PROLOG channel.

Todo

The PROLOG channel is bogus and never used.

Definition at line 124 of file hook.h.

Referenced by cmdStatus(), and hkInitialize().

5.8.2.11 #define CDG_SEARCHRESULT (1L << 5)

identification of the SEARCHRESULT channel. Results in a parser are emitted on this channel. All parsing favours should obey to this channel semantics. Most of them do. Definition at line 137 of file hook.h.

Referenced by cmdNetsearch(), cmdStatus(), and hkInitialize().

5.8.2.12 #define CDG_WARNING (1L << 2)

identification of the WARNING channel. A message to this channel informs about an unexpected situation. Operations continue. If a normal continuation is not possible the message should be emitted on the ERROR channel. Definition at line 118 of file hook.h.

Referenced by cdgAgInsert(), cdgPrintf(), cmdAnno2Parse(), cmdLs(), cmdNewnet(), cmdStatus(), cnBuild(), cnBuildFinal(), cnDelete(), cnFindNode(), cnOptimizeNode(), cnPrintInfo(), comApprove(), commandLoop(), dbAvailable(), evalBinaryConstraint(), evalChunker(), evalConstraint(), evalFormula(), evalTerm(), evalUnaryConstraint(), evalValidateEvalMethod(), getCategory(), getChunks(), getNextArgument(), hkInitialize(), IgDeleteNode(), IgDeleteNodes(), IgMostProbablePath(), IgNewFinal(), IgNewIter(), IgSimultaneous(), peekValue(), and terminateChild().

5.8.2.13 #define CDG_XML (1L << 12)

identification of the PROGRESS channel. All xml output goes in this channel. See the write module for more information on xml output. Definition at line 188 of file hook.h.

Referenced by cmdStatus(), commandEval(), and hkInitialize().

5.8.2.14 #define HOOK_CNBUILDNODES 0

callback after building constraintnodes. Definition at line 43 of file hook.h.

Referenced by `cnBuildNodes()`, and `hkInitialize()`.

5.8.2.15 #define HOOK_EVAL 1

callback after evaluation of constraints. Definition at line 48 of file hook.h.

Referenced by `evalBinaryConstraint()`, `evalConstraint()`, `evalUnaryConstraint()`, `hkInitialize()`, and `hooker_init()`.

5.8.2.16 #define HOOK_FLUSH 4

callback for flushing the output channels. Definition at line 63 of file hook.h.

Referenced by `cdgFlush()`, `hkInitialize()`, `hooker_init()`, and `logFlush()`.

5.8.2.17 #define HOOK_GETS 7

callback in to get a string from the user. Definition at line 79 of file hook.h.

Referenced by `cdgGetString()`, `hkInitialize()`, and `hooker_init()`.

5.8.2.18 #define HOOK_GLSINTERACTION 6

callback for user interaction in the gls module. Definition at line 74 of file hook.h.

Referenced by `hkInitialize()`, and `hooker_init()`.

5.8.2.19 #define HOOK_ICINTERACTION 10

callback for user interaction in the incrementalcompletion module. Definition at line 94 of file hook.h.

Referenced by `hkInitialize()`, and `hooker_init()`.

5.8.2.20 #define HOOK_NSSEARCH 2

callback in netsearching. Definition at line 53 of file hook.h.

Referenced by `hkInitialize()`, and `hooker_init()`.

5.8.2.21 #define HOOK_PARTIALRESULT 9

callback to hand over partial results. Definition at line 89 of file hook.h.

Referenced by `hkInitialize()`, and `hooker_init()`.

5.8.2.22 #define HOOK_PRINTF 3

callback for printfing. Definition at line 58 of file hook.h.

Referenced by `cdgPrintf()`, `hkInitialize()`, `hooker_init()`, `logFlush()`, and `logPrintf()`.

5.8.2.23 `#define HOOK_PROGRESS 8`

callback to indicate progress in a heavy computation. Definition at line 84 of file `hook.h`.

Referenced by `cdgPrintf()`, `hkInitialize()`, and `hooker_init()`.

5.8.2.24 `#define HOOK_RESET 5`

notification that parses, constraint nets etc. have become invalid because of grammar changes. Definition at line 69 of file `hook.h`.

Referenced by `cdgDeleteComputed()`, `hkInitialize()`, and `hooker_init()`.

5.8.3 Typedef Documentation

5.8.3.1 `typedef HookStruct* Hook`

type of a Hook Definition at line 216 of file `hook.h`.

Referenced by `cdgExecHook()`, `cdgFlush()`, `cdgGetString()`, `cdgPrintf()`, `cmdHook()`, `evalHookHandle()`, `getHookCmd()`, `getHookHandle()`, `glsInteractionHookHandle()`, `hkFindNoOfHook()`, `hkInitialize()`, `hook_completion_function()`, `hooker_init()`, `ICinteractionHookHandle()`, `initHookResult()`, `logFlush()`, `netsearchHookHandle()`, `partialResultHookHandle()`, `progressHookHandle()`, `resetHookHandle()`, `setHookCmd()`, and `tclHookHandle()`.

5.8.3.2 `typedef void HookFunction()`

type of a callback function. This function is installed into a callback hook and executed whenever this callback is triggered. This is only a indicator definition without any grants on the arguments. But be aware of the arguments that are actually needed by every callback. Definition at line 201 of file `hook.h`.

Referenced by `hkInitialize()`, and `hooker_init()`.

5.8.4 Function Documentation

5.8.4.1 `void cdgExecHook (int hookNo, ...)`

execute a defined callback. This is the standard way to call a hook. If the corresponding `HookStruct::active` is set, its `HookStruct::count` is increased and its `HookStruct::function` is called handing over a pointer to the `HookStruct` we are executing and the entire `va_list` passed to this function. For example, `cdgExecHook(HOOK_PARTIALRESULT, parse)` is executing `HookStruct::function(hook, parse)`.

Parameters:

hookNo the identifier of the hook to be called, e.g. `HOOK_PROGRESS`

Definition at line 306 of file `hook.c`.

References `HookStruct::active`, `CDG_ERROR`, `cdgPrintf()`, `HookStruct::count`, `HookStruct::function`, `hkHooks`, `Hook`, and `HookStruct::name`.

Referenced by `cdgDeleteComputed()`, `cnBuildNodes()`, `evalBinaryConstraint()`, `evalConstraint()`, and `evalUnaryConstraint()`.

5.8.4.2 void `cdgFlush` (void)

default flush callback. This is the default callback for `HOOK_FLUSH`. If no hook is installed, i.e. `HookStruct::function` is `NULL`, then `flush` is called. In addition `writeXmlFlush()` is called to sync the `CDG_XML` output channel. Definition at line 454 of file `hook.c`.

References `HookStruct::count`, `HookStruct::function`, `hkHooks`, `Hook`, and `HOOK_FLUSH`.

Referenced by `cdgPrintf()`, `cnBuildEdges()`, `cnBuildIter()`, `comApprove()`, `comMake()`, `commandEval()`, and `comTranslate()`.

5.8.4.3 void `cdgGetString` (String *buffer*, int *size*)

default get string callback. This function gets one newline-terminated chunk of console input from the user. Up to `size` characters are transferred into `buffer`. For this purpose, `HOOK_GETS` is executed. If this hook has no associated function, `fgets()` is called instead with an additional argument of `stdin`. This provides a transparent way of reading input from a user. (Note that the normal interaction with the shell of the `cdgp` parser uses a different and non-transparent input routine.) Definition at line 375 of file `hook.c`.

References `HookStruct::count`, `HookStruct::function`, `hkHooks`, `Hook`, and `HOOK_GETS`.

5.8.4.4 void `cdgPrintf` (int *mode*, String *format*, ...)

output channel router. This function takes care of routing messages to appropriate output channels. The first argument `mode` defines bitwise the channels this message is to be routed to. The rest of the arguments is passed to the `HOOK_PRINTF` in `printf` manner (so see the manual pages for `printf`). We use `HOOK_PROGRESS` exclusive here when the `hkVerbosity` contains the `CDG_PROGRESS` flag and only `HOOK_PRINTF` otherwise. If we are producing xml output and the `CDG_WARNING` or `CDG_ERROR` channel is used, then the message doubled according to the `cdg-logfile.dtd` by `writeXmlVaNotification()`. Definition at line 403 of file `hook.c`.

References `CDG_ERROR`, `CDG_PROGRESS`, `CDG_WARNING`, `cdgFlush()`, `HookStruct::count`, `HookStruct::function`, `hkHooks`, `hkVerbosity`, `Hook`, `HOOK_PRINTF`, `HOOK_PROGRESS`, and `NULL`.

Referenced by `bPrint()`, `cdgAgInsert()`, `cdgDeleteComputed()`, `cdgExecHook()`, `chunkerChunk()`, `chunkerCommandValidate()`, `chunkerNew()`, `chunkerPrintChunks()`, `cmdAnno2Parse()`, `cmdAnnos2prolog()`, `cmdAnnotation()`, `cmdChunk()`, `cmdCompareParses()`, `cmdCompile()`, `cmdConstraint()`, `cmdDistance()`, `cmdEdges()`, `cmdHelp()`, `cmdHierarchy()`, `cmdHook()`, `cmdIncrementalCompletion()`, `cmdInputwordgraph()`, `cmdISearch()`, `cmdLevel()`, `cmdLevelsort()`, `cmdLexicon()`, `cmdLicense()`, `cmdLoad()`, `cmdLs()`, `cmdNet()`, `cmdNetdelete()`, `cmdNetsearch()`, `cmdNewnet()`, `cmdNonSpecCompatible()`, `cmdParsedelete()`, `cmdParses2prolog()`, `cmdPrintParse()`, `cmdPrintParses()`, `cmdRenewnet()`, `cmdReset()`, `cmdSection()`, `cmdSet()`, `cmdShowlevel()`, `cmdStatus()`, `cmdTagger()`, `cmdTesting()`, `cmdUseconstraint()`, `cmdUselevel()`, `cmdUseLexicon()`, `cmdVerify()`, `cmdVersion()`, `cmdWeight()`, `cmdWordgraph()`, `cmdWriteAnno()`, `cmdWriteNet()`, `cmdWriteParses()`, `cmdWriteWordgraph()`, `cnBuild()`, `cnBuildEdges()`, `cnBuildFinal()`, `cnBuildLv()`, `cnBuildNodes()`, `cnBuildUpdateArcs()`, `cnCallback()`, `cnDelete()`, `cnFindNode()`, `cnOptimizeNet()`, `cnOptimizeNode()`, `cnPrint()`, `cnPrintActiveLVs()`, `cnPrintEdge()`, `cnPrintInfo()`, `cnPrintNode()`, `cnPrintParses()`, `cnSortNodes()`, `cnTag()`, `cnUnaryPruning()`, `comApprove()`, `comCompareAllLvPairs()`, `comCompareAllLvs()`, `comCompareLvs()`, `comCompareNets()`, `comCompareWithContext()`, `comCompile()`, `comFinitGrammar()`, `comInitGrammar()`, `comLoad()`, `comMake()`, `commandEval()`, `commandLoop()`, `computeNoOfPathsFromStart()`, `computeNoOfPathsToEnd()`, `comTranslate()`, `cvAnalyse()`, `cvPrint()`, `dbAvailable()`, `dbClose()`, `dbGetEntries()`, `dbLoad()`, `dbLoadAll()`, `dbLoadEntries()`, `dbOpen()`, `dbOpen`

IndexFile(), embedChunk(), evalBinary(), evalBinaryConstraint(), evalChunker(), evalConstraint(), evalFormula(), evalTerm(), evalUnary(), evalUnaryConstraint(), evalValidateEvalMethod(), findChunk(), getCategories(), getCategory(), getChunks(), getFakeChunksAt(), getNextArgument(), initChunker(), initFakeChunker(), initRealChunker(), lgAreDeletableNodes(), lgCompatibleNodes(), lgDeleteNode(), lgDeleteNodes(), lgDistanceOfNodes(), lgMostProbablePath(), lgNewFinal(), lgNewIter(), lgPartitions(), lgPrint(), lgPrintNode(), lgSimultaneous(), mergeChunk(), parseGetGrapheme(), peekValue(), postProcessChunks(), printChunk(), scSetScore(), smDelete(), smGetFlag(), smGetScore(), smSetAllFlags(), smSetFlag(), smSetScore(), terminateChild(), timerElapsed(), and unlock_tree().

5.8.4.5 void hkCallback (String name, Boolean * var)

monitor access to variables in this module.

This function is installed using setRegister() in the [hkInitialize\(\)](#) function of this module. When a variable is changed [hkCallback\(\)](#) is called handling appropriate sideeffects depending on the new variable value. Right now only the variable `xml` is handled here.

Switching on `xml` writes an appropriate xml header to the CDG_XML channel. Switching off `xml` finishes all open xml tags. See the write module for more information on the xml output channel. Definition at line 141 of file hook.c.

Referenced by [hkInitialize\(\)](#).

5.8.4.6 void hkFinalize (void)

finalization of this module. After this call the CDG library is nearly shut down. That's why this finalizer is to be called at last after calling the finalizer of all other modules. Definition at line 286 of file hook.c.

References [hkHooks](#).

Referenced by [cdgFinalize\(\)](#).

5.8.4.7 int hkFindNoOfHook (String name)

get the numerical identifier of a named hook.

Parameters:

name the name of a known hook

Returns:

the numerical identifier on success or -1 if no hook of that name exists.

Definition at line 347 of file hook.c.

References `FALSE`, [hkHooks](#), [Hook](#), and [HookStruct::name](#).

Referenced by [cmdHook\(\)](#), [getHookCmd\(\)](#), and [setHookCmd\(\)](#).

5.8.4.8 void hkInitialize (void)

initialization of the hook module. This function must be called before the CDG library is functional. Definition at line 156 of file hook.c.

References [HookStruct::active](#), `CDG_DEBUG`, `CDG_DEFAULT`, `CDG_ERROR`, `CDG_EVAL`, `CDG_HINT`, `CDG_INFO`, `CDG_PROFILE`, `CDG_PROGRESS`, `CDG_PROLOG`, `CDG_SEARCHRESULT`,

CDG_WARNING, CDG_XML, HookStruct::cmd, HookStruct::count, FALSE, HookStruct::function, hk-Callback(), hkHooks, hkValidate(), hkVerbosity, Hook, HOOK_CNBUILDNODES, HOOK_EVAL, HOOK_FLUSH, HOOK_GETS, HOOK_GLSINTERACTION, HOOK_ICINTERACTION, HOOK_NSSEARCH, HOOK_PARTIALRESULT, HOOK_PRINTF, HOOK_PROGRESS, HOOK_RESET, HookFunction, HookStruct::name, HookStruct::no, NULL, and TRUE.

Referenced by cdgInitialize().

5.8.4.9 Boolean `hkValidate` (String *name*, String *value*, Boolean * *var*)

validate access to variables in this module.

This function is installed using setRegister() in the `hkInitialize()` function of this module. It is monitoring the access to the variable `xml` being registered in `hkInitialize()`. Definition at line 113 of file `hook.c`.

References FALSE, and TRUE.

Referenced by `hkInitialize()`.

5.8.5 Variable Documentation

5.8.5.1 Vector `hkHooks`

This Vector holds all defined hooks. It is initialized by the function `hkInitialize()`. After that, the symbolic constants can be used to execute any of the hooks. Definition at line 88 of file `hook.c`.

Referenced by `cdgExecHook()`, `cdgFlush()`, `cdgGetString()`, `cdgPrintf()`, `cmdHook()`, `getHookCmd()`, `hk-Finalize()`, `hkFindNoOfHook()`, `hkInitialize()`, `hook_completion_function()`, `hooker_init()`, `logFlush()`, `log-Printf()`, and `setHookCmd()`.

5.8.5.2 Vector `hkHooks` = NULL

This Vector holds all defined hooks. It is initialized by the function `hkInitialize()`. After that, the symbolic constants can be used to execute any of the hooks. Definition at line 88 of file `hook.c`.

Referenced by `cdgExecHook()`, `cdgFlush()`, `cdgGetString()`, `cdgPrintf()`, `cmdHook()`, `getHookCmd()`, `hk-Finalize()`, `hkFindNoOfHook()`, `hkInitialize()`, `hook_completion_function()`, `hooker_init()`, `logFlush()`, `log-Printf()`, and `setHookCmd()`.

5.8.5.3 unsigned long int `hkVerbosity`

bitwise encoding of active output channels.

This variable controls the currently active output channels. All output is performed by the function `cdg-Printf()`, whose first argument is a bitwise encoding denoting the output channels (e.g. `CDG_INFO`, `CDG_ERROR`, or `CDG_DEBUG` output) to which a message is to be routed.

All messages tagged with a bit that is not set in `hkVerbosity` are suppressed. Definition at line 101 of file `hook.c`.

Referenced by `cdgPrintf()`, `cmdHook()`, `cmdNetsearch()`, `cmdStatus()`, `cnBuildNodes()`, `cnPrintInfo()`, `cn-SortNodes()`, `cnTag()`, `comApprove()`, `commandEval()`, `dbLoadEntries()`, `evalBinaryConstraint()`, `eval-Constraint()`, `evalUnaryConstraint()`, and `hkInitialize()`.

5.8.5.4 unsigned long int `hkVerbosity` = `CDG_DEFAULT` | `CDG_ERROR` | `CDG_INFO` | `CDG_WARNING` | `CDG_SEARCHRESULT`

bitwise encoding of active output channels.

This variable controls the currently active output channels. All output is performed by the function `cdgPrintf()`, whose first argument is a bitwise encoding denoting the output channels (e.g. `CDG_INFO`, `CDG_ERROR`, or `CDG_DEBUG` output) to which a message is to be routed.

All messages tagged with a bit that is not set in `hkVerbosity` are suppressed. Definition at line 101 of file `hook.c`.

Referenced by `cdgPrintf()`, `cmdHook()`, `cmdNetsearch()`, `cmdStatus()`, `cnBuildNodes()`, `cnPrintInfo()`, `cnSortNodes()`, `cnTag()`, `comApprove()`, `commandEval()`, `dbLoadEntries()`, `evalBinaryConstraint()`, `evalConstraint()`, `evalUnaryConstraint()`, and `hkInitialize()`.

5.9 Hook - A callback system

5.9.1 Detailed Description

Author:

Michael Schulz

Modules

- group [HookCore](#) - C Part in the core system
- group [HookBindings](#) - Adaptor to the callback system

5.10 Lexemgraph - maintenance of lexem graphs

5.10.1 Detailed Description

Author:

Ingo Schroeder (see also AUTHORS and THANKS for more)

Date:

1997-03-04

Id

[lexemgraph.c](#),v 1.140 2004/09/27 17:07:05 micha Exp

Id

[lexemgraph.h](#),v 1.68 2004/09/01 14:01:31 micha Exp

Functions

- long long [computeNoOfPathsFromStart](#) ([LexemGraph](#) lg, [GraphemNode](#) gn, long long sofar, long long maximal)
- long long [computeNoOfPathsToEnd](#) ([LexemGraph](#) lg, [GraphemNode](#) gn, long long sofar, long long maximal)
- [GraphemNode](#) gnClone ([GraphemNode](#) gn, [Lattice](#) lat)
- Boolean [lgAreDeletableNodes](#) ([LexemGraph](#) lg, [List](#) lexemes)
- Boolean [lgAreDeletedNodes](#) ([LexemGraph](#) lg, [List](#) lexemes)
- [LexemGraph](#) lgClone ([LexemGraph](#) lg)
- Boolean [lgCompatibleNodes](#) ([LexemGraph](#) lg, [LexemNode](#) a, [LexemNode](#) b)
- Boolean [lgCompatibleSets](#) ([LexemGraph](#) lg, [List](#) a, [List](#) b)
- void [lgComputeDistances](#) ([LexemGraph](#) lg)
- void [lgComputeNoOfPaths](#) ([LexemGraph](#) lg)
- Boolean [lgContains](#) ([LexemGraph](#) lg, [String](#) form)
- Boolean [lgCopySelection](#) ([LexemGraph](#) destination, [LexemGraph](#) source)
- void [lgCopyTagScores](#) ([LexemGraph](#) destination, [LexemGraph](#) source)
- void [lgDelete](#) ([LexemGraph](#) lg)
- void [lgDeleteNode](#) ([LexemGraph](#) lg, [LexemNode](#) ln)
- void [lgDeleteNodes](#) ([LexemGraph](#) lg, [List](#) nodes)
- int [lgDistanceOfNodes](#) ([LexemGraph](#) lg, [LexemNode](#) a, [LexemNode](#) b)
- Boolean [lgForbiddenBy](#) ([LexemGraph](#) lg, [LexemNode](#) ln, [List](#) lexemes)
- void [lgInitialize](#) ()
- Boolean [lgIntersectingSets](#) ([List](#) a, [List](#) b)
- Boolean [lgIsDeletedNode](#) ([LexemGraph](#) lg, [LexemNode](#) ln)
- Boolean [lgIsEndNode](#) ([GraphemNode](#) n)
- Boolean [lgIsStartNode](#) ([GraphemNode](#) n)
- Boolean [lgLexemeInLexemNodeList](#) ([LexiconItem](#) le, [List](#) list)
- [List](#) [lgMakePath](#) ([LexemGraph](#) lg, [List](#) nodes)
- Boolean [lgMayModify](#) ([LexemGraph](#) lg, [GraphemNode](#) down, [GraphemNode](#) up)
- Boolean [lgMember](#) ([LexemNode](#) ln, [List](#) lexemes)
- [List](#) [lgMostProbablePath](#) ([LexemGraph](#) lg)
- [LexemGraph](#) lgNew ([Lattice](#) lat)
- Boolean [lgNewFinal](#) ([LexemGraph](#) lg)
- [LexemGraph](#) lgNewInit ()

- Boolean `IgNewIter` (`LexemGraph` `Ig`, `Arc` `arc`)
- Boolean `IgOverlap` (`LexemNode` `a`, `LexemNode` `b`)
- List `IgPartitions` (`GraphemNode` `gn`, `BitString` `features`)
- void `IgPrint` (`long unsigned int` `mode`, `LexemGraph` `Ig`)
- void `IgPrintNode` (`unsigned long` `mode`, `LexemNode` `ln`)
- List `IgQueryCat` (`LexemGraph` `Ig`, `GraphemNode` `gn`)
- void `IgRequireLexeme` (`LexemGraph` `Ig`, `ByteVector` `v`, `LexemNode` `ln`)
- void `IgRequireLexemes` (`LexemGraph` `Ig`, `ByteVector` `v`, `List` `which`)
- Boolean `IgSimultaneous` (`LexemNode` `a`, `LexemNode` `b`)
- Boolean `IgSpuriousUppercase` (`LexemGraph` `Ig`, `Arc` `arc`)
- Boolean `IgSubset` (`List` `a`, `List` `b`)
- Boolean `IgUpdateArcs` (`LexemGraph` `Ig`, `Lattice` `lat`, `List` `listArcs`)
- int `IgWidth` (`LexemGraph` `Ig`)

Variables

- Boolean `IgCompactLVs` = TRUE

5.10.2 Function Documentation

5.10.2.1 `long long computeNoOfPathsFromStart` (`LexemGraph` `Ig`, `GraphemNode` `gn`, `long long` `sofar`, `long long` `maximal`)

computes `LexemGraph::noOfPathsFromStart`

This function computes the number of paths leading to `gn` from the start of `Ig`. If `gn` corresponds to a start node, this is simply the number of lexeme nodes sprung from `gn`. Otherwise it is that number multiplied by the sum of the numbers of paths leading from the start to immediately preceding grapheme nodes. If `gn` is deleted, the number is always zero. Definition at line 128 of file `lexemgraph.c`.

References `GraphemNodeStruct::arc`, `CDG_ERROR`, `cdgPrintf()`, `GraphemNode`, `LexemGraphStruct::graphemnodes`, `GraphemNodeStruct::live`, `LexemGraphStruct::min`, `GraphemNodeStruct::no`, and `LexemGraphStruct::noOfPathsFromStart`.

Referenced by `IgComputeNoOfPaths()`.

5.10.2.2 `long long computeNoOfPathsToEnd` (`LexemGraph` `Ig`, `GraphemNode` `gn`, `long long` `sofar`, `long long` `maximal`)

computes `LexemGraph::noOfPathsToEnd`

This function computes the number of paths leading from `gn` to the end of `Ig`. If `gn` corresponds to an end node, this is simply the number of lexeme nodes sprung from `g`. Otherwise it is that number multiplied by the sum of the numbers of paths leading to the end from immediately following grapheme nodes. If `gn` is deleted, the number is always zero. Definition at line 180 of file `lexemgraph.c`.

References `GraphemNodeStruct::arc`, `CDG_ERROR`, `cdgPrintf()`, `GraphemNode`, `LexemGraphStruct::graphemnodes`, `GraphemNodeStruct::live`, `LexemGraphStruct::max`, `GraphemNodeStruct::no`, and `LexemGraphStruct::noOfPathsToEnd`.

Referenced by `IgComputeNoOfPaths()`.

5.10.2.3 GraphemNode gnClone (GraphemNode gn, Lattice lat)

Clone a grapheme node.

The field `GraphemNode::lexemes` is not set; the caller has to do that. (The two-way links between grapheme nodes and lexeme nodes can be set easier when all nodes are known.) Definition at line 1713 of file `lexemgraph.c`.

References `GraphemNodeStruct::arc`, `GraphemNodeStruct::chunk`, `GraphemNode`, `GraphemNodeStruct::lexemes`, `GraphemNodeStruct::lexemgraph`, `GraphemNodeStruct::no`, and `NULL`.

Referenced by `IgClone()`.

5.10.2.4 Boolean IgAreDeletableNodes (LexemGraph lg, List lexemes)

This function checks whether all lexeme nodes passed in *lexemes* can be deleted at the same time. This is the case if doing so will leave at least one complete path though the lexeme graph, according to the current state of deletions. For this end, the function checks whether the sum of the number of paths through each lexeme node is smaller than the total number of paths in *lg*.

Precondition:

lexemes must be a List of lexeme nodes with identical time spans. If this is not the case, the behaviour is undefined.

Definition at line 979 of file `lexemgraph.c`.

References `CDG_ERROR`, `cdgPrintf()`, `GraphemNode`, `LexemGraphStruct::isDeletedNode`, `GraphemNodeStruct::lexemes`, `LexemNode`, `LexemNodeStruct::no`, `GraphemNodeStruct::no`, `LexemGraphStruct::noOfPaths`, `LexemGraphStruct::noOfPathsFromStart`, `LexemGraphStruct::noOfPathsToEnd`, `NULL`, and `TRUE`.

Referenced by `cnOptimizeNode()`.

5.10.2.5 Boolean IgAreDeletedNodes (LexemGraph lg, List lexemes)

This checks if the *lexemes* have been deleted `TRUE` is returned, if not `FALSE` Definition at line 957 of file `lexemgraph.c`.

References `FALSE`, `LexemNode`, `IgIsDeletedNode()`, `NULL`, and `TRUE`.

Referenced by `cnBuildIter()`, `cnOptimizeNode()`, `cnPrint()`, and `IgComputeDistances()`.

5.10.2.6 LexemGraph IgClone (LexemGraph lg)

Clone a lexeme graph.

This performs a totally deep copy; even the underlying lattice, lexicon items etc. are cloned. Definition at line 1735 of file `lexemgraph.c`.

References `LexemNodeStruct::arc`, `LexemGraphStruct::chunks`, `LexemGraphStruct::distance`, `gnClone()`, `LexemNodeStruct::grapheme`, `GraphemNode`, `LexemGraphStruct::graphemnodes`, `LexemGraphStruct::isDeletedNode`, `LexemGraphStruct::lattice`, `LexemNodeStruct::lexem`, `LexemNode`, `IgComputeDistances()`, `IgComputeNoOfPaths()`, `IgCopyTagScores()`, `LexemNodeStruct::limit`, `LexemGraphStruct::max`, `LexemGraphStruct::min`, `LexemNodeStruct::no`, `GraphemNodeStruct::no`, `LexemGraphStruct::nodes`, `LexemGraphStruct::noOfPathsFromStart`, `LexemGraphStruct::noOfPathsToEnd`, and `NULL`.

5.10.2.7 Boolean lgCompatibleNodes (LexemGraph lg, LexemNode a, LexemNode b)

returns TRUE if lexem nodes *a* and *b* exist on one path.

This function checks whether, in principle, a complete path can exist through *lg* that includes both *a* and *b*. This is independent of the current state of deletions. In fact, the function merely checks whether the distance between the nodes is not 0 by using `lgDistanceOfNodes()`. Note that two nodes are not automatically compatible merely because they do not overlap in time. Also, a lexeme node is not compatible with itself by this definition. Definition at line 847 of file `lexemgraph.c`.

References `CDG_ERROR`, `cdgPrintf()`, `LexemNode`, `lgDistanceOfNodes()`, and `NULL`.

Referenced by `cnOptimizeNode()`, `lgCompatibleSets()`, `lgForbiddenBy()`, `lgMakePath()`, `lgRequireLexeme()`, and `lgRequireLexemes()`.

5.10.2.8 Boolean lgCompatibleSets (LexemGraph lg, List a, List b)

checks if these sets of lexemes are compatible, i.e. either unrelated or intersecting?

Precondition:

Both *a* and *b* must be Lists of lexeme nodes spanning the same respective time interval. If this is not the case, the behaviour is undefined.

This function checks whether both sets of lexeme nodes may be selected in a solution. This is defined as follows:

- If either set is empty, the result is TRUE
- If the first elements of *a* and *b* are compatible, the result is TRUE
- If the sets intersect, the result is TRUE
- Otherwise the result is FALSE

Definition at line 903 of file `lexemgraph.c`.

References `LexemNodeStruct::arc`, `FALSE`, `LexemNode`, `lgCompatibleNodes()`, `NULL`, and `TRUE`.

5.10.2.9 void lgComputeDistances (LexemGraph lg)

(re-)computes the distance matrix `LexemGraph::distance`

This function computes the distance between any two lexeme nodes in *lg* and stores the result in *lg->distance*. Definition at line 56 of file `lexemgraph.c`.

References `GraphemNodeStruct::arc`, `LexemGraphStruct::distance`, `GraphemNode`, `LexemGraphStruct::graphemnodes`, `GraphemNodeStruct::lexemes`, and `lgAreDeletedNodes()`.

Referenced by `cnRenew()`, `lgClone()`, `lgDeleteNode()`, `lgDeleteNodes()`, and `lgNewFinal()`.

5.10.2.10 void lgComputeNoOfPaths (LexemGraph lg)

computes # of paths possible in the graph, given the current state of deletions.

This function computes the number of paths possible in *lg*, according to the state of its Vector `LexemGraph::isDeletedNode`. It calls `computeNoOfPathsToEnd()` and `computeNoOfPathsFromStart()` for each

lexeme node. The total number of all paths is the sum of all numbers of paths leading to grapheme nodes that are end nodes. Definition at line 232 of file lexemgraph.c.

References `GraphemNodeStruct::arc`, `computeNoOfPathsFromStart()`, `computeNoOfPathsToEnd()`, `FALSE`, `GraphemNode`, `LexemGraphStruct::graphemnodes`, `LexemGraphStruct::isDeletedNode`, `GraphemNodeStruct::lexemes`, `LexemNode`, `GraphemNodeStruct::live`, `LexemGraphStruct::min`, `LexemNodeStruct::no`, `GraphemNodeStruct::no`, `LexemGraphStruct::nodes`, `LexemGraphStruct::noOfPaths`, `LexemGraphStruct::noOfPathsFromStart`, `LexemGraphStruct::noOfPathsToEnd`, `NULL`, and `TRUE`.

Referenced by `cnRenew()`, `IgClone()`, `IgDeleteNode()`, `IgDeleteNodes()`, and `IgNewFinal()`.

5.10.2.11 Boolean `IgContains` (`LexemGraph` *Ig*, `String` *form*)

Does a lexemgraph contain at least one instance of a given form?

This function checks whether *Ig* contains at least one instance of the form *form*. Capitalized versions of *form* are permissible if they are spurious (cf. `IgSpuriousUppercase()`). Definition at line 1657 of file lexemgraph.c.

References `GraphemNodeStruct::arc`, `FALSE`, `GraphemNode`, `LexemGraphStruct::graphemnodes`, `IgSpuriousUppercase()`, and `TRUE`.

5.10.2.12 Boolean `IgCopySelection` (`LexemGraph` *destination*, `LexemGraph` *source*)

Select the path in DST whose parts most closely match SRC.

This function inspects the undeleted words in *source* and undeletes those words in *destination* that most closely correspond to them. (This is necessary because two lexeme graphs built from the same lattice may have their nodes in different order, so you cannot simply re-use an `LexemGraph::isDeletedNode` vector across lexeme graphs.) Definition at line 1569 of file lexemgraph.c.

References `LexemNodeStruct::arc`, `FALSE`, `LexemGraphStruct::isDeletedNode`, `LexemNodeStruct::lexem`, `LexemNode`, `LexemNodeStruct::no`, `LexemGraphStruct::nodes`, `NULL`, and `TRUE`.

5.10.2.13 `void IgCopyTagScores` (`LexemGraph` *destination*, `LexemGraph` *source*)

This function simply transfers the field `LexemGraph::tagscore` from each node in *source* to the corresponding node in *destination*. (This is only useful to save repeated invocation of `taggerTag()` for two graphs produced from the same lattice.) Definition at line 1498 of file lexemgraph.c.

References `Chunk`, `GraphemNodeStruct::chunk`, `chunkerCloneChunk()`, `chunkerReplaceGraphemes()`, `LexemGraphStruct::chunks`, `GraphemNode`, `LexemNode`, `ChunkStruct::nodes`, `LexemGraphStruct::nodes`, `NULL`, `ChunkStruct::subChunks`, `LexemGraphStruct::tags`, and `LexemNodeStruct::tagscore`.

Referenced by `IgClone()`.

5.10.2.14 `void IgDelete` (`LexemGraph` *Ig*)

deletes `LexemGraph`

This function deallocates a lexeme graph. This deallocates all parts of the structure, even the lexeme nodes and lexical entries themselves. The lexicon remains unchanged as the `LexicalEntry` structures are merely clones of the structures in `inputCurrentGrammar`. Definition at line 1313 of file lexemgraph.c.

References `chunkerChunkDelete()`, `LexemGraphStruct::chunks`, `LexemGraphStruct::distance`, `GraphemNode`, `LexemGraphStruct::graphemnodes`, `LexemGraphStruct::isDeletedNode`, `LexemNodeStruct::lexem`,

GraphemNodeStruct::lexemes, LexemNode, LexemGraphStruct::nodes, LexemGraphStruct::noOfPathsFromStart, LexemGraphStruct::noOfPathsToEnd, NULL, and LexemGraphStruct::tags.

Referenced by cmdAnno2Parse(), cmdChunk(), cnDelete(), and lgNew().

5.10.2.15 void lgDeleteNode (LexemGraph lg, LexemNode ln)

deletes a node from the lexeme graph itself.

This function marks a lexeme node as deleted. It does this by setting the cell *ln->no* in the Vector *lg->isDeletedNode*. If this destroys the last possible path through *lg*, a warning is displayed. This function always re-computes the number of remaining paths in *lg*. Definition at line 1045 of file lexemgraph.c.

References CDG_WARNING, cdgPrintf(), LexemGraphStruct::isDeletedNode, LexemNodeStruct::lexem, LexemNode, lgComputeDistances(), lgComputeNoOfPaths(), LexemNodeStruct::no, LexemGraphStruct::noOfPaths, and TRUE.

Referenced by cnOptimizeNode().

5.10.2.16 void lgDeleteNodes (LexemGraph lg, List nodes)

delete a list of lexeme nodes

This function behaves as `lgDeleteNode()` were called on each element of the *nodes*, but it is more efficient since it only re-computes the number of remaining paths once. Definition at line 1075 of file lexemgraph.c.

References CDG_WARNING, cdgPrintf(), LexemGraphStruct::isDeletedNode, LexemNodeStruct::lexem, LexemNode, lgComputeDistances(), lgComputeNoOfPaths(), LexemNodeStruct::no, LexemGraphStruct::noOfPaths, NULL, and TRUE.

Referenced by cnOptimizeNode().

5.10.2.17 int lgDistanceOfNodes (LexemGraph lg, LexemNode a, LexemNode b)

returns a distance measure for two lexem nodes

This function computes the logical distance between *a* and *b*, measured in words. Usually this is just the corresponding element of `LexemGraph::distance`. If either of the nodes is underspecified it is treated as if it followed the latest specified lexeme node directly. Hence, the return value may be greater than value in `LexemGraph::distance`. Two underspecified lexeme nodes are considered to have distance zero. Definition at line 776 of file lexemgraph.c.

References CDG_ERROR, cdgPrintf(), LexemGraphStruct::distance, LexemNodeStruct::grapheme, LexemNode, GraphemNodeStruct::no, and NULL.

Referenced by cmdDistance(), lgCompatibleNodes(), and lgMayModify().

5.10.2.18 Boolean lgForbiddenBy (LexemGraph lg, LexemNode ln, List lexemes)

does existence of these lexemes exclude that lexeme.

Precondition:

lexemes must be a List of lexeme nodes with identical time spans. If this is not the case, the behaviour is undefined.

This function checks whether the List *lexemes* and the lexeme node *ln* can be selected in a solution. This is the case if either of the following holds:

- *lexemes* is empty
- *ln* is compatible with the first element of *lexemes*
- *lexemes* contains *ln*

In these cases FALSE is returned (*ln* is not forbidden). Otherwise TRUE is returned. Definition at line 872 of file lexemgraph.c.

References FALSE, LexemNode, lgCompatibleNodes(), and TRUE.

5.10.2.19 void lgInitialize ()

Initialize the input module.

This function initializes the module Lexemgraph and registers the variable *compactlevelvalues*. Definition at line 1615 of file lexemgraph.c.

References lgCompactLVs, and NULL.

Referenced by cdgInitialize().

5.10.2.20 Boolean lgIntersectingSets (List *a*, List *b*)

Do two lexeme lists intersect.

Precondition:

Both *a* and *b* must be Lists of lexeme nodes spanning the same respective time interval. If this is not the case, the behaviour is undefined.

This function checks whether *a* and *b* intersect. Definition at line 1211 of file lexemgraph.c.

References FALSE, LexemNode, lgSimultaneous(), NULL, and TRUE.

5.10.2.21 Boolean lgIsDeletedNode (LexemGraph *lg*, LexemNode *ln*)

This checks if a lexem node has been deleted TRUE is returned, if not FALSE. Definition at line 947 of file lexemgraph.c.

References LexemGraphStruct::isDeletedNode, LexemNode, and LexemNodeStruct::no.

Referenced by cnOptimizeNode(), lgAreDeletedNodes(), and lgMakePath().

5.10.2.22 Boolean lgIsEndNode (GraphemNode *n*)

returns TRUE if node is an end node

This function checks whether *n->arc->to* is equal to the maximal time point in the lexeme graph.

Todo

should nodes on segment boundaries of incremental parsed input be considered to be endnodes

Definition at line 758 of file lexemgraph.c.

References GraphemNodeStruct::arc, FALSE, GraphemNode, GraphemNodeStruct::lexemgraph, LexemGraphStruct::max, and NULL.

Referenced by cnIsEndNode().

5.10.2.23 Boolean lgIsStartNode (GraphemNode *n*)

returns TRUE if node is a start node

This function checks whether `n->arc->from` is equal to the minimal time point in the lexeme graph. Definition at line 741 of file `lexemgraph.c`.

References `GraphemNodeStruct::arc`, `FALSE`, `GraphemNode`, `GraphemNodeStruct::lexemgraph`, `LexemGraphStruct::min`, and `NULL`.

Referenced by `cnIsStartNode()`.

5.10.2.24 Boolean lgLexemeInLexemNodeList (LexiconItem *le*, List *list*)

This function checks whether at least one of the `LexemNode` structures in *list* points to a lexicon element *le*. Definition at line 1549 of file `lexemgraph.c`.

References `FALSE`, `LexemNodeStruct::lexem`, `LexemNode`, `NULL`, and `TRUE`.

5.10.2.25 List lgMakePath (LexemGraph *lg*, List *nodes*)

Takes a set of lexeme nodes, and extends it to a complete path through the graph, composed of undeleted `LexemNodes`. Returns `NULL` if this is impossible, It returns a List of lexeme nodes that

1. is a superset of *nodes*
2. corresponds to a complete path through the graph and
3. contains only undeleted lexeme nodes.

If this is not possible, `NULL` is returned.

We do this by simply appending arbitrary non-contradictory nodes until we have bound all time points. Note that for this approach to be correct, there must not be any undeleted dangling nodes in the graph. This condition must have ensured by `cnOptimizeNet()`. Definition at line 1250 of file `lexemgraph.c`.

References `LexemNodeStruct::arc`, `FALSE`, `LexemNode`, `lgCompatibleNodes()`, `lgIsDeletedNode()`, `LexemGraphStruct::max`, `LexemGraphStruct::min`, `LexemGraphStruct::nodes`, `NULL`, and `TRUE`.

5.10.2.26 Boolean lgMayModify (LexemGraph *lg*, GraphemNode *down*, GraphemNode *up*)

may these words modify each other?

Precondition:

lexemes must be a List of lexeme nodes with identical time spans. If this is not the case, the behaviour is undefined.

This function checks whether an `LevelValue` can exist with the modifiers *exemes* and a modifiee from *gn*. This is the case iff both can coexist on one path and do not overlap. Definition at line 824 of file `lexemgraph.c`.

References `GraphemNode`, `GraphemNodeStruct::lexemes`, `LexemNode`, `lgDistanceOfNodes()`, and `TRUE`.

Referenced by `cnBuildLevelValues()`.

5.10.2.27 Boolean `IgMember` (`LexemNode ln`, `List lexemes`)

is this lexeme a member of the this set?

Precondition:

lexemes must be a List of lexeme nodes with identical time spans. If this is not the case, the behaviour is undefined.

This function checks whether *ln* is an element of *lexemes*.

In the following cases *ln* is not a member (return FALSE);

- a NIL binding (*ln* is NULL) is not an element of anything
- an empty set (*lexemes* is NULL) has no member,
- *ln* belongs to another timespan
- *ln* is not contained literally in the set

Otherwise TRUE is returned. Definition at line 1152 of file `lexemgraph.c`.

References FALSE, `LexemNode`, `IgSimultaneous()`, and NULL.

Referenced by `IgSubset()`.

5.10.2.28 List `IgMostProbablePath` (`LexemGraph lg`)

Returns the most probable path, as defined by tagging scores. Definition at line 1676 of file `lexemgraph.c`.

References `CDG_WARNING`, `cdgPrintf()`, `GraphemNode`, `LexemGraphStruct::graphemnodes`, `LexemGraphStruct::lattice`, `GraphemNodeStruct::lexemes`, `LexemNode`, NULL, and `LexemNodeStruct::tagscore`.

5.10.2.29 `LexemGraph IgNew` (`Lattice lat`)

This function creates a lexeme graph from a Lattice *lat* and a cdg lexicon. For each arcs of the lattice a grapheme node is allocated and annotated with all possible lexical entries. (If there is no lexical entry for an arc, a warning is given, but processing continues.)

For each grapheme node, as many lexeme nodes are created as there are lexical alternatives in the lexicon.

Furthermore:

- `LexemGraph::isDeletedNode` is initialized to FALSE
- `LexemGraph::noOfPathsFromStart` and `LexemGraph::noOfPathsFromStart` are computed using `computeNoOfPaths()`.
- `LexemGraph::distance` is computed using `IgComputeDistances()`.

Definition at line 685 of file `lexemgraph.c`.

References `LexemGraphStruct::lattice`, `IgDelete()`, `IgNewFinal()`, `IgNewInit()`, `IgNewIter()`, and NULL.

Referenced by `cmdAnno2Parse()`, `cmdChunk()`, and `cnTag()`.

5.10.2.30 Boolean `IgNewFinal` (**LexemGraph** *lg*)

does the final computations for the lexemgraph

This function sets those fields of *lg* that can only be computed after all lexeme nodes are present:

- It initializes `LexemGraph::isDeletedNode` to all FALSE.
- It uses `IgComputeDistances()` to compute the distance between any to nodes in the graph.
- It applies `IgComputeNoOfPaths()` to compute the number of paths to and from all lexeme nodes in the graph. If no complete path exists, *lg* is deallocated with a warning.

The function can fail returning FALSE if there is no valid path through the lexeme graph. Definition at line 618 of file lexemgraph.c.

References `CDG_INFO`, `CDG_WARNING`, `cdgPrintf()`, `FALSE`, `LexemGraphStruct::graphemnodes`, `LexemGraphStruct::isDeletedNode`, `LexemGraphStruct::lattice`, `IgComputeDistances()`, `IgComputeNoOfPaths()`, `IgPrint()`, `LexemGraphStruct::max`, `LexemGraphStruct::noOfPaths`, `NULL`, `LexemGraphStruct::tags`, and `TRUE`.

Referenced by `IgNew()`, and `IgUpdateArcs()`.

5.10.2.31 **LexemGraph** `IgNewInit` ()

initializes the lexemgraph

This function returns a new `LexemGraph` structure with all fields initialized to meaningless values. In particular, it contains no nodes whatsoever. Definition at line 427 of file lexemgraph.c.

References `LexemGraphStruct::chunks`, `LexemGraphStruct::distance`, `LexemGraphStruct::graphemnodes`, `LexemGraphStruct::isDeletedNode`, `LexemGraphStruct::lattice`, `LexemGraphStruct::max`, `LexemGraphStruct::min`, `LexemGraphStruct::nodes`, `LexemGraphStruct::noOfPathsFromStart`, `LexemGraphStruct::noOfPathsToEnd`, `NULL`, and `LexemGraphStruct::tags`.

Referenced by `IgNew()`.

5.10.2.32 Boolean `IgNewIter` (**LexemGraph** *lg*, **Arc** *arc*)

Insert lexeme nodes into the `LexemGraph` that correspond the `Arc`.

This function builds all possible lexeme nodes for the specific *arc* and adds them to *lg*. It fails if there is no matching entry in the lexicon.

Maybe undo capitalisation introduced by orthographic convention.

If the written word is uppercase, but that uppercase-ness is suspect because it is at the start of a phrase and might be mere orthographic convention, we have to decide which version we use for lexicon lookup.

If our lexicon contains items for the lower-case version but none for the upper-case versions, we use only those; if it contains only items for the upper-case version, we use those; and if it contains neither, we allow both and hope that there is a lexical template which will catch this word.

We do not use the obvious solution - look up both versions whenever a word is spurious - because it has the following defect: If a sentence starts with 'Der', some naive lexical template could introduce a noun reading, and if POS tagging allows, it might actually survive even though it is exceedingly unlikely. Since we do not want this, we effectively force the reading to be 'der'.

Moral: If you really need to have open-class items in your lexicon that are near-homonymous with closed-class items, you can bloody well write proper lexicon items for them and not templates.

Much the same goes for words in ALL UPPER CAPS, except that those can occur anywhere in a sentence, not only at the start, and we have to check three different spellings instead of two.

In one-letter words, the intermediate version is indistinguishable from the third one, so we suppress it. Definition at line 454 of file lexemgraph.c.

References GraphemNodeStruct::arc, LexemNodeStruct::arc, CDG_WARNING, cdgPrintf(), GraphemNodeStruct::chunk, FALSE, LexemNodeStruct::grapheme, GraphemNode, LexemGraphStruct::graphemnodes, LexemGraphStruct::isDeletedNode, LexemNodeStruct::lexem, GraphemNodeStruct::lexemes, GraphemNodeStruct::lexemgraph, LexemNodeStruct::lexemgraph, LexemNode, IgSpuriousUppercase(), LexemNodeStruct::limit, LexemGraphStruct::max, max, LexemGraphStruct::min, min, GraphemNodeStruct::no, LexemNodeStruct::no, LexemGraphStruct::nodes, NULL, LexemNodeStruct::tagscore, and TRUE.

Referenced by IgNew(), and IgUpdateArcs().

5.10.2.33 Boolean IgOverlap (LexemNode *a*, LexemNode *b*)

Do these lexeme nodes overlap?

Returns TRUE if the two lexeme nodes have at least one time point in common.

This is subtly different from the more common question, "Can the two nodes coexist on one path?": two nodes can be compatible although they overlap if they are identical. Conversely, *a* and *b* may be incompatible even if they do not overlap if there is no path between them. Definition at line 1416 of file lexemgraph.c.

References LexemNodeStruct::arc, and LexemNode.

5.10.2.34 List IgPartitions (GraphemNode *gn*, BitString *features*)

partitions a set of lexeme nodes into equivalence classes

This function partitions the set of lexeme nodes of *gn* into equivalence classes. The equivalence relation used is the function inputCompareLeByAtts() with the argument *features*. The function returns a new List of new lists of lexemes. (The latter are re-used in ConstraintNode structures, the former are deallocated by cnBuildNodes().)

```

result = [];

FOR each lexeme l:
  IF l fits into one of the known classes,
    insert l there;
  ELSE
    create new class [lexem];
    insert the new class into result;
  FI
ROF

return result.
```

Definition at line 306 of file lexemgraph.c.

References CDG_DEBUG, cdgPrintf(), GraphemNode, LexemNodeStruct::lexem, GraphemNodeStruct::lexemes, LexemNode, IgCompactLVs, and NULL.

Referenced by cnBuildLevelValues().

5.10.2.35 void lgPrint (long unsigned int *mode*, [LexemGraph](#) *lg*)

print lexem graph

This function displays a textual representation of the lexeme graph *lg*. Definition at line 708 of file lexemgraph.c.

References [LexemNodeStruct::arc](#), [cdgPrintf\(\)](#), [chunkerPrintChunks\(\)](#), [LexemGraphStruct::chunks](#), [LexemGraphStruct::isDeletedNode](#), [LexemGraphStruct::lattice](#), [LexemNodeStruct::lexem](#), [LexemNode](#), [lgPrint\(\)](#), [LexemNodeStruct::no](#), [LexemGraphStruct::nodes](#), and [LexemNodeStruct::tagscore](#).

Referenced by [lgNewFinal\(\)](#), and [lgPrint\(\)](#).

5.10.2.36 void lgPrintNode (unsigned long *mode*, [LexemNode](#) *ln*)

prints out a lexeme node

This function displays the identifier and the time span of *ln* in the format `der_nom(0,1)`. Definition at line 1395 of file lexemgraph.c.

References [LexemNodeStruct::arc](#), [cdgPrintf\(\)](#), [LexemGraphStruct::isDeletedNode](#), [LexemNodeStruct::lexem](#), [LexemNodeStruct::lexemgraph](#), [LexemNode](#), and [LexemNodeStruct::no](#).

Referenced by [cnOptimizeNode\(\)](#).

5.10.2.37 List lgQueryCat ([LexemGraph](#) *lg*, [GraphemNode](#) *gn*)

What categories can this node represent? (Needed while tagging.)

This function queries the lexicon about what syntactical categories *gn* can represent. (The syntactical category is that feature whose index is `taggerCategoryIndex`.) This function is used to check whether an assignment by the tagger can be honored by the lexicon. Definition at line 1629 of file lexemgraph.c.

References [GraphemNode](#), [LexemNodeStruct::lexem](#), [GraphemNodeStruct::lexemes](#), [LexemNode](#), and `NULL`.

5.10.2.38 void lgRequireLexeme ([LexemGraph](#) *lg*, [ByteVector](#) *v*, [LexemNode](#) *ln*)

Takes a [Vector of Boolean](#), and sets all cells that correspond to the numbers of nodes incompatible with *ln*. This function can be used in combination with [lvVectorCompatible\(\)](#) to decide whether an LV is compatible with a set of other LVs. Definition at line 1428 of file lexemgraph.c.

References [LexemNode](#), [lgCompatibleNodes\(\)](#), [LexemNodeStruct::no](#), [LexemGraphStruct::nodes](#), and `TRUE`.

5.10.2.39 void lgRequireLexemes ([LexemGraph](#) *lg*, [ByteVector](#) *v*, [List](#) *which*)

This function is similar to [lgRequireLexeme\(\)](#), but takes a [List of lexeme nodes](#). It marks all those lexeme nodes that are incompatible with all lexemnodes of *which*. Definition at line 1449 of file lexemgraph.c.

References [LexemNode](#), [lgCompatibleNodes\(\)](#), [LexemNodeStruct::no](#), [LexemGraphStruct::nodes](#), and `TRUE`.

5.10.2.40 Boolean `IgSimultaneous` (`LexemNode a`, `LexemNode b`)

do the lexemes span the same time interval?

This function checks whether *a* and *b* cover the same time span. An argument of `NONSPEC` always causes `TRUE` to be returned. However, the `NULL` node is not simultaneous to any lexeme node, not even to another root node. Definition at line 1108 of file `lexemgraph.c`.

References `LexemNodeStruct::arc`, `CDG_WARNING`, `cdgPrintf()`, `FALSE`, `LexemNodeStruct::lexem`, `LexemNode`, `NULL`, and `TRUE`.

Referenced by `IgIntersectingSets()`, `IgMember()`, and `IgSubset()`.

5.10.2.41 Boolean `IgSpuriousUppercase` (`LexemGraph lg`, `Arc arc`)

Might this be a lowercase word that is spelled in upper case because of orthographic convention?

Spurious uppercase must be an upper case letter...

... followed by a lower case letter.

This is another instance of the "wordgraphs start at 0" assumption.

Ordinarily, this would be wrong, since the lexeme graph might start at some other time point. However, at this time `Ig->min` may not be initialized, so we can't check it. Since spurious upper case only occurs in written text, and weird time points occur mainly in recognizer output for spoken text, I'm letting it pass here. Definition at line 1785 of file `lexemgraph.c`.

References `FALSE`, `LexemGraphStruct::lattice`, and `TRUE`.

Referenced by `IgContains()`, and `IgNewIter()`.

5.10.2.42 Boolean `IgSubset` (`List a`, `List b`)

This function checks whether *a* is a subset of *b*.

Precondition:

Both *a* and *b* must be Lists of lexeme nodes spanning the same respective time interval. If this is not the case, the behaviour is undefined.

Definition at line 1173 of file `lexemgraph.c`.

References `FALSE`, `LexemNode`, `IgMember()`, `IgSimultaneous()`, `NULL`, and `TRUE`.

5.10.2.43 Boolean `IgUpdateArcs` (`LexemGraph lg`, `Lattice lat`, `List listArcs`)

updates the partial lexemgraph with the incoming arcs.

This function extends a lexeme graph by the `Arc` structures contained in *listArcs*. Definition at line 1476 of file `lexemgraph.c`.

References `LexemGraphStruct::distance`, `LexemGraphStruct::lattice`, `IgNewFinal()`, `IgNewIter()`, and `NULL`.

5.10.2.44 `int IgWidth` (`LexemGraph lg`)

Returns:

maximal ambiguity per time point

This function computes the maximal number of overlapping lexeme nodes for any time point in *lg*. Thus it gives an upper bound (not an estimate) of the average acoustical and lexical ambiguity of the graph. Definition at line 1362 of file lexemgraph.c.

References LexemNodeStruct::arc, LexemNode, LexemGraphStruct::max, and LexemGraphStruct::nodes.

5.10.3 Variable Documentation

5.10.3.1 Boolean `lgCompactLVs = TRUE`

This variable controls whether the levelvalues should be deflated if they are equivalent. Usually, we want this switched on, only for testing a value of FALSE might be appropriate. Definition at line 48 of file lexemgraph.c.

Referenced by cmdStatus(), lgInitialize(), and lgPartitions().

5.11 Scache - Cache structures for binary LV scores

5.11.1 Detailed Description

Author:

Michael Schulz

Date:

somewhen in 1998

A score cache serves much the same purpose as the score matrices in the edges of a constraint net, except that it can be used even if the constraint net has no edges in it. This simply means that the binary score of a pair of LVs will be computed when it is used for the first time rather than upon initializing the net.

Data Structures

- struct [ScoreCacheStruct](#)

Defines

- #define [DEBUG_SCORECACHE](#) 0
- #define [indexOfPair](#)(x, y) ((x<y) ? (((y*(y-1))>>1)+x) : (((x*(x-1))>>1)+y))

Typedefs

- typedef [ScoreCacheStruct](#) * [ScoreCache](#)

Functions

- void [scDelete](#) ([ScoreCache](#) cache)
- Number [scGetScore](#) ([ScoreCache](#) cache, LevelValue a, LevelValue b)
- void [scInitialize](#) ()
- [ScoreCache](#) [scNew](#) (int noValues)
- void [scSetScore](#) ([ScoreCache](#) cache, LevelValue a, LevelValue b, Number score)

Variables

- Boolean [scUseCache](#)
- Boolean [scUseCache](#) = FALSE

5.11.2 Define Documentation

5.11.2.1 #define [DEBUG_SCORECACHE](#) 0

debug flag Definition at line 41 of file [scache.c](#).

5.11.2.2 #define indexOfPair(x, y) ((x<y) ? (((y*(y-1))>>1)+x) : (((x*(x-1))>>1)+y))

determines the index of a given pair Definition at line 39 of file scache.c.

Referenced by scGetScore(), scNew(), and scSetScore().

5.11.3 Typedef Documentation

5.11.3.1 typedef ScoreCacheStruct* ScoreCache

Pointer to [ScoreCacheStruct](#) Definition at line 45 of file scache.h.

Referenced by scDelete(), scGetScore(), scNew(), and scSetScore().

5.11.4 Function Documentation

5.11.4.1 void scDelete ([ScoreCache](#) cache)

free memory from a cache

This function deallocates a cache. Definition at line 88 of file scache.c.

References [ScoreCacheStruct::data](#), and [ScoreCache](#).

Referenced by [cnDelete\(\)](#), [cnRenew\(\)](#), [comCompareAllLvPairs\(\)](#), and [comCompareWithContext\(\)](#).

5.11.4.2 Number scGetScore ([ScoreCache](#) cache, LevelValue a, LevelValue b)

retrieve score from cache if available, else return -1.0

This function returns the mutual score for the LVs **a** and **b** from the score. If it is not yet known, -1 is returned. Definition at line 102 of file scache.c.

References [ScoreCacheStruct::data](#), [ScoreCacheStruct::hits](#), [indexOfPair](#), [ScoreCache](#), and [ScoreCacheStruct::size](#).

Referenced by [evalBinary\(\)](#).

5.11.4.3 void scInitialize ()

initialize the module **Scache**. It registers the module's CDG variables. Definition at line 216 of file scache.c.

References [NULL](#), and [scUseCache](#).

Referenced by [cdgInitialize\(\)](#).

5.11.4.4 [ScoreCache](#) scNew (int size)

return an initial score-cache

This function returns a pointer to a new cache. The fields **count** and **hits** are initialized to 0. All elements of the underlying Vector are set to -1. Definition at line 62 of file scache.c.

References [ScoreCacheStruct::capacity](#), [ScoreCacheStruct::count](#), [ScoreCacheStruct::data](#), [ScoreCacheStruct::hits](#), [indexOfPair](#), [ScoreCache](#), and [ScoreCacheStruct::size](#).

Referenced by `cnBuildFinal()`, `cnRenew()`, `comCompareAllLvPairs()`, and `comCompareWithContext()`.

5.11.4.5 void `scSetScore` (**ScoreCache** *cache*, **LevelValue** *a*, **LevelValue** *b*, **Number** *score*)

store score in cache

This function registers a binary score in the cache. It behaves as follows:

- If **cache** is **NULL** the function returns.
- If either of the LVs still has its **no** set to -1, `registerValue()` is applied to it.
- A unique index is computed from the fields **no** of both LVs.
- The underlying Vector is resize if necessary.
- **score** is entered in the corresponding cell.

Definition at line 137 of file `scache.c`.

References `ScoreCacheStruct::capacity`, `CDG_DEBUG`, `CDG_ERROR`, `cdgPrintf()`, `ScoreCacheStruct::count`, `ScoreCacheStruct::data`, `indexOfPair`, `ScoreCache`, and `ScoreCacheStruct::size`.

Referenced by `evalBinary()`.

5.11.5 Variable Documentation

5.11.5.1 Boolean `scUseCache`

implements the CDG variable **cache**. If this flag is set, binary constraints are never evaluated twice on the same pair of LVs. The function `evalBinary()` will then simply call `scGetScore()` instead of `evalConstraint()`.

This variable is exported because it's cheaper for the calling function to check a boolean variable than calling a cache function which does nothing again and again. Definition at line 53 of file `scache.c`.

Referenced by `cmdIncrementalCompletion()`, `cmdStatus()`, `cnBuildFinal()`, `cnRenew()`, `comCompareAllLvPairs()`, `comCompareWithContext()`, `evalBinary()`, and `scInitialize()`.

5.11.5.2 Boolean `scUseCache = FALSE`

implements the CDG variable **cache**. If this flag is set, binary constraints are never evaluated twice on the same pair of LVs. The function `evalBinary()` will then simply call `scGetScore()` instead of `evalConstraint()`.

This variable is exported because it's cheaper for the calling function to check a boolean variable than calling a cache function which does nothing again and again. Definition at line 53 of file `scache.c`.

Referenced by `cmdIncrementalCompletion()`, `cmdStatus()`, `cnBuildFinal()`, `cnRenew()`, `comCompareAllLvPairs()`, `comCompareWithContext()`, `evalBinary()`, and `scInitialize()`.

5.12 Scorematrix - The matrix of scores appearing in each ConstraintEdge

5.12.1 Detailed Description

Author:

Ingo Schroeder

Date:

6/3/97

Note: The functions for accessing these matrices check whether the matrix is non- **NULL**, but they do not check whether the index is meaningful. Hence the behaviour of these functions is undefined if an invalid array index is passed to them.

Data Structures

- struct [ScoreMatrixStruct](#)
- struct [SMEntryStruct](#)

Typedefs

- typedef [ScoreMatrixStruct](#) * [ScoreMatrix](#)
- typedef [ScoreMatrixStruct](#) [ScoreMatrixStruct](#)
- typedef [SMEntryStruct](#) * [SMEntry](#)
- typedef [SMEntryStruct](#) [SMEntryStruct](#)

Functions

- void [smDelete](#) ([ScoreMatrix](#) sm)
- Boolean [smGetFlag](#) ([ScoreMatrix](#) sm, int r, int c)
- double [smGetScore](#) ([ScoreMatrix](#) sm, int r, int c)
- [ScoreMatrix](#) [smNew](#) (int r, int c)
- void [smSetAllFlags](#) ([ScoreMatrix](#) sm, Boolean flag)
- Boolean [smSetFlag](#) ([ScoreMatrix](#) sm, Boolean flag, int r, int c)
- double [smSetScore](#) ([ScoreMatrix](#) sm, double score, int r, int c)

5.12.2 Typedef Documentation

5.12.2.1 typedef struct [ScoreMatrixStruct](#)* [ScoreMatrix](#)

type of matrix structure pointer Definition at line 27 of file scorematrix.h.

Referenced by [smDelete\(\)](#), [smGetFlag\(\)](#), [smGetScore\(\)](#), [smNew\(\)](#), [smSetAllFlags\(\)](#), [smSetFlag\(\)](#), and [smSetScore\(\)](#).

5.12.2.2 typedef struct [ScoreMatrixStruct](#) [ScoreMatrixStruct](#)

type of score matrix structure Definition at line 81 of file scorematrix.c.

Referenced by [smNew\(\)](#).

5.12.2.3 typedef [SMEntryStruct](#)* [SMEntry](#)

type of matrix entry pointer Definition at line 54 of file scorematrix.c.

Referenced by [smNew\(\)](#).

5.12.2.4 typedef struct [SMEntryStruct](#) [SMEntryStruct](#)

type of matrix entry structure Definition at line 51 of file scorematrix.c.

Referenced by [smNew\(\)](#).

5.12.3 Function Documentation

5.12.3.1 void [smDelete](#) ([ScoreMatrix](#) *sm*)

deletes score matrix

This function deallocates a **ScoreMatrix**. It first deallocates all elements of **sm->entries[]** and then the array itself as well as the **ScoreMatrixStruct**. Definition at line 122 of file scorematrix.c.

References [CDG_ERROR](#), [cdgPrintf\(\)](#), [ScoreMatrixStruct::cols](#), [ScoreMatrixStruct::entries](#), [NULL](#), [ScoreMatrixStruct::rows](#), and [ScoreMatrix](#).

Referenced by [cnBuildEdges\(\)](#), and [cnDeleteEdge\(\)](#).

5.12.3.2 Boolean [smGetFlag](#) ([ScoreMatrix](#) *sm*, int *r*, int *c*)

retrieves a score matrix element flag

Returns:

the **flag** of the specified element.

Definition at line 211 of file scorematrix.c.

References [CDG_ERROR](#), [cdgPrintf\(\)](#), [ScoreMatrixStruct::entries](#), [SMEntryStruct::flag](#), [NULL](#), [ScoreMatrixStruct::rows](#), and [ScoreMatrix](#).

5.12.3.3 double [smGetScore](#) ([ScoreMatrix](#) *sm*, int *r*, int *c*)

retrieves a score matrix element score Definition at line 197 of file scorematrix.c.

References [CDG_ERROR](#), [cdgPrintf\(\)](#), [ScoreMatrixStruct::entries](#), [NULL](#), [ScoreMatrixStruct::rows](#), [SMEntryStruct::score](#), and [ScoreMatrix](#).

Referenced by [cnPrintEdge\(\)](#).

5.12.3.4 [ScoreMatrix](#) [smNew](#) (int *r*, int *c*)

creates and returns a new score matrix

This function allocates a new **ScoreMatrix** and returns a pointer to it. It allocates an array of size **r * c** . All elements are initialized to pairs of the form **(0.0, FALSE)** . Definition at line 94 of file scorematrix.c.

References ScoreMatrixStruct::cols, ScoreMatrixStruct::entries, FALSE, SMEntryStruct::flag, ScoreMatrixStruct::rows, SMEntryStruct::score, ScoreMatrix, ScoreMatrixStruct, SMEntry, and SMEntryStruct.

Referenced by cnBuildEdges().

5.12.3.5 void smSetAllFlags (ScoreMatrix *sm*, Boolean *f*)

sets flags of all matrix elements to a new value

This function sets the **flag** of all elements of **sm->entries[]** to **flag**. Definition at line 180 of file scorematrix.c.

References CDG_ERROR, cdgPrintf(), ScoreMatrixStruct::cols, ScoreMatrixStruct::entries, SMEntryStruct::flag, NULL, ScoreMatrixStruct::rows, and ScoreMatrix.

5.12.3.6 Boolean smSetFlag (ScoreMatrix *sm*, Boolean *f*, int *r*, int *c*)

sets a flag matrix element to a new value, returns old value Definition at line 159 of file scorematrix.c.

References CDG_ERROR, cdgPrintf(), ScoreMatrixStruct::entries, SMEntryStruct::flag, NULL, ScoreMatrixStruct::rows, and ScoreMatrix.

Referenced by cnBuildEdges().

5.12.3.7 double smSetScore (ScoreMatrix *sm*, double *d*, int *r*, int *c*)

sets a score matrix element to a new value, returns old value Definition at line 141 of file scorematrix.c.

References CDG_ERROR, cdgPrintf(), ScoreMatrixStruct::entries, NULL, ScoreMatrixStruct::rows, SMEntryStruct::score, and ScoreMatrix.

Referenced by cnBuildEdges().

5.13 Skel - A Skeleton Module

5.13.1 Detailed Description

Author:

your name

Date:

date of birth

Demonstration of the current [coding style](#) and blueprint for new modules.

This module demonstrates how a module for the CDG typically looks like. In addition it serves as a rough demonstration of some of the doxygen capabilities. See [skel.h](#) and [skel.c](#) for the sources of this documentation.

Note, that most of the documentation of a module should be placed into its implementations file (e.g. [skel.c](#)) and only the documentation of exported data structures (e.g. `MyExportedType`) and exported macros are left in the declarations file (e.g. [skel.h](#)). So avoid redundant documentation of functions and stuff in both places.

Data Structures

- struct [MyExportedTypeStruct](#)
- struct [MyPrivateTypeStruct](#)

Defines

- `#define` [SOMEMACRO](#)

Typedefs

- typedef [MyExportedTypeStruct](#) * [MyExportedType](#)
- typedef [MyPrivateTypeStruct](#) * [MyPrivateType](#)

Functions

- int [myExportedFunction](#) (int a, int b)
- int [myPrivateFunction](#) (int a, int b)

Variables

- int [MyExportedTypeStruct::memberA](#)
- int [MyExportedTypeStruct::memberB](#)
- int [myExportedVariable](#)
- int [myExportedVariable](#)
- int [myPrivateVariable](#)

5.13.2 Define Documentation

5.13.2.1 #define SOMEMACRO

short description Definition at line 28 of file skel.h.

5.13.3 Typedef Documentation

5.13.3.1 typedef MyExportedTypeStruct* MyExportedType

type definition Definition at line 40 of file skel.h.

5.13.3.2 typedef MyPrivateTypeStruct* MyPrivateType

type definition Definition at line 52 of file skel.c.

5.13.4 Function Documentation

5.13.4.1 int myExportedFunction (int *a*, int *b*)

short description. long description

Parameters:

a comments on *a*

b comments on *b*

Returns:

the result

Definition at line 86 of file skel.c.

References myExportedVariable.

5.13.4.2 int myPrivateFunction (int *a*, int *b*) [inline, static]

short description. long description

Parameters:

a comments on *a*

b comments on *b*

Returns:

the result

Definition at line 74 of file skel.c.

References myPrivateVariable.

5.13.5 Variable Documentation

5.13.5.1 `MyExportedTypeStruct::memberA` [inherited]

short description. long description.

5.13.5.2 `MyExportedTypeStruct::memberB` [inherited]

short description. long description.

5.13.5.3 `int myExportedVariable`

short description. long description Definition at line 57 of file skel.c.

Referenced by `myExportedFunction()`.

5.13.5.4 `int myExportedVariable`

short description. long description Definition at line 57 of file skel.c.

Referenced by `myExportedFunction()`.

5.13.5.5 `int myPrivateVariable` [static]

short description. This variable is private to this module Definition at line 60 of file skel.c.

Referenced by `myPrivateFunction()`.

5.14 Timer - timekeeping functions

5.14.1 Detailed Description

Author:

Michael Schulz

Date:

somewhen in 1998

This module exports functions for measuring the time spent in various parts of the application. The time measured is always user time + kernel time.

Defines

- #define [MyMAXLONG](#) ((long)~(1L << ((8 * (int)sizeof(long)) - 1)))

Typedefs

- typedef timeval TimerStruct * [Timer](#)
- typedef clock_t [TimerFast](#)

Functions

- void [expire](#) (int n)
- unsigned long [timerElapsed](#) ([Timer](#) timer)
- unsigned long [timerElapsedFast](#) ([TimerFast](#) timer)
- void [timerFree](#) ([Timer](#) timer)
- void [timerInitialize](#) (void)
- [Timer](#) [timerNew](#) (void)
- void [timerSetAlarm](#) (long unsigned int limit)
- void [timerStart](#) ([Timer](#) timer)
- [TimerFast](#) [timerStartFast](#) (void)
- void [timerStopAlarm](#) (void)

Variables

- double [timerClockTicks](#) = -1
- Boolean [timerExpired](#)
- Boolean [timerExpired](#)

5.14.2 Define Documentation

5.14.2.1 #define MyMAXLONG ((long)~(1L << ((8 * (int)sizeof(long)) - 1)))

a local macro to define the maximal long value Definition at line 28 of file timer.h.

5.14.3 Typedef Documentation

5.14.3.1 typedef struct timeval TimerStruct* **Timer**

the slow timer Definition at line 32 of file timer.h.

Referenced by cmdChunk(), cmdIncrementalCompletion(), cmdISearch(), cmdNewnet(), cnBuildEdges(), cnBuildNodes(), timerElapsed(), timerFree(), timerNew(), and timerStart().

5.14.3.2 typedef clock_t **TimerFast**

the fast timer Definition at line 31 of file timer.h.

Referenced by timerElapsedFast(), and timerStartFast().

5.14.4 Function Documentation

5.14.4.1 void expire (int *n*) [static]

Signal handler for when timer expires.

this function sets **timerexpired**. Definition at line 156 of file timer.c.

References timerExpired, and TRUE.

Referenced by timerSetAlarm().

5.14.4.2 unsigned long timerElapsed (**Timer** *timer*)

Return milliseconds elapsed since last **timerStart()** on this timer.

this function returns the difference between the time used so far and a time determined earlier by **timer-start()** in milliseconds. Definition at line 130 of file timer.c.

References CDG_ERROR, cdgPrintf(), and Timer.

Referenced by cmdChunk(), cmdIncrementalCompletion(), cmdISearch(), cmdNewnet(), cnBuildEdges(), cnBuildNodes(), comApprove(), comCompareAllLvPairs(), and comCompareWithContext().

5.14.4.3 unsigned long timerElapsedFast (**TimerFast** *timer*)

return milliseconds elapsed since last timerStartFast

this function returns the difference between the time used so far and a time determined earlier by **timer-startfast()** in milliseconds. Definition at line 72 of file timer.c.

References timerClockTicks, and TimerFast.

5.14.4.4 void timerFree (**Timer** *timer*)

frees the timer memory

this function deallocates a **timer**. Definition at line 116 of file timer.c.

References NULL, and Timer.

Referenced by cmdChunk(), cmdIncrementalCompletion(), cmdISearch(), cmdNewnet(), cnBuildEdges(), cnBuildNodes(), and comFreeApprover().

5.14.4.5 void timerInitialize (void)

timerInitialize

this function stops any alarms pending and initializes the variable **timerclockticks**. Definition at line 244 of file timer.c.

References timerClockTicks, and timerStopAlarm().

Referenced by cdgInitialize().

5.14.4.6 Timer timerNew (void)

allocates memory for the timer and calls timerStart Definition at line 103 of file timer.c.

References Timer, and timerStart().

Referenced by cmdChunk(), cmdIncrementalCompletion(), cmdISearch(), cmdNewnet(), cnBuildEdges(), cnBuildNodes(), and comNewApprover().

5.14.4.7 void timerSetAlarm (long unsigned int limit)

Set an alarm-clock that will set the flag timerExpired after “limit” milliseconds.

this function sets a time limit for the completion of a task. after **limit** milliseconds the variable **timerexpired** is asynchronously set to ~1 by **expire()**. this can be used as follows:

```
\ \ * build lvs for one second \ * \ /
```

```
void fastbuildvalues( constraintnet net, leveldecl level, ConstraintNode cn, List modifiers) {
```

```
LexemNode modiffee; * LevelValue lv;
```

```
int i; List l; String label;
```

```
timerSetAlarm(1000);
```

```
for (i = 0; i < vectorSize(net->lexemgraph->nodes); i++) { modiffee = (LexemNode) vectorElement(net->lexemgraph->nodes, i)
```

```
for (l = level->labels; l != NULL; l = l->next) { label = (String) l->item;
```

```
lv = lvNew(modifiers, level, label, modiffee); vectorAddElement(node->values, lv); }
```

```
if(timerExpired) { break; }
```

```
}
```

```
} Definition at line 204 of file timer.c.
```

References expire(), FALSE, NULL, and timerExpired.

Referenced by cmdFrobbing(), and cmdShift().

5.14.4.8 void timerStart (Timer timer)

Store number of cycles elapsed since program start (user + system) in timer.

This function performs a call to **getrusage()** and fills the corresponding fields in the **Timer** structure. Definition at line 88 of file timer.c.

References NULL, and Timer.

Referenced by comApprove(), comCompareAllLvPairs(), comCompareWithContext(), and timerNew().

5.14.4.9 **TimerFast** timerStartFast (void)

return clocks elapse since program start (user + system)

this function returns the time used so far by the process, as computed by the system function **clock()**=. Definition at line 61 of file timer.c.

References TimerFast.

5.14.4.10 void timerStopAlarm (void)

stop the alarm clock

This function cancels the alarm set by **timerSetAlarm()**. Definition at line 229 of file timer.c.

References FALSE, and timerExpired.

Referenced by cmdFrobbing(), cmdShift(), and timerInitialize().

5.14.5 Variable Documentation

5.14.5.1 double **timerClockTicks** = -1 [static]

initialized correctly in timerInitialize Definition at line 48 of file timer.c.

Referenced by timerElapsedFast(), and timerInitialize().

5.14.5.2 Boolean **timerExpired**

This variable **timerExpired**= is a flag to be set after a user-defined timespan has elapsed. Definition at line 46 of file timer.c.

Referenced by expire(), timerSetAlarm(), and timerStopAlarm().

5.14.5.3 Boolean **timerExpired**

This variable **timerExpired**= is a flag to be set after a user-defined timespan has elapsed. Definition at line 46 of file timer.c.

Referenced by expire(), timerSetAlarm(), and timerStopAlarm().

5.15 HookBindings - Adaptor to the callback system

5.15.1 Detailed Description

Todo

Please explain this module.

Data Structures

- struct [HookResultStruct](#)
- struct [LoggerStruct](#)

Typedefs

- typedef [HookResultStruct](#) * [HookResult](#)
- typedef [LoggerStruct](#) * [Logger](#)
- typedef int [TclResultType](#)

Enumerations

- enum [HookResultType](#) { [HTNone](#), [HTInt](#), [HTString](#), [HTError](#) }

Functions

- [TclResultType](#) [evalHookHandle](#) ([Hook](#) hook, va_list ap)
- [TclResultType](#) [getHookCmd](#) (char *hookName)
- [TclResultType](#) [getHookHandle](#) ([Hook](#) hook, String buffer, int size)
- [TclResultType](#) [glsInteractionHookHandle](#) ([Hook](#) hook, va_list ap)
- void [hooker_init](#) ([Tcl_Interp](#) *interp)
- [TclResultType](#) [ICinteractionHookHandle](#) ([Hook](#) hook, va_list ap)
- void [initHookResult](#) ([Hook](#) hook)
- [TclResultType](#) [logFlush](#) ()
- int [loggerSize](#) (int maxsize)
- [TclResultType](#) [logPrintf](#) (char *format, va_list ap)
- [TclResultType](#) [logWrite](#) (char *text)
- [TclResultType](#) [logWriteChar](#) (char c)
- [TclResultType](#) [netsearchHookHandle](#) ([Hook](#) hook, va_list ap)
- [TclResultType](#) [partialResultHookHandle](#) ([Hook](#) hook, va_list ap)
- [TclResultType](#) [progressHookHandle](#) ([Hook](#) hook, char *format, va_list ap)
- [TclResultType](#) [resetHookHandle](#) ([Hook](#) hook, va_list ap)
- [TclResultType](#) [setHookCmd](#) (char *hookName, char *cmd)
- [TclResultType](#) [tclHookHandle](#) ([Hook](#) hook, va_list ap)

Variables

- [Tcl_Interp](#) * [hkInterp](#)
- [Tcl_Interp](#) * [hkInterp](#)
- [HookResult](#) [hkResult](#)
- [HookResult](#) [hkResult](#)
- [Logger](#) [logger](#)

5.15.2 Typedef Documentation

5.15.2.1 typedef [HookResultStruct*](#) [HookResult](#)

type of a HookResult Definition at line 73 of file hooker.h.

Referenced by hooker_init().

5.15.2.2 typedef [LoggerStruct*](#) [Logger](#)

type of a Logger Definition at line 46 of file hooker.h.

Referenced by hooker_init().

5.15.2.3 typedef int [TclResultType](#)

tagging the tcl result type. This typedef is used to allow special typemaps in the swig interface. Definition at line 81 of file hooker.h.

Referenced by evalHookHandle(), getHookCmd(), getHookHandle(), glsInteractionHookHandle(), ICinteractionHookHandle(), logFlush(), logPrintf(), logWrite(), logWriteChar(), netsearchHookHandle(), partialResultHookHandle(), progressHookHandle(), resetHookHandle(), setHookCmd(), and tclHookHandle().

5.15.3 Enumeration Type Documentation

5.15.3.1 enum [HookResultType](#)

type of HookResults. Definition at line 51 of file hooker.h.

5.15.4 Function Documentation

5.15.4.1 [TclResultType](#) evalHookHandle ([Hook](#) *hook*, *va_list ap*)

evalHookHandle: called inside evalConstraint when a constraint fails the constraint can be unary or binary; levelvalues are still assigned arg1 : constraint Definition at line 356 of file hooker.c.

References HookStruct::cmd, hkInterp, hkResult, Hook, initHookResult(), NULL, TclResultType, and HookResultStruct::type.

Referenced by hooker_init().

5.15.4.2 [TclResultType](#) getHookCmd (*char * hookName*)

getHookCmd: get the tcl-command of a named hook Definition at line 245 of file hooker.c.

References HookStruct::cmd, HookResultStruct::data, hkFindNoOfHook(), hkHooks, hkInterp, hkResult, HookResultStruct::hook, Hook, NULL, TclResultType, and HookResultStruct::type.

5.15.4.3 [TclResultType](#) getHookHandle ([Hook](#) *hook*, *String buffer*, *int size*)

getHookHandle: handle gets from within tcl Definition at line 426 of file hooker.c.

References HookStruct::cmd, hkInterp, hkResult, Hook, initHookResult(), NULL, TclResultType, and HookResultStruct::type.

Referenced by hooker_init().

5.15.4.4 **TclResultType** glsInteractionHookHandle (**Hook** hook, va_list ap)

glsInteractionHookHandle: called inside the gls-module when a interaction with the algorithm is desired
arg1 : GlsNet glsNet arg2 : String message Definition at line 391 of file hooker.c.

References HookStruct::cmd, hkInterp, hkResult, Hook, initHookResult(), NULL, TclResultType, and HookResultStruct::type.

Referenced by hooker_init().

5.15.4.5 **void** hooker_init (Tcl_Interp *interp)

initialize the module when loading this stores the tcl-hooks cdgHooks Definition at line 67 of file hooker.c.

References LoggerStruct::buffer, HookStruct::cmd, evalHookHandle(), HookStruct::function, getsHookHandle(), glsInteractionHookHandle(), hkHooks, hkInterp, hkResult, Hook, HOOK_EVAL, HOOK_FLUSH, HOOK_GETS, HOOK_GLSINTERACTION, HOOK_ICINTERACTION, HOOK_NSSEARCH, HOOK_PARTIALRESULT, HOOK_PRINTF, HOOK_PROGRESS, HOOK_RESET, HookFunction, HookResult, ICinteractionHookHandle(), initHookResult(), logFlush(), Logger, logger, loggerSize(), logPrintf(), LoggerStruct::maxsize, netsearchHookHandle(), NULL, partialResultHookHandle(), progressHookHandle(), LoggerStruct::size, and tclHookHandle().

5.15.4.6 **TclResultType** ICinteractionHookHandle (**Hook** hook, va_list ap)

ICinteractionHookHandle: get another word from the IC textbox. Definition at line 512 of file hooker.c.

References HookStruct::cmd, hkInterp, hkResult, Hook, initHookResult(), NULL, TclResultType, and HookResultStruct::type.

Referenced by hooker_init().

5.15.4.7 **void** initHookResult (**Hook** hook) [static]

to be called by every hook. Definition at line 55 of file hooker.c.

References HookResultStruct::data, hkResult, HookResultStruct::hook, Hook, NULL, and HookResultStruct::type.

Referenced by evalHookHandle(), getsHookHandle(), glsInteractionHookHandle(), hooker_init(), ICinteractionHookHandle(), logFlush(), logWriteChar(), netsearchHookHandle(), partialResultHookHandle(), progressHookHandle(), resetHookHandle(), setHookCmd(), and tclHookHandle().

5.15.4.8 **TclResultType** logFlush ()

logFlush: flush the logwindow Definition at line 278 of file hooker.c.

References LoggerStruct::buffer, HookStruct::cmd, hkHooks, hkInterp, hkResult, Hook, HOOK_FLUSH, HOOK_PRINTF, initHookResult(), logger, NULL, LoggerStruct::size, TclResultType, and HookResultStruct::type.

Referenced by `hooker_init()`, and `logWriteChar()`.

5.15.4.9 `int loggerSize (int maxsize)`

set the buffersize of the logger.

Parameters:

maxsize the size which to set the log buffer to.

Returns:

the old size of the log buffer.

Definition at line 134 of file `hooker.c`.

References `LoggerStruct::buffer`, `logger`, `LoggerStruct::maxsize`, and `LoggerStruct::size`.

Referenced by `hooker_init()`.

5.15.4.10 `TclResultType logPrintf (char * format, va_list ap)`

`logPrintf`: print formatted text into the logger Definition at line 268 of file `hooker.c`.

References `hkHooks`, `hkResult`, `HookResultStruct::hook`, `HOOK_PRINTF`, `logWrite()`, and `TclResultType`.

Referenced by `hooker_init()`.

5.15.4.11 `TclResultType logWrite (char * text)`

`logWrite`: writes some text into the logger Definition at line 194 of file `hooker.c`.

References `HookResultStruct::data`, `hkResult`, `logWriteChar()`, `TclResultType`, and `HookResultStruct::type`.

Referenced by `logPrintf()`.

5.15.4.12 `TclResultType logWriteChar (char c)`

write a char to the logger. This functions writes a single char and check if it is a specail char which should be escaped. Definition at line 153 of file `hooker.c`.

References `LoggerStruct::buffer`, `HookResultStruct::data`, `hkResult`, `initHookResult()`, `logFlush()`, `logger`, `LoggerStruct::maxsize`, `NULL`, `LoggerStruct::size`, `TclResultType`, and `HookResultStruct::type`.

Referenced by `logWrite()`.

5.15.4.13 `TclResultType netsearchHookHandle (Hook hook, va_list ap)`

`netsearchHookHandle`: arg1: NetSearchState arg3: mode 1 - add rootnode 2 - solution node found 3 - add child 4 - add skipper 5 - widen searchspace 6 - closing open node Definition at line 319 of file `hooker.c`.

References `HookStruct::cmd`, `hkInterp`, `hkResult`, `Hook`, `initHookResult()`, `NULL`, `TclResultType`, and `HookResultStruct::type`.

Referenced by `hooker_init()`.

5.15.4.14 TclResultType partialResultHookHandle (Hook hook, va_list ap)

partialResultHookHandle: order displaying of another Parse Definition at line 482 of file hooker.c.

References HookStruct::cmd, hkInterp, hkResult, Hook, initHookResult(), NULL, TclResultType, and HookResultStruct::type.

Referenced by hooker_init().

5.15.4.15 TclResultType progressHookHandle (Hook hook, char *format, va_list ap)

progressHookHandle: show progress messages in tcl Definition at line 452 of file hooker.c.

References HookStruct::cmd, hkInterp, hkResult, Hook, initHookResult(), NULL, TclResultType, and HookResultStruct::type.

Referenced by hooker_init().

5.15.4.16 TclResultType resetHookHandle (Hook hook, va_list ap)

Call the Tcl-specified command, with no arguments. Definition at line 540 of file hooker.c.

References HookStruct::cmd, hkInterp, hkResult, Hook, initHookResult(), NULL, TclResultType, and HookResultStruct::type.

5.15.4.17 TclResultType setHookCmd (char *hookName, char *cmd)

setHookCmd: set the tcl-command of a named hook Definition at line 217 of file hooker.c.

References HookStruct::cmd, hkFindNoOfHook(), hkHooks, hkInterp, Hook, initHookResult(), NULL, and TclResultType.

5.15.4.18 TclResultType tclHookHandle (Hook hook, va_list ap)

tclHookHandle: default tclHook-Handler arguments aren't used Definition at line 568 of file hooker.c.

References HookStruct::cmd, hkInterp, hkResult, Hook, initHookResult(), NULL, TclResultType, and HookResultStruct::type.

Referenced by hooker_init().

5.15.5 Variable Documentation

5.15.5.1 Tcl_Interp* hkInterp

pointer to the currently used tcl interpreter.

Todo

This pointer isn't needed for newer swigs.

Definition at line 39 of file hooker.c.

Referenced by evalHookHandle(), getHookCmd(), getsHookHandle(), glsInteractionHookHandle(), hooker_init(), ICinteractionHookHandle(), logFlush(), netsearchHookHandle(), partialResultHookHandle(), progressHookHandle(), resetHookHandle(), setHookCmd(), and tclHookHandle().

5.15.5.2 Tcl_Interp* [hkInterp](#)

pointer to the currently used tcl interpreter.

Todo

This pointer isn't needed for newer swigs.

Definition at line 39 of file hooker.c.

Referenced by `evalHookHandle()`, `getHookCmd()`, `getHookHandle()`, `glsInteractionHookHandle()`, `hooker_init()`, `ICinteractionHookHandle()`, `logFlush()`, `netsearchHookHandle()`, `partialResultHookHandle()`, `progressHookHandle()`, `resetHookHandle()`, `setHookCmd()`, and `tclHookHandle()`.

5.15.5.3 [HookResult hkResult](#)

typed hook result. This pointer is used in our special typemaps. Definition at line 50 of file hooker.c.

Referenced by `evalHookHandle()`, `getHookCmd()`, `getHookHandle()`, `glsInteractionHookHandle()`, `hooker_init()`, `ICinteractionHookHandle()`, `initHookResult()`, `logFlush()`, `logPrintf()`, `logWrite()`, `logWriteChar()`, `netsearchHookHandle()`, `partialResultHookHandle()`, `progressHookHandle()`, `resetHookHandle()`, and `tclHookHandle()`.

5.15.5.4 [HookResult hkResult](#)

typed hook result. This pointer is used in our special typemaps. Definition at line 50 of file hooker.c.

Referenced by `evalHookHandle()`, `getHookCmd()`, `getHookHandle()`, `glsInteractionHookHandle()`, `hooker_init()`, `ICinteractionHookHandle()`, `initHookResult()`, `logFlush()`, `logPrintf()`, `logWrite()`, `logWriteChar()`, `netsearchHookHandle()`, `partialResultHookHandle()`, `progressHookHandle()`, `resetHookHandle()`, and `tclHookHandle()`.

5.15.5.5 [Logger logger](#) [static]

the singleton Logger instance used in this module. Definition at line 44 of file hooker.c.

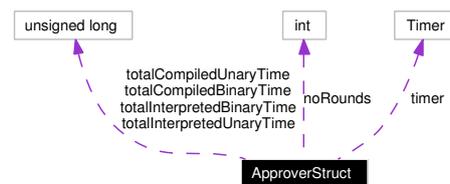
Referenced by `hooker_init()`, `logFlush()`, `loggerSize()`, and `logWriteChar()`.

Chapter 6

CDG Data Structure Documentation

6.1 ApproverStruct Struct Reference

Collaboration diagram for ApproverStruct:



6.1.1 Detailed Description

definition of a approver object.

Definition at line 81 of file compile.c.

Data Fields

- int `noRounds`
- Timer `timer`
- unsigned long `totalCompiledBinaryTime`
- unsigned long `totalCompiledUnaryTime`
- unsigned long `totalInterpretedBinaryTime`
- unsigned long `totalInterpretedUnaryTime`

6.1.2 Field Documentation

6.1.2.1 int `ApproverStruct::noRounds`

no of times to eval all lv pairs Definition at line 82 of file compile.c.

Referenced by `comCompareAllLvPairs()`, `comCompareWithContext()`, and `comNewApprover()`.

6.1.2.2 **Timer ApproverStruct::timer**

profiling Definition at line 83 of file compile.c.

Referenced by comApprove(), comCompareAllLvPairs(), comCompareWithContext(), comFreeApprover(), and comNewApprover().

6.1.2.3 **unsigned long ApproverStruct::totalCompiledBinaryTime**

profiling information Definition at line 87 of file compile.c.

Referenced by comApprove(), comCompareAllLvPairs(), comCompareWithContext(), and comNewApprover().

6.1.2.4 **unsigned long ApproverStruct::totalCompiledUnaryTime**

profiling information Definition at line 85 of file compile.c.

Referenced by comApprove(), and comNewApprover().

6.1.2.5 **unsigned long ApproverStruct::totalInterpretedBinaryTime**

profiling information Definition at line 86 of file compile.c.

Referenced by comApprove(), comCompareAllLvPairs(), comCompareWithContext(), and comNewApprover().

6.1.2.6 **unsigned long ApproverStruct::totalInterpretedUnaryTime**

profiling information Definition at line 84 of file compile.c.

Referenced by comApprove(), and comNewApprover().

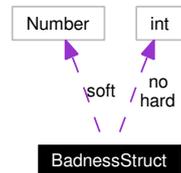
The documentation for this struct was generated from the following file:

- compile.c

6.2 BadnessStruct Struct Reference

```
#include <eval.h>
```

Collaboration diagram for BadnessStruct:



6.2.1 Detailed Description

Generalized score for comparing any two structures.

This structure counts both the total number of constraint violations (called ‘conflicts’ henceforth) and the number of hard conflicts (those with score 0.0). The field `soft` is the product of the penalties of all soft (non-zero) conflicts. Note that this structure allows different criteria for comparing two analyses; the function `bCompare()` is provided as one method, but others could be devised.

Definition at line 54 of file `eval.h`.

Data Fields

- int `hard`
- int `no`
- Number `soft`

6.2.2 Field Documentation

6.2.2.1 int `BadnessStruct::hard`

number of hard conflicts Definition at line 56 of file `eval.h`.

Referenced by `bAdd()`, `bAddBadness()`, `bClone()`, `bCompare()`, `bEqual()`, `bNew()`, `bPrint()`, `bSubtract()`, `bSubtractBadness()`, `comCompareAllLvPairs()`, and `comCompareWithContext()`.

6.2.2.2 int `BadnessStruct::no`

number of conflicts Definition at line 55 of file `eval.h`.

Referenced by `bAdd()`, `bAddBadness()`, `bClone()`, `bNew()`, `bPrint()`, `bSubtract()`, `bSubtractBadness()`, `comCompareAllLvPairs()`, and `comCompareWithContext()`.

6.2.2.3 Number `BadnessStruct::soft`

combined effect of soft conflicts Definition at line 57 of file `eval.h`.

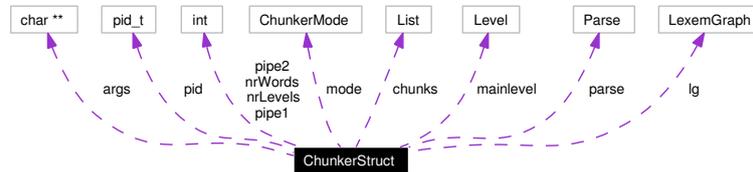
Referenced by `bAdd()`, `bAddBadness()`, `bClone()`, `bCompare()`, `bEqual()`, `bNew()`, `bPrint()`, `bSubtract()`, `bSubtractBadness()`, `comCompareAllLvPairs()`, and `comCompareWithContext()`.

The documentation for this struct was generated from the following file:

- eval.h

6.3 ChunkerStruct Struct Reference

Collaboration diagram for ChunkerStruct:



6.3.1 Detailed Description

representation of the chunker.

Definition at line 55 of file chunker.c.

Data Fields

- `char **` [args](#)
- `List` [chunks](#)
- `LexemGraph` [lg](#)
- `Level` [mainlevel](#)
- `ChunkerMode` [mode](#)
- `int` [nrLevels](#)
- `int` [nrWords](#)
- `Parse` [parse](#)
- `pid_t` [pid](#)
- `int` [pipe1](#) [2]
- `int` [pipe2](#) [2]

6.3.2 Field Documentation

6.3.2.1 `char**` `ChunkerStruct::args`

command to start this chunker

See also:

[chunkerArgs](#)

Definition at line 63 of file chunker.c.

Referenced by `chunkerNew()`, `initRealChunker()`, and `resetChunker()`.

6.3.2.2 `List` `ChunkerStruct::chunks`

found chunks Definition at line 62 of file chunker.c.

Referenced by `chunkerChunk()`, `chunkerNew()`, `evalChunker()`, and `resetChunker()`.

6.3.2.3 **LexemGraph ChunkerStruct::lg**

the current lexemgraph that we are chunking Definition at line 57 of file chunker.c.

Referenced by chunkerChunk(), chunkerNew(), getChunks(), initFakeChunker(), and resetChunker().

6.3.2.4 **Level ChunkerStruct::mainlevel**

mainlevels index in the current parse Definition at line 59 of file chunker.c.

Referenced by getFakeChunksAt(), initFakeChunker(), initRealChunker(), parseGetLabel(), parseGetLevelValue(), and parseGetModifiee().

6.3.2.5 **ChunkerMode ChunkerStruct::mode**

determines the mode of operation Definition at line 56 of file chunker.c.

Referenced by chunkerChunk(), chunkerNew(), and initChunker().

6.3.2.6 **int ChunkerStruct::nrLevels**

number of levels in the current parse Definition at line 61 of file chunker.c.

Referenced by chunkerNew(), initFakeChunker(), parseGetLabel(), parseGetModifiee(), and resetChunker().

6.3.2.7 **int ChunkerStruct::nrWords**

number of words in the current parse Definition at line 60 of file chunker.c.

Referenced by chunkerNew(), initFakeChunker(), parseGetRoots(), and resetChunker().

6.3.2.8 **Parse ChunkerStruct::parse**

parse generated from the annotation of the lattice Definition at line 58 of file chunker.c.

Referenced by chunkerNew(), getFakeChunksAt(), initFakeChunker(), parseGetLabel(), parseGetLevelValue(), parseGetModifiee(), and resetChunker().

6.3.2.9 **pid_t ChunkerStruct::pid**

process id of a started chunker job Definition at line 64 of file chunker.c.

Referenced by chunkerNew(), initRealChunker(), and resetChunker().

6.3.2.10 **int ChunkerStruct::pipe1[2]**

pipe connected to the stdin of the chunker process Definition at line 65 of file chunker.c.

Referenced by getChunks(), initRealChunker(), and resetChunker().

6.3.2.11 int `ChunkerStruct::pipe2`[2]

pipe connected to the stdout of the chunker process Definition at line 66 of file chunker.c.

Referenced by `getChunks()`, `initRealChunker()`, and `resetChunker()`.

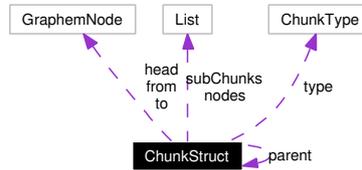
The documentation for this struct was generated from the following file:

- `chunker.c`

6.4 ChunkStruct Struct Reference

```
#include <chunker.h>
```

Collaboration diagram for ChunkStruct:



6.4.1 Detailed Description

internal representation of a chunk.

Definition at line 55 of file chunker.h.

Data Fields

- [GraphemNode](#) from
- [GraphemNode](#) head
- [List](#) nodes
- [ChunkStruct](#) * parent
- [List](#) subChunks
- [GraphemNode](#) to
- [ChunkType](#) type

6.4.2 Field Documentation

6.4.2.1 [GraphemNode ChunkStruct::from](#)

first element in the chunk Definition at line 58 of file chunker.h.

Referenced by [chunkerCloneChunk\(\)](#), [chunkerReplaceGraphemes\(\)](#), [cmpChunks\(\)](#), [compareChunks\(\)](#), [embedChunk\(\)](#), [evalChunker\(\)](#), [evalTerm\(\)](#), [findChunk\(\)](#), [getChunks\(\)](#), [getFakeChunksAt\(\)](#), [mergeChunk\(\)](#), [newChunk\(\)](#), and [postProcessChunks\(\)](#).

6.4.2.2 [GraphemNode ChunkStruct::head](#)

head of the chunk Definition at line 60 of file chunker.h.

Referenced by [chunkerCloneChunk\(\)](#), [chunkerReplaceGraphemes\(\)](#), [getChunks\(\)](#), [getFakeChunksAt\(\)](#), [newChunk\(\)](#), and [printChunk\(\)](#).

6.4.2.3 [List ChunkStruct::nodes](#)

list of all lexem nodes in the chunk Definition at line 57 of file chunker.h.

Referenced by `chunkerChunk()`, `chunkerChunkDelete()`, `chunkerCloneChunk()`, `chunkerReplaceGraphemes()`, `embedChunk()`, `getChunks()`, `getFakeChunksAt()`, `lgCopyTagScores()`, `mergeChunk()`, `newChunk()`, and `printChunk()`.

6.4.2.4 struct `ChunkStruct*` `ChunkStruct::parent`

direct dominating span (only used in fake-chunking Definition at line 61 of file `chunker.h`).

Referenced by `chunkerCloneChunk()`, `getFakeChunksAt()`, and `newChunk()`.

6.4.2.5 List `ChunkStruct::subChunks`

embedded chunks, e.g. [PC ... [NC ...]] Definition at line 62 of file `chunker.h`.

Referenced by `chunkerChunk()`, `chunkerChunkDelete()`, `chunkerCloneChunk()`, `chunkerReplaceGraphemes()`, `compareChunks()`, `countChunks()`, `embedChunk()`, `findChunk()`, `getChunks()`, `lgCopyTagScores()`, `mergeChunk()`, `newChunk()`, and `printChunk()`.

6.4.2.6 `GraphNode` `ChunkStruct::to`

last element in the chunk Definition at line 59 of file `chunker.h`.

Referenced by `chunkerCloneChunk()`, `chunkerReplaceGraphemes()`, `compareChunks()`, `embedChunk()`, `evalChunker()`, `evalTerm()`, `findChunk()`, `getChunks()`, `getFakeChunksAt()`, `mergeChunk()`, `newChunk()`, `postProcessChunks()`, and `printChunk()`.

6.4.2.7 `ChunkType` `ChunkStruct::type`

label of the chunk Definition at line 56 of file `chunker.h`.

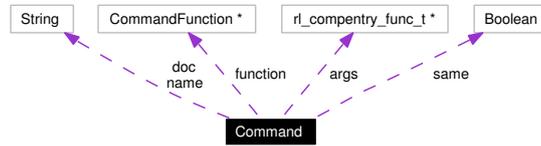
Referenced by `chunkerCloneChunk()`, `chunkerStringOfChunkType()`, `compareChunks()`, `countChunks()`, `evalChunker()`, `getFakeChunksAt()`, `newChunk()`, `postProcessChunks()`, and `printChunk()`.

The documentation for this struct was generated from the following file:

- `chunker.h`

6.5 Command Struct Reference

Collaboration diagram for Command:



6.5.1 Detailed Description

command definition for the [interface_commands](#).

Definition at line 166 of file command.c.

Data Fields

- `rl_comprentry_func_t * args` [MAXARGNO]
- `String doc`
- `CommandFunction * function`
- `String name`
- `Boolean same`

6.5.2 Field Documentation

6.5.2.1 `rl_comprentry_func_t* Command::args`[MAXARGNO]

array of completion functions for each argument Definition at line 170 of file command.c.

Referenced by `interface_completion()`.

6.5.2.2 `String Command::doc`

little help text Definition at line 177 of file command.c.

6.5.2.3 `CommandFunction* Command::function`

function that implements the command Definition at line 172 of file command.c.

Referenced by `commandEval()`.

6.5.2.4 `String Command::name`

the name of the command Definition at line 168 of file command.c.

Referenced by `cmdHelp()`, `command_completion_function()`, `commandEval()`, and `interface_completion()`.

6.5.2.5 Boolean `Command::same`

completion flag. This is set to 1 for those commands which use the same completion-function for all of their arguments Definition at line 174 of file `command.c`.

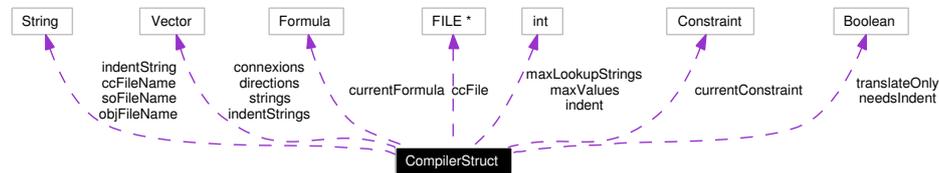
Referenced by `interface_completion()`.

The documentation for this struct was generated from the following file:

- `command.c`

6.6 CompilerStruct Struct Reference

Collaboration diagram for CompilerStruct:



6.6.1 Detailed Description

definition of a CDG compiler object.

Definition at line 58 of file compile.c.

Data Fields

- FILE * `ccFile`
- String `ccFileName`
- Vector `connexions`
- Constraint `currentConstraint`
- Formula `currentFormula`
- Vector `directions`
- int `indent`
- String `indentString`
- Vector `indentStrings`
- int `maxLookupStrings`
- int `maxValues`
- Boolean `needsIndent`
- String `objFileName`
- String `soFileName`
- Vector `strings`
- Boolean `translateOnly`

6.6.2 Field Documentation

6.6.2.1 FILE* `CompilerStruct::ccFile`

filehandle of the file in `ccFileName` Definition at line 60 of file compile.c.

Referenced by `comCompile()`, `comPrint()`, `comPrintln()`, `comWriteError()`, and `comWriteWarning()`.

6.6.2.2 String `CompilerStruct::ccFileName`

filename for C code Definition at line 59 of file compile.c.

Referenced by `comCompile()`, `comMake()`, `comNew()`, and `comTranslate()`.

6.6.2.3 Vector `CompilerStruct::connexions`

collection of all defined connexions Definition at line 68 of file compile.c.

Referenced by `comFree()`, and `comNew()`.

6.6.2.4 Constraint `CompilerStruct::currentConstraint`

the constraint being currently translated Definition at line 69 of file compile.c.

Referenced by `comIndexOfVarInfo()`, `comNew()`, `comTranslateConnected()`, `comTranslateConstraint()`, `comTranslateEquation()`, `comTranslateIs()`, `comTranslateLookup()`, `comTranslateMatch()`, `comTranslateParent()`, `comTranslateSubsumes()`, `comTranslateUnEquation()`, `comWriteError()`, and `comWriteWarning()`.

6.6.2.5 Formula `CompilerStruct::currentFormula`

the Formula being currently translated Definition at line 70 of file compile.c.

Referenced by `comTranslateFormula()`, and `comTranslateHas()`.

6.6.2.6 Vector `CompilerStruct::directions`

collection of all defined directions Definition at line 67 of file compile.c.

Referenced by `comFree()`, and `comNew()`.

6.6.2.7 int `CompilerStruct::indent`

current indentation level Definition at line 63 of file compile.c.

Referenced by `comIndent()`, `comNew()`, `comOutdent()`, `comTranslateBinaryConstraints()`, `comTranslateUnaryConstraints()`, and `comWriteFunctions()`.

6.6.2.8 String `CompilerStruct::indentString`

current indentation Definition at line 64 of file compile.c.

Referenced by `comIndent()`, `comNew()`, `comOutdent()`, `comPrint()`, and `comPrintln()`.

6.6.2.9 Vector `CompilerStruct::indentStrings`

possible indentation Definition at line 65 of file compile.c.

Referenced by `comFree()`, `comIndent()`, `comNew()`, and `comOutdent()`.

6.6.2.10 int `CompilerStruct::maxLookupStrings`

max number of Strings used in 'lookup' constraints Definition at line 72 of file compile.c.

Referenced by `comAnalyzeGrammar()`, and `comWriteInitFunction()`.

6.6.2.11 int [CompilerStruct::maxValues](#)

max number of variables to eval all constraints Definition at line 71 of file compile.c.

Referenced by `comAnalyzeGrammar()`, and `comWriteDeclarations()`.

6.6.2.12 Boolean [CompilerStruct::needsIndent](#)

TRUE by `comPrintln`, FALSE by `comPrint` Definition at line 66 of file compile.c.

Referenced by `comNew()`, `comPrint()`, `comPrintln()`, `comWriteError()`, and `comWriteWarning()`.

6.6.2.13 String [CompilerStruct::objFileName](#)

filename for object code Definition at line 61 of file compile.c.

Referenced by `comCompile()`, `comMake()`, and `comNew()`.

6.6.2.14 String [CompilerStruct::soFileName](#)

filename for C dll grammar Definition at line 62 of file compile.c.

Referenced by `comCompile()`, `comLoad()`, `comMake()`, and `comNew()`.

6.6.2.15 Vector [CompilerStruct::strings](#)

list of strings to allocated statically Definition at line 73 of file compile.c.

Referenced by `comFree()`, `comNew()`, `comRegisterString()`, `comWriteFinitFunction()`, and `comWriteInitFunction()`.

6.6.2.16 Boolean [CompilerStruct::translateOnly](#)

if TRUE we dont compile+load a shared object Definition at line 74 of file compile.c.

Referenced by `comCompile()`, and `comNew()`.

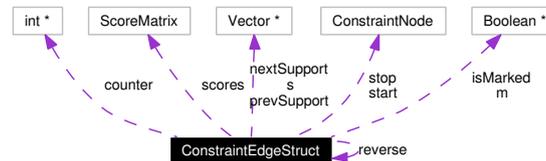
The documentation for this struct was generated from the following file:

- `compile.c`

6.7 ConstraintEdgeStruct Struct Reference

```
#include <constraintnet.h>
```

Collaboration diagram for ConstraintEdgeStruct:



6.7.1 Detailed Description

Models an edge between the two constraint nodes **start** and **stop**. These two fields are shallow copies of the nodes in the corresponding constraint net.

Definition at line 107 of file constraintnet.h.

Data Fields

- int * [counter](#)
- Boolean * [isMarked](#)
- Boolean * [m](#)
- Vector * [nextSupport](#)
- Vector * [prevSupport](#)
- [ConstraintEdgeStruct](#) * [reverse](#)
- Vector * [s](#)
- [ScoreMatrix](#) [scores](#)
- [ConstraintNode](#) [start](#)
- [ConstraintNode](#) [stop](#)

6.7.2 Field Documentation

6.7.2.1 int* [ConstraintEdgeStruct::counter](#)

counts support Definition at line 118 of file constraintnet.h.

6.7.2.2 Boolean* [ConstraintEdgeStruct::isMarked](#)

flag for marking, obsolete? Definition at line 119 of file constraintnet.h.

Referenced by [cnBuildEdges\(\)](#), and [cnDeleteEdge\(\)](#).

6.7.2.3 Boolean* [ConstraintEdgeStruct::m](#)

flag whether $\langle x, a \rangle$ is already processed Definition at line 114 of file constraintnet.h.

6.7.2.4 `Vector* ConstraintEdgeStruct::nextSupport`

used by module arconsistency Definition at line 116 of file constraintnet.h.

6.7.2.5 `Vector* ConstraintEdgeStruct::prevSupport`

used by module arconsistency Definition at line 115 of file constraintnet.h.

6.7.2.6 `struct ConstraintEdgeStruct* ConstraintEdgeStruct::reverse`

points to the inverse of the edge itself. Definition at line 110 of file constraintnet.h.

Referenced by `cnBuildEdges()`.

6.7.2.7 `Vector* ConstraintEdgeStruct::s`

used by module arconsistency Definition at line 117 of file constraintnet.h.

6.7.2.8 `ScoreMatrix ConstraintEdgeStruct::scores`

holds all binary scores calculated for pairs of LVs from the two nodes Definition at line 112 of file constraintnet.h.

Referenced by `cnBuildEdges()`, `cnDeleteEdge()`, and `cnPrintEdge()`.

6.7.2.9 `ConstraintNode ConstraintEdgeStruct::start`

start node Definition at line 108 of file constraintnet.h.

Referenced by `cmdEdges()`, `cnBuildEdges()`, and `cnPrintEdge()`.

6.7.2.10 `ConstraintNode ConstraintEdgeStruct::stop`

end node Definition at line 109 of file constraintnet.h.

Referenced by `cmdEdges()`, `cnBuildEdges()`, and `cnPrintEdge()`.

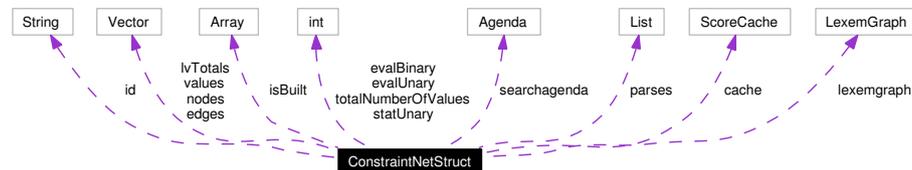
The documentation for this struct was generated from the following file:

- constraintnet.h

6.8 ConstraintNetStruct Struct Reference

```
#include <constraintnet.h>
```

Collaboration diagram for ConstraintNetStruct:



6.8.1 Detailed Description

The constraint net.

Definition at line 44 of file `constraintnet.h`.

Data Fields

- [ScoreCache](#) `cache`
- [Vector](#) `edges`
- [int](#) `evalBinary`
- [int](#) `evalUnary`
- [String](#) `id`
- [Array](#) `isBuilt`
- [LexemGraph](#) `lexemgraph`
- [Vector](#) `lvTotals`
- [Vector](#) `nodes`
- [List](#) `parses`
- [Agenda](#) `searchagenda`
- [int](#) `statUnary`
- [int](#) `totalNumberOfValues`
- [Vector](#) `values`

6.8.2 Field Documentation

6.8.2.1 [ScoreCache](#) `ConstraintNetStruct::cache`

Holds the cache used to hold the results of binary constraint evaluations. (This cache only exists if `scUseCache` was set when creating the net.) Definition at line 62 of file `constraintnet.h`.

Referenced by `cnBuildFinal()`, `cnBuildInit()`, `cnDelete()`, `cnPrintInfo()`, `cnRenew()`, `comCompareAllLvPairs()`, `comCompareWithContext()`, and `evalBinary()`.

6.8.2.2 [Vector](#) `ConstraintNetStruct::edges`

Vector of `ConstraintEdge` Definition at line 53 of file `constraintnet.h`.

Referenced by `cmdEdges()`, `cnBuildEdges()`, `cnBuildFinal()`, `cnBuildInit()`, `cnDelete()`, `cnPrint()`, and `cnPrintInfo()`.

6.8.2.3 int `ConstraintNetStruct::evalBinary`

Counting binary evaluations Definition at line 68 of file constraintnet.h.

Referenced by `cnBuildInit()`, `cnPrintInfo()`, `evalBinaryConstraint()`, and `evalConstraint()`.

6.8.2.4 int `ConstraintNetStruct::evalUnary`

Counting unary evaluations Definition at line 66 of file constraintnet.h.

Referenced by `cnBuildInit()`, `cnPrintInfo()`, `evalConstraint()`, and `evalUnaryConstraint()`.

6.8.2.5 String `ConstraintNetStruct::id`

A unique identifier for the net. Each net created by the system is labeled as net n , where n is the current value of `cnCounter`. Definition at line 45 of file constraintnet.h.

Referenced by `cmdEdges()`, `cmdNetsearch()`, `cmdNewnet()`, `cmdWriteAnno()`, `cnBuildInit()`, `cnDelete()`, `cnPrint()`, `cnPrintInfo()`, and `comApprove()`.

6.8.2.6 Array `ConstraintNetStruct::isBuilt`

have LVs for word x ->word y been built? Definition at line 54 of file constraintnet.h.

Referenced by `cnBuildFinal()`, `cnBuildInit()`, `cnBuildNodes()`, and `cnBuildTriple()`.

6.8.2.7 `LexemGraph` `ConstraintNetStruct::lexemgraph`

Points to the enriched word graph used in constructing the net. Definition at line 49 of file constraintnet.h.

Referenced by `cmdDistance()`, `cnBuildEdges()`, `cnBuildFinal()`, `cnBuildIter()`, `cnBuildLevelValues()`, `cnBuildNodes()`, `cnBuildTriple()`, `cnDelete()`, `cnGetGraphemNodeFromArc()`, `cnGetLattice()`, `cnOptimizeNode()`, `cnPrint()`, `cnPrintActiveLVs()`, `cnPrintInfo()`, `cnRenew()`, and `cnTag()`.

6.8.2.8 Vector `ConstraintNetStruct::lvTotals`

Records how many LVs the net contained at each successive step in its history. This is only used by incrementalcompletion. Definition at line 69 of file constraintnet.h.

Referenced by `cnBuildInit()`, and `cnDelete()`.

6.8.2.9 Vector `ConstraintNetStruct::nodes`

Vector of `ConstraintNode` Definition at line 52 of file constraintnet.h.

Referenced by `cmdEdges()`, `cnBuildEdges()`, `cnBuildFinal()`, `cnBuildInit()`, `cnBuildIter()`, `cnBuildNodes()`, `cnBuildTriple()`, `cnBuildUpdateArcs()`, `cnDelete()`, `cnFindNode()`, `cnOptimizeNet()`, `cnPrint()`, `cnPrintActiveLVs()`, `cnPrintInfo()`, `cnRenew()`, `cnSortLVs()`, `cnSortNodes()`, and `comCompareNets()`.

6.8.2.10 List `ConstraintNetStruct::parses`

Contains all structures of type `Parse` found by any solution method. Definition at line 60 of file constraintnet.h.

Referenced by `cmdNetsearch()`, `cmdPrintParses()`, `cmdWriteAnno()`, `cmdWriteParses()`, `cnBuildInit()`, `cnDelete()`, `cnPrint()`, `cnPrintParses()`, and `cnRenew()`.

6.8.2.11 Agenda `ConstraintNetStruct::searchagenda`

Agenda for searching, it is used by `netsearch()` Definition at line 55 of file `constraintnet.h`.

Referenced by `cnBuildInit()`, and `cnDelete()`.

6.8.2.12 int `ConstraintNetStruct::statUnary`

Counting unary statistics Definition at line 67 of file `constraintnet.h`.

Referenced by `cnBuildInit()`, and `cnPrintInfo()`.

6.8.2.13 int `ConstraintNetStruct::totalNumberOfValues`

Holds the total number of LVs in the constraint net. It should always be equal to `vectorSize(net->values)`. Definition at line 57 of file `constraintnet.h`.

Referenced by `cnBuildInit()`, and `cnBuildLv()`.

6.8.2.14 Vector `ConstraintNetStruct::values`

Vector of LevelValues Definition at line 51 of file `constraintnet.h`.

Referenced by `cnBuildFinal()`, `cnBuildInit()`, `cnBuildLv()`, `cnBuildNodes()`, `cnDelete()`, `cnDeleteAllLVs()`, `cnPrintActiveLVs()`, `cnRenew()`, `cnUndeleteAllLVs()`, `comCompareAllLvPairs()`, `comCompareAllLVs()`, `comCompareNets()`, `comCompareWithContext()`, `comFindComparableLv()`, and `countValidValues()`.

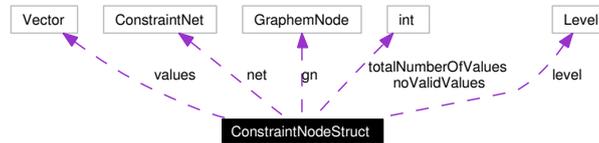
The documentation for this struct was generated from the following file:

- `constraintnet.h`

6.9 ConstraintNodeStruct Struct Reference

```
#include <constraintnet.h>
```

Collaboration diagram for ConstraintNodeStruct:



6.9.1 Detailed Description

Models a node in a constraint net

Definition at line 84 of file constraintnet.h.

Data Fields

- [GraphemNode](#) gn
- [Level](#) level
- [ConstraintNet](#) net
- [int](#) noValidValues
- [int](#) totalNumberOfValues
- [Vector](#) values

6.9.2 Field Documentation

6.9.2.1 [GraphemNode ConstraintNodeStruct::gn](#)

corresponding grapheme node Definition at line 88 of file constraintnet.h.

Referenced by [cnBuildIter\(\)](#), [cnBuildTriple\(\)](#), [cnBuildUpdateArcs\(\)](#), [cnConnectedByArc\(\)](#), [cnIsEndNode\(\)](#), [cnIsStartNode\(\)](#), [cnOptimizeNode\(\)](#), [cnPrint\(\)](#), [cnPrintEdge\(\)](#), and [cnPrintNode\(\)](#).

6.9.2.2 [Level ConstraintNodeStruct::level](#)

points to the level for which a constraint node was built. Definition at line 86 of file constraintnet.h.

Referenced by [cnBuildEdges\(\)](#), [cnBuildIter\(\)](#), [cnBuildTriple\(\)](#), [cnBuildUpdateArcs\(\)](#), [cnConnectedByArc\(\)](#), [cnIsEndNode\(\)](#), [cnIsStartNode\(\)](#), [cnNodeComparePrio\(\)](#), [cnOptimizeNode\(\)](#), [cnPrint\(\)](#), [cnPrintEdge\(\)](#), and [cnPrintNode\(\)](#).

6.9.2.3 [ConstraintNet ConstraintNodeStruct::net](#)

corresponding constraint net Definition at line 85 of file constraintnet.h.

Referenced by [cnBuildIter\(\)](#), [cnBuildLevelValues\(\)](#), and [cnBuildLv\(\)](#).

6.9.2.4 int `ConstraintNodeStruct::noValidValues`

number of not yet deleted LVs Definition at line 93 of file constraintnet.h.

Referenced by `cnBuildIter()`, `cnBuildTriple()`, `cnBuildUpdateArcs()`, `cnNodeCompareSmallest()`, `cnOptimizeNode()`, `cnPrint()`, `cnPrintInfo()`, `cnRenew()`, `cnSortNodes()`, and `cnUnaryPruning()`.

6.9.2.5 int `ConstraintNodeStruct::totalNumberOfValues`

number of level values Definition at line 92 of file constraintnet.h.

Referenced by `cnBuildIter()`, `cnBuildLv()`, `cnBuildTriple()`, and `cnBuildUpdateArcs()`.

6.9.2.6 Vector `ConstraintNodeStruct::values`

holds shallow copies of all LVs that may be used to bind this constraint node. Definition at line 89 of file constraintnet.h.

Referenced by `cnBuildEdges()`, `cnBuildIter()`, `cnBuildLv()`, `cnBuildTriple()`, `cnBuildUpdateArcs()`, `cnDeleteNode()`, `cnFindNode()`, `cnOptimizeNode()`, `cnPrint()`, `cnPrintActiveLVs()`, `cnPrintEdge()`, `cnRenew()`, `cnSortLVs()`, `cnUnaryPruning()`, and `comCompareNets()`.

The documentation for this struct was generated from the following file:

- constraintnet.h

6.10 ConstraintViolationStruct Struct Reference

```
#include <constraintnet.h>
```

Collaboration diagram for ConstraintViolationStruct:



6.10.1 Detailed Description

Holds information about a constraint violated by a solution.

Definition at line 145 of file constraintnet.h.

Data Fields

- Constraint [constraint](#)
- LevelValue [lv1](#)
- LevelValue [lv2](#)
- int [nodeBindingIndex1](#)
- int [nodeBindingIndex2](#)
- Number [penalty](#)

6.10.2 Field Documentation

6.10.2.1 Constraint [ConstraintViolationStruct::constraint](#)

violated constraint Definition at line 146 of file constraintnet.h.

Referenced by [cnCompareViolation\(\)](#), [comCompareAllLvPairs\(\)](#), [comCompareWithContext\(\)](#), [cvAnalyse\(\)](#), [cvClone\(\)](#), [cvCompare\(\)](#), [cvCompareNatural\(\)](#), [cvContains\(\)](#), [cvNew\(\)](#), and [cvPrint\(\)](#).

6.10.2.2 LevelValue [ConstraintViolationStruct::lv1](#)

first node binding Definition at line 155 of file constraintnet.h.

Referenced by [comCompareAllLvPairs\(\)](#), [comCompareWithContext\(\)](#), [cvAnalyse\(\)](#), [cvClone\(\)](#), [cvCompare\(\)](#), [cvDelete\(\)](#), and [cvNew\(\)](#).

6.10.2.3 LevelValue [ConstraintViolationStruct::lv2](#)

second node binding, maybe empty Definition at line 159 of file constraintnet.h.

Referenced by [comCompareAllLvPairs\(\)](#), [comCompareWithContext\(\)](#), [cvAnalyse\(\)](#), [cvClone\(\)](#), [cvCompare\(\)](#), [cvDelete\(\)](#), and [cvNew\(\)](#).

6.10.2.4 int [ConstraintViolationStruct::nodeBindingIndex1](#)

holds the position of the LV causing the conflict, as calculated by `lvIndex()`. Definition at line 152 of file `constraintnet.h`.

Referenced by `cnCompareViolation()`, `cvCompareNatural()`, `cvContains()`, `cvNew()`, and `cvPrint()`.

6.10.2.5 int [ConstraintViolationStruct::nodeBindingIndex2](#)

holds the position of the LV causing the conflict, as calculated by `lvIndex()`. Definition at line 156 of file `constraintnet.h`.

Referenced by `cnCompareViolation()`, `cvCompareNatural()`, `cvContains()`, `cvNew()`, and `cvPrint()`.

6.10.2.6 Number [ConstraintViolationStruct::penalty](#)

holds the penalty of this particular instance of **constraint** (recall that a constraint declaration specifies an entire set of constraints, possibly with variable penalties). Definition at line 147 of file `constraintnet.h`.

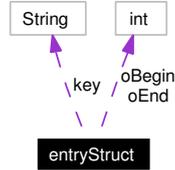
Referenced by `cnCompareViolation()`, `comCompareAllLvPairs()`, `comCompareWithContext()`, `cvClone()`, `cvCompare()`, `cvNew()`, and `cvPrint()`.

The documentation for this struct was generated from the following file:

- `constraintnet.h`

6.11 entryStruct Struct Reference

Collaboration diagram for entryStruct:



6.11.1 Detailed Description

a database entry. This is an entry for the database which holds a key and the byte position of its lexical entry in the underlying cdg file.

Definition at line 78 of file cdgdb.c.

Data Fields

- String [key](#)
- int [oBegin](#)
- int [oEnd](#)

6.11.2 Field Documentation

6.11.2.1 String [entryStruct::key](#)

the key of the lexical entry. Definition at line 79 of file cdgdb.c.

Referenced by [dbGetEntries\(\)](#), and [newDbEntry\(\)](#).

6.11.2.2 int [entryStruct::oBegin](#)

offset of its Beginning in the file Definition at line 80 of file cdgdb.c.

Referenced by [dbGetEntries\(\)](#), [dbLoadEntries\(\)](#), and [newDbEntry\(\)](#).

6.11.2.3 int [entryStruct::oEnd](#)

offset of its End in the file Definition at line 81 of file cdgdb.c.

Referenced by [dbGetEntries\(\)](#), [dbLoadEntries\(\)](#), and [newDbEntry\(\)](#).

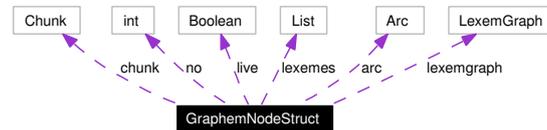
The documentation for this struct was generated from the following file:

- [cdgdb.c](#)

6.12 GraphemNodeStruct Struct Reference

```
#include <lexemgraph.h>
```

Collaboration diagram for GraphemNodeStruct:



6.12.1 Detailed Description

A grapheme node represents the hypothesis of a specific phonetic form for a specific time interval. Thus, it corresponds biuniquely to an Arc in the underlying Lattice. However, it also holds information about the state of processing by a particular grammar.

The field *no* is an index into the field *graphemnodes* of the enclosing lexeme graph.

The field *lexemgraph* points to this graph.

The field *arcpoints* to the corresponding Arc.

The field *lexicalEntries* contains a List of all known lexical entries with the same phonetic form as the word on the Arc.

The field *lexemes* contains all lexeme nodes built from the elements of *lexicalEntries*.

The field *ambiguity* counts how many of these lexeme nodes are currently undeleted.

Definition at line 113 of file *lexemgraph.h*.

Data Fields

- Arc [arc](#)
- Chunk [chunk](#)
- List [lexemes](#)
- LexemGraph [lexemgraph](#)
- Boolean [live](#)
- int [no](#)

6.12.2 Field Documentation

6.12.2.1 Arc [GraphemNodeStruct::arc](#)

arc in word lattice Definition at line 116 of file *lexemgraph.h*.

Referenced by [cmpChunks\(\)](#), [cmpGraphemes\(\)](#), [cnBuildLevelValues\(\)](#), [cnBuildTriple\(\)](#), [cnGetGraphemNodeFromArc\(\)](#), [cnOptimizeNode\(\)](#), [cnPrintEdge\(\)](#), [cnPrintNode\(\)](#), [compareChunks\(\)](#), [computeNoOfPathsFromStart\(\)](#), [computeNoOfPathsToEnd\(\)](#), [embedChunk\(\)](#), [evalChunker\(\)](#), [evalTerm\(\)](#), [findChunk\(\)](#), [findGrapheme\(\)](#), [getChunks\(\)](#), [getFakeChunksAt\(\)](#), [gnClone\(\)](#), [lgComputeDistances\(\)](#), [lgComputeNoOfPaths\(\)](#), [lgContains\(\)](#), [lgIsEndNode\(\)](#), [lgIsStartNode\(\)](#), [lgNewIter\(\)](#), [mergeChunk\(\)](#), [postProcessChunks\(\)](#), and [printChunk\(\)](#).

6.12.2.2 **Chunk** `GraphemNodeStruct::chunk`

to which chunk do we belong Definition at line 119 of file lexemgraph.h.

Referenced by `chunkerChunk()`, `evalTerm()`, `gnClone()`, `lgCopyTagScores()`, and `lgNewIter()`.

6.12.2.3 **List** `GraphemNodeStruct::lexemes`

list of disambiguated lexeme nodes Definition at line 117 of file lexemgraph.h.

Referenced by `cnBuildIter()`, `cnBuildNodes()`, `cnBuildUpdateArcs()`, `cnOptimizeNode()`, `cnPrint()`, `cnPrintNode()`, `getCategories()`, `gnClone()`, `lgAreDeletableNodes()`, `lgComputeDistances()`, `lgComputeNoOfPaths()`, `lgDelete()`, `lgMayModify()`, `lgMostProbablePath()`, `lgNewIter()`, `lgPartitions()`, and `lgQuery-Cat()`.

6.12.2.4 **LexemGraph** `GraphemNodeStruct::lexemgraph`

pointer back to the lexem graph Definition at line 115 of file lexemgraph.h.

Referenced by `getCategories()`, `gnClone()`, `lgIsEndNode()`, `lgIsStartNode()`, and `lgNewIter()`.

6.12.2.5 **Boolean** `GraphemNodeStruct::live`

are there any lexemes left undeleted? Definition at line 118 of file lexemgraph.h.

Referenced by `computeNoOfPathsFromStart()`, `computeNoOfPathsToEnd()`, and `lgComputeNoOfPaths()`.

6.12.2.6 **int** `GraphemNodeStruct::no`

index in `lg->graphemnodes` Definition at line 114 of file lexemgraph.h.

Referenced by `cnOptimizeNode()`, `computeNoOfPathsFromStart()`, `computeNoOfPathsToEnd()`, `gnClone()`, `lgAreDeletableNodes()`, `lgClone()`, `lgComputeNoOfPaths()`, `lgDistanceOfNodes()`, and `lgNewIter()`.

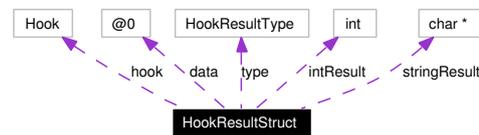
The documentation for this struct was generated from the following file:

- `lexemgraph.h`

6.13 HookResultStruct Struct Reference

```
#include <hooker.h>
```

Collaboration diagram for HookResultStruct:



6.13.1 Detailed Description

typed result of a hook. This structure tracks the result of a hook execution. The return value stored within is used in our special typemaps for the swig interface generator.

Definition at line 65 of file hooker.h.

Data Fields

- union {
 - int [intResult](#)
 - char * [stringResult](#)
 } [data](#)
- [Hook hook](#)
- [HookResultType type](#)

6.13.2 Field Documentation

6.13.2.1 union { ... } [HookResultStruct::data](#)

result data of different type

Referenced by [getHookCmd\(\)](#), [initHookResult\(\)](#), [logWrite\(\)](#), and [logWriteChar\(\)](#).

6.13.2.2 [Hook HookResultStruct::hook](#)

the hook that produced this result Definition at line 71 of file hooker.h.

Referenced by [getHookCmd\(\)](#), [initHookResult\(\)](#), and [logPrintf\(\)](#).

6.13.2.3 int [HookResultStruct::intResult](#)

integer result value Definition at line 68 of file hooker.h.

6.13.2.4 char* [HookResultStruct::stringResult](#)

string result value Definition at line 69 of file hooker.h.

6.13.2.5 HookResultType HookResultStruct::type

type of the result data Definition at line 66 of file hooker.h.

Referenced by evalHookHandle(), getHookCmd(), getsHookHandle(), glsInteractionHookHandle(), ICinteractionHookHandle(), initHookResult(), logFlush(), logWrite(), logWriteChar(), netsearchHookHandle(), partialResultHookHandle(), progressHookHandle(), resetHookHandle(), and tclHookHandle().

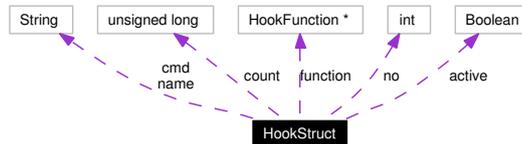
The documentation for this struct was generated from the following file:

- hooker.h

6.14 HookStruct Struct Reference

```
#include <hook.h>
```

Collaboration diagram for HookStruct:



6.14.1 Detailed Description

callback information. This structure bundles the state of a callback.

Definition at line 207 of file hook.h.

Data Fields

- Boolean [active](#)
- String [cmd](#)
- unsigned long [count](#)
- [HookFunction](#) * [function](#)
- String [name](#)
- int [no](#)

6.14.2 Field Documentation

6.14.2.1 Boolean [HookStruct::active](#)

flag indicating the status of the hook Definition at line 212 of file hook.h.

Referenced by [cdgExecHook\(\)](#), [cmdHook\(\)](#), and [hkInitialize\(\)](#).

6.14.2.2 String [HookStruct::cmd](#)

command to be executed in tcl Definition at line 211 of file hook.h.

Referenced by [evalHookHandle\(\)](#), [getHookCmd\(\)](#), [getHookHandle\(\)](#), [glsInteractionHookHandle\(\)](#), [hkInitialize\(\)](#), [hooker_init\(\)](#), [ICinteractionHookHandle\(\)](#), [logFlush\(\)](#), [netsearchHookHandle\(\)](#), [partialResultHookHandle\(\)](#), [progressHookHandle\(\)](#), [resetHookHandle\(\)](#), [setHookCmd\(\)](#), and [tclHookHandle\(\)](#).

6.14.2.3 unsigned long [HookStruct::count](#)

number of past invocations Definition at line 209 of file hook.h.

Referenced by [cdgExecHook\(\)](#), [cdgFlush\(\)](#), [cdgGetString\(\)](#), [cdgPrintf\(\)](#), [cmdHook\(\)](#), and [hkInitialize\(\)](#).

6.14.2.4 HookFunction* HookStruct::function

function which is called with a va_list argument Definition at line 213 of file hook.h.

Referenced by cdgExecHook(), cdgFlush(), cdgGetString(), cdgPrintf(), hkInitialize(), and hooker_init().

6.14.2.5 String HookStruct::name

identifier required to be unique among all hooks Definition at line 210 of file hook.h.

Referenced by cdgExecHook(), cmdHook(), hkFindNoOfHook(), and hkInitialize().

6.14.2.6 int HookStruct::no

index in vector hkHooks Definition at line 208 of file hook.h.

Referenced by hkInitialize().

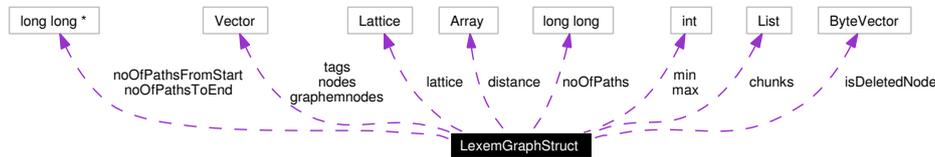
The documentation for this struct was generated from the following file:

- hook.h

6.15 LexemGraphStruct Struct Reference

```
#include <lexemgraph.h>
```

Collaboration diagram for LexemGraphStruct:



6.15.1 Detailed Description

A LexemGraph is a word graph enriched with lexical information. In particular, it contains several lexemes for each arc of the Lattice that is lexically ambiguous. Each of these pairs of lexical entry and time span is called a lexeme node. The field *lattice* points to the underlying word graph.

The Vectors *graphemnodes* and *nodes* contains all grapheme nodes and lexeme nodes. Grapheme nodes are an intermediary data structure between word arcs and lexeme nodes that is not strictly necessary.

The fields *min* and *max* correspond to the fields with the same names in the Lattice.

The field *distance* holds an Array of the distance between any two lexeme nodes. The distance is measured in words, hence any two adjacent lexeme nodes have distance~1. The distance array is also used to check whether two lexeme nodes are compatible with each other, i.e. whether there is a path through the lexeme graph that includes them both.

The fields *noOfPathsFromStart* and *noOfPathsToEnd* hold arrays that map each grapheme node to the number of complete paths from the start or to the end of the entire graph that go through it. This is used to determine whether a lexeme node can be deleted or not.

The field *noOfPaths* holds the total number of paths from start to end possible in the word graph. This should reflect the state of the Vector *isDeletedNode*.

WARNING: The last three fields use GNU's long long int type as a cheap way to get 64-bit integers because the number of paths in realistic word graphs really does need that. However, the tool SWIG cannot deal with this type, and as a result XCDG cannot access the LexemGraph structure at all. Several functions in this library exist only to work around this restriction. Ultimately these fields should be converted to a proper Bigint type.

The Vector *isDeletedNode* marks those lexeme nodes that have been deleted and should be ignored. For instance, when an LV is added to a partial solution, all lexeme nodes that are not `lgCompatibleNodes()` to its lexeme nodes should be marked as deleted.

Definition at line 72 of file lexemgraph.h.

Data Fields

- List [chunks](#)
- Array [distance](#)
- Vector [graphemnodes](#)
- ByteVector [isDeletedNode](#)
- Lattice [lattice](#)
- int [max](#)

- int [min](#)
- Vector [nodes](#)
- long long [noOfPaths](#)
- long long * [noOfPathsFromStart](#)
- long long * [noOfPathsToEnd](#)
- Vector [tags](#)

6.15.2 Field Documentation

6.15.2.1 List [LexemGraphStruct::chunks](#)

set of all chunks of the lattice Definition at line 87 of file lexemgraph.h.

Referenced by [chunkerChunk\(\)](#), [cnPrint\(\)](#), [lgClone\(\)](#), [lgCopyTagScores\(\)](#), [lgDelete\(\)](#), [lgNewInit\(\)](#), and [lgPrint\(\)](#).

6.15.2.2 Array [LexemGraphStruct::distance](#)

matrix of distances Definition at line 81 of file lexemgraph.h.

Referenced by [lgClone\(\)](#), [lgComputeDistances\(\)](#), [lgDelete\(\)](#), [lgDistanceOfNodes\(\)](#), [lgNewInit\(\)](#), and [lgUpdateArcs\(\)](#).

6.15.2.3 Vector [LexemGraphStruct::graphemnodes](#)

vector of grapheme nodes Definition at line 74 of file lexemgraph.h.

Referenced by [cmdDistance\(\)](#), [cnBuildIter\(\)](#), [cnBuildNodes\(\)](#), [cnBuildTriple\(\)](#), [cnGetGraphemNodeFromArc\(\)](#), [cnTag\(\)](#), [computeNoOfPathsFromStart\(\)](#), [computeNoOfPathsToEnd\(\)](#), [findGrapheme\(\)](#), [getChunks\(\)](#), [lgClone\(\)](#), [lgComputeDistances\(\)](#), [lgComputeNoOfPaths\(\)](#), [lgContains\(\)](#), [lgDelete\(\)](#), [lgMostProbablePath\(\)](#), [lgNewFinal\(\)](#), [lgNewInit\(\)](#), and [lgNewIter\(\)](#).

6.15.2.4 ByteVector [LexemGraphStruct::isDeletedNode](#)

vector of boolean flags Definition at line 85 of file lexemgraph.h.

Referenced by [cnPrint\(\)](#), [cnPrintActiveLVs\(\)](#), [cnRenew\(\)](#), [getCategories\(\)](#), [lgAreDeletableNodes\(\)](#), [lgClone\(\)](#), [lgComputeNoOfPaths\(\)](#), [lgCopySelection\(\)](#), [lgDelete\(\)](#), [lgDeleteNode\(\)](#), [lgDeleteNodes\(\)](#), [lgIsDeletedNode\(\)](#), [lgNewFinal\(\)](#), [lgNewInit\(\)](#), [lgNewIter\(\)](#), [lgPrint\(\)](#), and [lgPrintNode\(\)](#).

6.15.2.5 Lattice [LexemGraphStruct::lattice](#)

underlying word graph Definition at line 73 of file lexemgraph.h.

Referenced by [cnGetLattice\(\)](#), [cnPrintInfo\(\)](#), [initFakeChunker\(\)](#), [lgClone\(\)](#), [lgMostProbablePath\(\)](#), [lgNew\(\)](#), [lgNewFinal\(\)](#), [lgNewInit\(\)](#), [lgPrint\(\)](#), [lgSpuriousUppercase\(\)](#), and [lgUpdateArcs\(\)](#).

6.15.2.6 int [LexemGraphStruct::max](#)

maximum end position Definition at line 77 of file lexemgraph.h.

Referenced by `cnBuildFinal()`, `cnBuildNodes()`, `cnBuildTriple()`, `cnTag()`, `computeNoOfPathsToEnd()`, `evalTerm()`, `lgClone()`, `lgIsEndNode()`, `lgMakePath()`, `lgNewFinal()`, `lgNewInit()`, `lgNewIter()`, and `lgWidth()`.

6.15.2.7 int `LexemGraphStruct::min`

minimum start position Definition at line 76 of file `lexemgraph.h`.

Referenced by `cnBuildFinal()`, `cnTag()`, `computeNoOfPathsFromStart()`, `lgClone()`, `lgComputeNoOfPaths()`, `lgIsStartNode()`, `lgMakePath()`, `lgNewInit()`, and `lgNewIter()`.

6.15.2.8 Vector `LexemGraphStruct::nodes`

vector of lexeme nodes Definition at line 75 of file `lexemgraph.h`.

Referenced by `cnBuildEdges()`, `cnBuildIter()`, `cnBuildNodes()`, `cnOptimizeNode()`, `cnRenew()`, `cnTag()`, `lgClone()`, `lgComputeNoOfPaths()`, `lgCopySelection()`, `lgCopyTagScores()`, `lgDelete()`, `lgMakePath()`, `lgNewInit()`, `lgNewIter()`, `lgPrint()`, `lgRequireLexeme()`, `lgRequireLexemes()`, and `lgWidth()`.

6.15.2.9 long long `LexemGraphStruct::noOfPaths`

total number of paths through graph Definition at line 84 of file `lexemgraph.h`.

Referenced by `cnPrint()`, `cnTag()`, `lgAreDeletableNodes()`, `lgComputeNoOfPaths()`, `lgDeleteNode()`, `lgDeleteNodes()`, and `lgNewFinal()`.

6.15.2.10 long long* `LexemGraphStruct::noOfPathsFromStart`

vector of numbers of paths Definition at line 82 of file `lexemgraph.h`.

Referenced by `cnOptimizeNode()`, `computeNoOfPathsFromStart()`, `lgAreDeletableNodes()`, `lgClone()`, `lgComputeNoOfPaths()`, `lgDelete()`, and `lgNewInit()`.

6.15.2.11 long long* `LexemGraphStruct::noOfPathsToEnd`

vector of numbers of paths Definition at line 83 of file `lexemgraph.h`.

Referenced by `cnOptimizeNode()`, `computeNoOfPathsToEnd()`, `lgAreDeletableNodes()`, `lgClone()`, `lgComputeNoOfPaths()`, `lgDelete()`, and `lgNewInit()`.

6.15.2.12 Vector `LexemGraphStruct::tags`

set of POS tags Definition at line 86 of file `lexemgraph.h`.

Referenced by `lgCopyTagScores()`, `lgDelete()`, `lgNewFinal()`, and `lgNewInit()`.

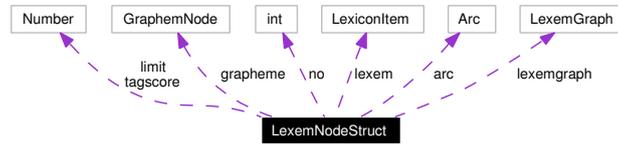
The documentation for this struct was generated from the following file:

- `lexemgraph.h`

6.16 LexemNodeStruct Struct Reference

```
#include <lexemgraph.h>
```

Collaboration diagram for LexemNodeStruct:



6.16.1 Detailed Description

A LexemNode represents the hypothesis of a specific lexical variant for a specific time interval.

The field *no* is the index of the lexeme node in the field *nodes* of the enclosing lexeme graph.

The field *lexemgraph* point to the enclosing lexeme graph.

The field *arc* points to the underlying Arc.

The field *lexem* points to the lexical entry postulated.

The field *grapheme* points to the grapheme node used to build the lexeme node.

The field *limit* corresponds to the field *limit* in an LV: a lexeme node with limit *x* can only appear in solutions not better than *x* (see Frobbing)

Definition at line 145 of file lexemgraph.h.

Data Fields

- Arc [arc](#)
- GraphemNode [grapheme](#)
- LexiconItem [lexem](#)
- LexemGraph [lexemgraph](#)
- Number [limit](#)
- int [no](#)
- Number [tagscore](#)

6.16.2 Field Documentation

6.16.2.1 Arc [LexemNodeStruct::arc](#)

arc in word lattice Definition at line 148 of file lexemgraph.h.

Referenced by [evalTerm\(\)](#), [lgClone\(\)](#), [lgCompatibleSets\(\)](#), [lgCopySelection\(\)](#), [lgMakePath\(\)](#), [lgNewIter\(\)](#), [lgOverlap\(\)](#), [lgPrint\(\)](#), [lgPrintNode\(\)](#), [lgSimultaneous\(\)](#), and [lgWidth\(\)](#).

6.16.2.2 GraphemNode [LexemNodeStruct::grapheme](#)

pointer to the original grapheme Definition at line 150 of file lexemgraph.h.

Referenced by [evalTerm\(\)](#), [lgClone\(\)](#), [lgDistanceOfNodes\(\)](#), and [lgNewIter\(\)](#).

6.16.2.3 LexiconItem [LexemNodeStruct::lexem](#)

lexical entry Definition at line 149 of file lexemgraph.h.

Referenced by cmdDistance(), cvAnalyse(), evalTerm(), getCategories(), lgClone(), lgCopySelection(), lgDelete(), lgDeleteNode(), lgDeleteNodes(), lgLexemeInLexemNodeList(), lgNewIter(), lgPartitions(), lgPrint(), lgPrintNode(), lgQueryCat(), lgSimultaneous(), and parseGetCategory().

6.16.2.4 [LexemGraph](#) [LexemNodeStruct::lexemgraph](#)

pointer back to the lexem graph Definition at line 147 of file lexemgraph.h.

Referenced by lgNewIter(), and lgPrintNode().

6.16.2.5 Number [LexemNodeStruct::limit](#)

limit calculated during frobbing Definition at line 153 of file lexemgraph.h.

Referenced by cnRenew(), lgClone(), and lgNewIter().

6.16.2.6 int [LexemNodeStruct::no](#)

index in [LexemGraph::nodes](#) Definition at line 146 of file lexemgraph.h.

Referenced by cnOptimizeNode(), getCategories(), lgAreDeletableNodes(), lgClone(), lgComputeNoOfPaths(), lgCopySelection(), lgDeleteNode(), lgDeleteNodes(), lgIsDeletedNode(), lgNewIter(), lgPrint(), lgPrintNode(), lgRequireLexeme(), and lgRequireLexemes().

6.16.2.7 Number [LexemNodeStruct::tagscore](#)

Tagger: probability of the associated category Definition at line 151 of file lexemgraph.h.

Referenced by getCategories(), lgCopyTagScores(), lgMostProbablePath(), lgNewIter(), and lgPrint().

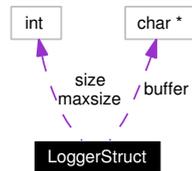
The documentation for this struct was generated from the following file:

- lexemgraph.h

6.17 LoggerStruct Struct Reference

```
#include <hooker.h>
```

Collaboration diagram for LoggerStruct:



6.17.1 Detailed Description

buffer of the output channel.

Definition at line 41 of file hooker.h.

Data Fields

- char * [buffer](#)
- int [maxsize](#)
- int [size](#)

6.17.2 Field Documentation

6.17.2.1 char* [LoggerStruct::buffer](#)

a chunk of memory to buffer stuff in Definition at line 44 of file hooker.h.

Referenced by [hooker_init\(\)](#), [logFlush\(\)](#), [loggerSize\(\)](#), and [logWriteChar\(\)](#).

6.17.2.2 int [LoggerStruct::maxsize](#)

memory allocated in this buffer Definition at line 43 of file hooker.h.

Referenced by [hooker_init\(\)](#), [loggerSize\(\)](#), and [logWriteChar\(\)](#).

6.17.2.3 int [LoggerStruct::size](#)

current size buffered Definition at line 42 of file hooker.h.

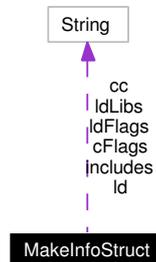
Referenced by [hooker_init\(\)](#), [logFlush\(\)](#), [loggerSize\(\)](#), and [logWriteChar\(\)](#).

The documentation for this struct was generated from the following file:

- [hooker.h](#)

6.18 MakeInfoStruct Struct Reference

Collaboration diagram for MakeInfoStruct:



6.18.1 Detailed Description

default make settings. This structure bundles all the information that is needed to compile and link a binary CDG grammar. This is called `MakeInfo` in according to the normal place where such information is kept.

Definition at line 97 of file `compile.c`.

Data Fields

- String `cc`
- String `cFlags`
- String `includes`
- String `ld`
- String `ldFlags`
- String `ldLibs`

6.18.2 Field Documentation

6.18.2.1 String `MakeInfoStruct::cc`

CC Definition at line 98 of file `compile.c`.

Referenced by `comCompile()`, `comInitialize()`, `comMake()`, `comMakeInfoFree()`, and `comMakeInfoNew()`.

6.18.2.2 String `MakeInfoStruct::cFlags`

CFLAGS Definition at line 100 of file `compile.c`.

Referenced by `comCompile()`, `comInitialize()`, `comMake()`, `comMakeInfoFree()`, and `comMakeInfoNew()`.

6.18.2.3 String `MakeInfoStruct::includes`

INCLUDES Definition at line 103 of file `compile.c`.

Referenced by `comCompile()`, `comInitialize()`, `comMake()`, `comMakeInfoFree()`, and `comMakeInfoNew()`.

6.18.2.4 String `MakeInfoStruct::ld`

LD Definition at line 99 of file `compile.c`.

Referenced by `comCompile()`, `comInitialize()`, `comMake()`, `comMakeInfoFree()`, and `comMakeInfoNew()`.

6.18.2.5 String `MakeInfoStruct::ldFlags`

LD_FLAGS Definition at line 101 of file `compile.c`.

Referenced by `comCompile()`, `comInitialize()`, `comMake()`, `comMakeInfoFree()`, and `comMakeInfoNew()`.

6.18.2.6 String `MakeInfoStruct::ldLibs`

LDLIBS Definition at line 102 of file `compile.c`.

Referenced by `comCompile()`, `comInitialize()`, `comMake()`, `comMakeInfoFree()`, and `comMakeInfoNew()`.

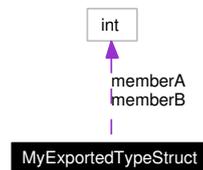
The documentation for this struct was generated from the following file:

- `compile.c`

6.19 MyExportedTypeStruct Struct Reference

```
#include <skel.h>
```

Collaboration diagram for MyExportedTypeStruct:



6.19.1 Detailed Description

short description. long description

Definition at line 36 of file skel.h.

Data Fields

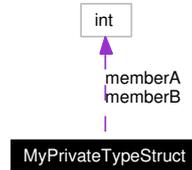
- `int memberA`
- `int memberB`

The documentation for this struct was generated from the following file:

- `skel.h`

6.20 MyPrivateTypeStruct Struct Reference

Collaboration diagram for MyPrivateTypeStruct:



6.20.1 Detailed Description

short description. long description

Definition at line 48 of file skel.c.

Data Fields

- int [memberA](#)
- int [memberB](#)

6.20.2 Field Documentation

6.20.2.1 int [MyPrivateTypeStruct::memberA](#)

description Definition at line 49 of file skel.c.

6.20.2.2 int [MyPrivateTypeStruct::memberB](#)

description Definition at line 50 of file skel.c.

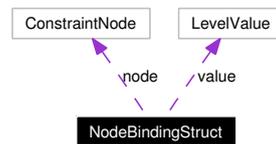
The documentation for this struct was generated from the following file:

- skel.c

6.21 NodeBindingStruct Struct Reference

```
#include <constraintnet.h>
```

Collaboration diagram for NodeBindingStruct:



6.21.1 Detailed Description

This serves merely to pair a constraint node and an LV.

Definition at line 130 of file constraintnet.h.

Data Fields

- [ConstraintNode](#) `node`
- [LevelValue](#) `value`

6.21.2 Field Documentation

6.21.2.1 [ConstraintNode](#) `NodeBindingStruct::node`

constraint node Definition at line 131 of file constraintnet.h.

6.21.2.2 [LevelValue](#) `NodeBindingStruct::value`

level value Definition at line 132 of file constraintnet.h.

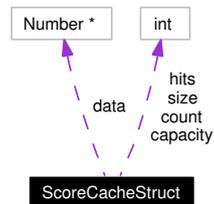
The documentation for this struct was generated from the following file:

- constraintnet.h

6.22 ScoreCacheStruct Struct Reference

```
#include <scache.h>
```

Collaboration diagram for ScoreCacheStruct:



6.22.1 Detailed Description

This structure administers a score for each pair of LVs in a constraint net.

Definition at line 34 of file scache.h.

Data Fields

- int [capacity](#)
- int [count](#)
- Number * [data](#)
- int [hits](#)
- int [size](#)

6.22.2 Field Documentation

6.22.2.1 int [ScoreCacheStruct::capacity](#)

currently available capacity Definition at line 40 of file scache.h.

Referenced by [cnPrintInfo\(\)](#), [scNew\(\)](#), and [scSetScore\(\)](#).

6.22.2.2 int [ScoreCacheStruct::count](#)

holds the total number of LVs registered in the cache Definition at line 37 of file scache.h.

Referenced by [scNew\(\)](#), and [scSetScore\(\)](#).

6.22.2.3 Number* [ScoreCacheStruct::data](#)

points to the underlying vector of Numbers Definition at line 41 of file scache.h.

Referenced by [scDelete\(\)](#), [scGetScore\(\)](#), [scNew\(\)](#), and [scSetScore\(\)](#).

6.22.2.4 int ScoreCacheStruct::hits

holds the number of successful requests to the cache. Definition at line 35 of file scache.h.

Referenced by cnPrintInfo(), scGetScore(), and scNew().

6.22.2.5 int ScoreCacheStruct::size

maximum used capacity Definition at line 39 of file scache.h.

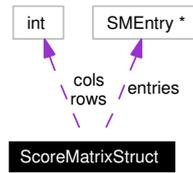
Referenced by cnPrintInfo(), scGetScore(), scNew(), and scSetScore().

The documentation for this struct was generated from the following file:

- scache.h

6.23 ScoreMatrixStruct Struct Reference

Collaboration diagram for ScoreMatrixStruct:



6.23.1 Detailed Description

The score matrix

Since not all matrices in the edges of a constraint net are of equal size, each matrix administrates the number of its rows and columns explicitly in addition to the array itself. The element (r,c) in a score matrix is always referenced by

sm->entries[c * sm->rows + r]

The last element of a matrix always has the index

sm->rows * sm->cols - 1

The type **ScoreMatrix** is only used in a **ConstraintEdge**. A similar function is fulfilled by the structure **ScoreCache** defined by the module [Scache - Cache structures for binary LV scores](#), which has the advantage of being usable even without constraint edges.

Definition at line 75 of file scorematrix.c.

Data Fields

- [int cols](#)
- [SMEntry * entries](#)
- [int rows](#)

6.23.2 Field Documentation

6.23.2.1 int [ScoreMatrixStruct::cols](#)

number of columns in the matrix Definition at line 77 of file scorematrix.c.

Referenced by [smDelete\(\)](#), [smNew\(\)](#), and [smSetAllFlags\(\)](#).

6.23.2.2 [SMEntry* ScoreMatrixStruct::entries](#)

entries in the matrix Definition at line 78 of file scorematrix.c.

Referenced by [smDelete\(\)](#), [smGetFlag\(\)](#), [smGetScore\(\)](#), [smNew\(\)](#), [smSetAllFlags\(\)](#), [smSetFlag\(\)](#), and [smSetScore\(\)](#).

6.23.2.3 int `ScoreMatrixStruct::rows`

number of rows in the matrix Definition at line 76 of file scorematrix.c.

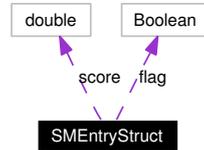
Referenced by `smDelete()`, `smGetFlag()`, `smGetScore()`, `smNew()`, `smSetAllFlags()`, `smSetFlag()`, and `smSetScore()`.

The documentation for this struct was generated from the following file:

- `scorematrix.c`

6.24 SMEntryStruct Struct Reference

Collaboration diagram for SMEntryStruct:



6.24.1 Detailed Description

Entry in the matrix. All entries in the matrix are pairs of a score and a flag indicating whether this pair is valid or not.

Definition at line 46 of file scorematrix.c.

Data Fields

- Boolean [flag](#)
- double [score](#)

6.24.2 Field Documentation

6.24.2.1 Boolean [SMEntryStruct::flag](#)

indicating if this is a valid pari Definition at line 48 of file scorematrix.c.

Referenced by [smGetFlag\(\)](#), [smNew\(\)](#), [smSetAllFlags\(\)](#), and [smSetFlag\(\)](#).

6.24.2.2 double [SMEntryStruct::score](#)

score of this entry Definition at line 47 of file scorematrix.c.

Referenced by [smGetScore\(\)](#), [smNew\(\)](#), and [smSetScore\(\)](#).

The documentation for this struct was generated from the following file:

- [scorematrix.c](#)

Chapter 7

CDG Page Documentation

7.1 Todo List

Global **interface_cell** Write an explanation of the interface and the completion gadget

Global **interface_hashiterator** Write an explanation of the interface and the completion gadget

Global **interface_index** Write an explanation of the interface and the completion gadget

Global **interface_length** Write an explanation of the interface and the completion gadget

Global **interface_list** Write an explanation of the interface and the completion gadget

Global **CDG_EVAL** The EVAL channel is bogus and never used.

Global **CDG_HINT** The HINT channel is bogus and never used.

Global **CDG_HOOK** By no means this is an output channel. This define should be removed and replaced with a proper variable that is altered by the hook command.

Global **CDG_PROLOG** The PROLOG channel is bogus and never used.

Global **lgIsEndNode(GraphemNode n)** should nodes on segment boundaries of incremental parsed input be considered to be endnodes

Group **HookBindings** Please explain this module.

Global **hkInterp** This pointer isn't needed for newer swigs.

Index

- active
 - HookStruct, 189
- annotation_completion_function
 - Command, 45
- Approver
 - Compiler, 62
- ApproverStruct, 161
- ApproverStruct
 - noRounds, 161
 - timer, 161
 - totalCompiledBinaryTime, 162
 - totalCompiledUnaryTime, 162
 - totalInterpretedBinaryTime, 162
 - totalInterpretedUnaryTime, 162
- arc
 - GraphemNodeStruct, 185
 - LexemNodeStruct, 194
- args
 - ChunkerStruct, 165
 - Command, 170
- bAdd
 - Eval, 102
- bAddBadness
 - Eval, 102
- Badness
 - Eval, 102
- BadnessStruct, 163
- BadnessStruct
 - hard, 163
 - no, 163
 - soft, 163
- bClone
 - Eval, 102
- bCompare
 - Eval, 103
- bCopy
 - Eval, 103
- bDelete
 - Eval, 103
- bEqual
 - Eval, 103
- bestBadness
 - Eval, 103
- bNew
 - Eval, 103
- bPrint
 - Eval, 103
- bSubtract
 - Eval, 104
- bSubtractBadness
 - Eval, 104
- buffer
 - LoggerStruct, 196
- cache
 - ConstraintNetStruct, 177
- capacity
 - ScoreCacheStruct, 202
- cc
 - MakeInfoStruct, 197
- ccFile
 - CompilerStruct, 172
- ccFileName
 - CompilerStruct, 172
- Cdg
 - cdgAgInsert, 16
 - cdgCtrlCAllowed, 17
 - cdgCtrlCTrapped, 18
 - cdgDeleteComputed, 16
 - cdgEncodeUmlauts, 18
 - cdgExecPragmas, 16
 - cdgFinalize, 17
 - cdgFreeString, 17
 - cdgInitialize, 17
 - cdgNets, 18, 19
 - cdgParses, 19
 - cdgProblems, 19
 - cdgTimeLimit, 19
 - cdgUser, 19, 20
 - cdgXCDG, 20
 - Chunk, 15
 - Chunker, 15
 - ChunkerStruct, 15
 - ChunkStruct, 15
 - FALSE, 12
 - GraphemNode, 15
 - GraphemNodeStruct, 15
 - LexemNode, 16
 - LexemNodeStruct, 16

- max, [13](#)
- min, [13](#)
- NULL, [13](#)
- TRUE, [14](#)
- Cdg - The Root Module, [11](#)
- CDG_DEBUG
 - HookCore, [118](#)
- CDG_DEFAULT
 - HookCore, [118](#)
- CDG_ERROR
 - HookCore, [118](#)
- CDG_EVAL
 - HookCore, [119](#)
- CDG_HINT
 - HookCore, [119](#)
- CDG_HOOK
 - HookCore, [119](#)
- CDG_INFO
 - HookCore, [119](#)
- CDG_PROFILE
 - HookCore, [119](#)
- CDG_PROGRESS
 - HookCore, [120](#)
- CDG_PROLOG
 - HookCore, [120](#)
- CDG_SEARCHRESULT
 - HookCore, [120](#)
- CDG_WARNING
 - HookCore, [120](#)
- CDG_XML
 - HookCore, [120](#)
- cdgAgInsert
 - Cdg, [16](#)
- cdgCtrlCAllowed
 - Cdg, [17](#)
- cdgCtrlCTrapped
 - Cdg, [18](#)
- Cd gdb
 - cdgstream, [25](#)
 - database, [25](#)
 - dbAge, [25](#)
 - dbAvailable, [22](#)
 - dbClose, [22](#)
 - dbEntry, [22](#)
 - dbFileName, [25](#)
 - dbFinalize, [22](#)
 - dbGetEntries, [22](#)
 - dbIndexName, [25](#)
 - dbInitialize, [23](#)
 - dbLoad, [23](#)
 - dbLoadAll, [23](#)
 - dbLoadEntries, [23](#)
 - dbOpen, [24](#)
 - dbOpenCdgFile, [24](#)
 - dbOpenIndexFile, [24](#)
 - done, [25](#)
 - newDbEntry, [24](#)
 - tmpFilename, [25](#)
- Cd gdb - Get lexical entries out of a berkeley database, [21](#)
- cdgDeleteComputed
 - Cdg, [16](#)
- cdgEncodeUmlauts
 - Cdg, [18](#)
- cdgExecHook
 - HookCore, [122](#)
- cdgExecPragmas
 - Cdg, [16](#)
- cdgFinalize
 - Cdg, [17](#)
- cdgFlush
 - HookCore, [123](#)
- cdgFreeString
 - Cdg, [17](#)
- cdgGetString
 - HookCore, [123](#)
- cdgInitialize
 - Cdg, [17](#)
- cdgNets
 - Cdg, [18, 19](#)
- cdgParses
 - Cdg, [19](#)
- cdgPrintf
 - HookCore, [123](#)
- cdgProblems
 - Cdg, [19](#)
- cdgstream
 - Cd gdb, [25](#)
- cdgTimeLimit
 - Cdg, [19](#)
- cdgUser
 - Cdg, [19, 20](#)
- cdgXCDG
 - Cdg, [20](#)
- cFlags
 - MakeInfoStruct, [197](#)
- Chunk
 - Cdg, [15](#)
- chunk
 - GraphemNodeStruct, [185](#)
- Chunker
 - Cdg, [15](#)
 - chunkerArgs, [40](#)
 - chunkerChunk, [29](#)
 - chunkerChunkDelete, [29](#)
 - chunkerChunkTypeOfString, [29](#)
 - chunkerCloneChunk, [30](#)
 - chunkerCommand, [40](#)

- chunkerCommandValidate, 30
- chunkerDelete, 30
- chunkerFinalize, 30
- chunkerInitialize, 31
- ChunkerMode, 28
- chunkerMode, 40
- chunkerNew, 31
- chunkerPrintChunks, 31
- chunkerReplaceGraphemes, 32
- chunkerStringOfChunkType, 32
- chunkerUseChunker, 40
- ChunkType, 28
- cmpArcs, 32
- cmpChunks, 32
- cmpGraphemes, 32
- compareChunks, 32
- countChunks, 33
- DefaultChunker, 28
- embedChunk, 33
- EvalChunker, 28
- evalChunker, 33
- FakeChunker, 28
- findChunk, 34
- findGrapheme, 34
- getCategories, 34
- getCategory, 35
- getChunks, 35
- getFakeChunks, 35
- getFakeChunksAt, 35
- getFakeChunkType, 36
- initChunker, 36
- initFakeChunker, 36
- initRealChunker, 37
- mergeChunk, 37
- NChunk, 29
- newChunk, 37
- NoChunk, 29
- parseGetCategory, 38
- parseGetGrapheme, 38
- parseGetLabel, 38
- parseGetLevelValue, 38
- parseGetModifiee, 39
- parseGetRoots, 39
- PChunk, 29
- postProcessChunks, 39
- printChunk, 39
- RealChunker, 28
- resetChunker, 39
- terminateChild, 40
- UnknownChunk, 29
- VChunk, 29
- Chunker - Interface to a Chunking Parser, 27
- chunkerArgs
 - Chunker, 40
- chunkerChunk
 - Chunker, 29
- chunkerChunkDelete
 - Chunker, 29
- chunkerChunkTypeOfString
 - Chunker, 29
- chunkerCloneChunk
 - Chunker, 30
- chunkerCommand
 - Chunker, 40
- chunkerCommandValidate
 - Chunker, 30
- chunkerDelete
 - Chunker, 30
- chunkerFinalize
 - Chunker, 30
- chunkerInitialize
 - Chunker, 31
- ChunkerMode
 - Chunker, 28
- chunkerMode
 - Chunker, 40
- chunkerNew
 - Chunker, 31
- chunkerPrintChunks
 - Chunker, 31
- chunkerReplaceGraphemes
 - Chunker, 32
- chunkerStringOfChunkType
 - Chunker, 32
- ChunkerStruct, 165
 - Cdg, 15
- ChunkerStruct
 - args, 165
 - chunks, 165
 - lg, 165
 - mainlevel, 166
 - mode, 166
 - nrLevels, 166
 - nrWords, 166
 - parse, 166
 - pid, 166
 - pipe1, 166
 - pipe2, 166
- chunkerUseChunker
 - Chunker, 40
- chunks
 - ChunkerStruct, 165
 - LexemGraphStruct, 192
- ChunkStruct, 168
 - Cdg, 15
- ChunkStruct
 - from, 168
 - head, 168

- nodes, 168
- parent, 169
- subChunks, 169
- to, 169
- type, 169
- ChunkType
 - Chunker, 28
- cmd
 - HookStruct, 189
- cmdActivate
 - Command, 45
- cmdAnno2Parse
 - Command, 46
- cmdAnnos2prolog
 - Command, 46
- cmdAnnotation
 - Command, 46
- cmdChart
 - Command, 46
- cmdChunk
 - Command, 46
- cmdCloseDB
 - Command, 46
- cmdCompareParses
 - Command, 46
- cmdCompile
 - Command, 47
- cmdConstraint
 - Command, 47
- cmdDeactivate
 - Command, 47
- cmdDistance
 - Command, 47
- cmdEdges
 - Command, 47
- cmdFrobbing
 - Command, 47
- cmdGls
 - Command, 48
- cmdHelp
 - Command, 48
- cmdHierarchy
 - Command, 48
- cmdHook
 - Command, 48
- cmdIncrementalCompletion
 - Command, 48
- cmdInputwordgraph
 - Command, 48
- cmdISearch
 - Command, 48
- cmdLevel
 - Command, 49
- cmdLevelsort
 - Command, 49
- cmdLexicon
 - Command, 49
- cmdLicense
 - Command, 49
- cmdLoad
 - Command, 49
- cmdLs
 - Command, 49
- cmdNet
 - Command, 49
- cmdNetdelete
 - Command, 49
- cmdNetsearch
 - Command, 50
- cmdNewnet
 - Command, 50
- cmdNonSpecCompatible
 - Command, 50
- cmdParsedelete
 - Command, 50
- cmdParses2prolog
 - Command, 50
- cmdPrintParse
 - Command, 50
- cmdPrintParses
 - Command, 50
- cmdQuit
 - Command, 51
- cmdRenewnet
 - Command, 51
- cmdReset
 - Command, 51
- cmdSection
 - Command, 51
- cmdSet
 - Command, 51
- cmdShift
 - Command, 51
- cmdShowlevel
 - Command, 51
- cmdStatus
 - Command, 52
- cmdTagger
 - Command, 52
- cmdTesting
 - Command, 52
- cmdUseconstraint
 - Command, 52
- cmdUselevel
 - Command, 52
- cmdUseLexicon
 - Command, 52
- cmdVerify

- Command, 52
- cmdVersion
 - Command, 53
- cmdWeight
 - Command, 53
- cmdWordgraph
 - Command, 53
- cmdWriteAnno
 - Command, 53
- cmdWriteNet
 - Command, 53
- cmdWriteParses
 - Command, 53
- cmdWriteWordgraph
 - Command, 53
- cmpArcs
 - Chunker, 32
- cmpChunks
 - Chunker, 32
- cmpGraphemes
 - Chunker, 32
- cnBuild
 - Constraintnet, 84
- cnBuildEdges
 - Constraintnet, 84
- cnBuildFinal
 - Constraintnet, 85
- cnBuildInit
 - Constraintnet, 85
- cnBuildIter
 - Constraintnet, 85
- cnBuildLevelValues
 - Constraintnet, 86
- cnBuildLv
 - Constraintnet, 86
- cnBuildNodes
 - Constraintnet, 86
- cnBuildTriple
 - Constraintnet, 87
- cnBuildUpdateArcs
 - Constraintnet, 87
- cnCallback
 - Constraintnet, 88
- cnCompareViolation
 - Constraintnet, 88
- cnConnectedByArc
 - Constraintnet, 88
- cnCounter
 - Constraintnet, 97
- cnDelete
 - Constraintnet, 88
- cnDeleteAllLVs
 - Constraintnet, 89
- cnDeleteBinding
 - Constraintnet, 89
- cnDeleteEdge
 - Constraintnet, 89
- cnDeleteNode
 - Constraintnet, 89
- cnEdgesFlag
 - Constraintnet, 97, 98
- CnEdgesType
 - Constraintnet, 84
- cnFindNet
 - Constraintnet, 90
- cnFindNode
 - Constraintnet, 90
- cnGetGraphemNodeFromArc
 - Constraintnet, 90
- cnGetLattice
 - Constraintnet, 90
- cnInitialize
 - Constraintnet, 90
- cnIsEndNode
 - Constraintnet, 91
- cnIsStartNode
 - Constraintnet, 91
- cnMostRecentlyCreatedNet
 - Constraintnet, 98
- cnNodeComparePrio
 - Constraintnet, 91
- cnNodeCompareSmallest
 - Constraintnet, 91
- cnOptimizeNet
 - Constraintnet, 91
- cnOptimizeNode
 - Constraintnet, 92
- cnPrint
 - Constraintnet, 92
- cnPrintActiveLVs
 - Constraintnet, 93
- cnPrintEdge
 - Constraintnet, 93
- cnPrintInfo
 - Constraintnet, 93
- cnPrintNode
 - Constraintnet, 93
- cnPrintParses
 - Constraintnet, 94
- cnRenew
 - Constraintnet, 94
- cnShowDeletedFlag
 - Constraintnet, 98
- cnSortLVs
 - Constraintnet, 94
- cnSortNodes
 - Constraintnet, 94
- cnSortNodesMethod

- Constraintnet, 98
 - cnTag
 - Constraintnet, 94
 - cnUnaryPruning
 - Constraintnet, 95
 - cnUnaryPruningCompare
 - Constraintnet, 95
 - cnUnaryPruningFraction
 - Constraintnet, 99
 - cnUndeleteAllLVs
 - Constraintnet, 95
 - cnUseNonSpec
 - Constraintnet, 99
 - cols
 - ScoreMatrixStruct, 204
 - com
 - Compiler, 80
 - comAnalyzeGrammar
 - Compiler, 63
 - comApprove
 - Compiler, 63
 - comCompareAllLvPairs
 - Compiler, 63
 - comCompareAllLvs
 - Compiler, 64
 - comCompareLvs
 - Compiler, 64
 - comCompareNets
 - Compiler, 64
 - comCompareWithContext
 - Compiler, 64
 - comCompile
 - Compiler, 64
 - comConnexionToString
 - Compiler, 65
 - comConstraintDepth
 - Compiler, 65
 - comDirectionToString
 - Compiler, 65
 - comEscapeQuotes
 - Compiler, 65
 - comFinalize
 - Compiler, 65
 - comFindComparableLv
 - Compiler, 65
 - comFinitGrammar
 - Compiler, 66
 - comFormulaDepth
 - Compiler, 66
 - comFormulaTypeToString
 - Compiler, 66
 - comFree
 - Compiler, 66
 - comFreeApprover
 - Compiler, 66
 - comFunctionDepth
 - Compiler, 66
 - comIndent
 - Compiler, 66
 - comIndexOfConstraint
 - Compiler, 67
 - comIndexOfHierarchy
 - Compiler, 67
 - comIndexOfVarInfo
 - Compiler, 67
 - comInitGrammar
 - Compiler, 67
 - comInitialize
 - Compiler, 67
 - comLoad
 - Compiler, 67
 - comMake
 - Compiler, 68
 - comMakeInfoFree
 - Compiler, 68
 - comMakeInfoNew
 - Compiler, 68
- Command, 170
 - annotation_completion_function, 45
 - args, 170
 - cmdActivate, 45
 - cmdAnno2Parse, 46
 - cmdAnnos2prolog, 46
 - cmdAnnotation, 46
 - cmdChart, 46
 - cmdChunk, 46
 - cmdCloseDB, 46
 - cmdCompareParses, 46
 - cmdCompile, 47
 - cmdConstraint, 47
 - cmdDeactivate, 47
 - cmdDistance, 47
 - cmdEdges, 47
 - cmdFrobbing, 47
 - cmdGls, 48
 - cmdHelp, 48
 - cmdHierarchy, 48
 - cmdHook, 48
 - cmdIncrementalCompletion, 48
 - cmdInputwordgraph, 48
 - cmdISearch, 48
 - cmdLevel, 49
 - cmdLevelsort, 49
 - cmdLexicon, 49
 - cmdLicense, 49
 - cmdLoad, 49
 - cmdLs, 49
 - cmdNet, 49

- cmdNetdelete, 49
- cmdNetsearch, 50
- cmdNewnet, 50
- cmdNonSpecCompatible, 50
- cmdParsedelete, 50
- cmdParses2prolog, 50
- cmdPrintParse, 50
- cmdPrintParses, 50
- cmdQuit, 51
- cmdRenewnet, 51
- cmdReset, 51
- cmdSection, 51
- cmdSet, 51
- cmdShift, 51
- cmdShowlevel, 51
- cmdStatus, 52
- cmdTagger, 52
- cmdTesting, 52
- cmdUseconstraint, 52
- cmdUselevel, 52
- cmdUseLexicon, 52
- cmdVerify, 52
- cmdVersion, 53
- cmdWeight, 53
- cmdWordgraph, 53
- cmdWriteAnno, 53
- cmdWriteNet, 53
- cmdWriteParses, 53
- cmdWriteWordgraph, 53
- command_completion_function, 53
- commandEval, 54
 - Command, 53
- CommandFunction, 45
 - Command, 53
- commandLoop, 54
 - Command, 54
- commandLoopFlag, 57
 - Command, 57
- comMaxLookupStrings, 68
 - Compiler, 68
- comMaxLookupStringsInFormula, 68
 - Compiler, 68
- comMaxLookupStringsInTerm, 68
 - Compiler, 68
- comNew, 69
 - Compiler, 69
- comNewApprover, 69
 - Compiler, 69
- comOutdent, 69
 - Compiler, 69
- compareChunks, 32
 - Chunker, 32
- Compiler, 62
 - Approver, 62
 - com, 80
 - comAnalyzeGrammar, 63
 - comApprove, 63
 - comCompareAllLvPairs, 63
 - comCompareAllLvs, 64
 - comCompareLvs, 64
 - comCompareNets, 64
 - comCompareWithContext, 64
 - comCompile, 64
 - comConnexionToString, 65
 - comConstraintDepth, 65
 - comDirectionToString, 65
 - comEscapeQuotes, 65
 - comFinalize, 65
 - comFindComparableLv, 65
 - comFinitGrammar, 66
 - comFormulaDepth, 66
 - comFormulaTypeToString, 66
 - comFree, 66
- parse_completion_function, 56
- same, 170
- search_method_completion_function, 56
- section_completion_function, 56
- set_completion_function, 57
- STARTUPMSG, 45
- word_completion_function, 57
- wordgraph_completion_function, 57
- Command - The CDG Scripting Language, 41
- command_completion_function
 - Command, 53
- commandEval
 - Command, 54
- CommandFunction
 - Command, 45
- commandLoop
 - Command, 54
- commandLoopFlag
 - Command, 57
- comMaxLookupStrings
 - Compiler, 68
- comMaxLookupStringsInFormula
 - Compiler, 68
- comMaxLookupStringsInTerm
 - Compiler, 68
- comNew
 - Compiler, 69
- comNewApprover
 - Compiler, 69
- comOutdent
 - Compiler, 69
- compareChunks
 - Chunker, 32
- Compiler
 - Approver, 62
 - com, 80
 - comAnalyzeGrammar, 63
 - comApprove, 63
 - comCompareAllLvPairs, 63
 - comCompareAllLvs, 64
 - comCompareLvs, 64
 - comCompareNets, 64
 - comCompareWithContext, 64
 - comCompile, 64
 - comConnexionToString, 65
 - comConstraintDepth, 65
 - comDirectionToString, 65
 - comEscapeQuotes, 65
 - comFinalize, 65
 - comFindComparableLv, 65
 - comFinitGrammar, 66
 - comFormulaDepth, 66
 - comFormulaTypeToString, 66
 - comFree, 66
- frob_method_completion_function, 54
- function, 170
- getNextArgument, 55
- hierarchy_completion_function, 55
- hook_completion_function, 55
- interface_cell, 57
- interface_commands, 57
- interface_completion, 55
- interface_hashiterator, 57
- interface_index, 58
- interface_length, 58
- interface_list, 58
- level_completion_function, 55
- levelsort_completion_function, 55
- lexicon_completion_function, 55
- make_rl_string, 56
- name, 170
- net_completion_function, 56

- comFreeApprover, 66
- comFunctionDepth, 66
- comIndent, 66
- comIndexOfConstraint, 67
- comIndexOfHierarchy, 67
- comIndexOfVarInfo, 67
- comInitGrammar, 67
- comInitialize, 67
- comLoad, 67
- comMake, 68
- comMakeInfoFree, 68
- comMakeInfoNew, 68
- comMaxLookupStrings, 68
- comMaxLookupStringsInFormula, 68
- comMaxLookupStringsInTerm, 68
- comNew, 69
- comNewApprover, 69
- comOutdent, 69
- Compiler, 62
- CompilerStruct, 62
- comPredicateDepth, 69
- comPrint, 69
- comPrintln, 70
- comRegisterString, 70
- comReturnTypeInfoOfFormula, 70
- comReturnTypeInfoOfFunction, 70
- comReturnTypeInfoOfPredicate, 71
- comReturnTypeInfoOfTerm, 71
- comReturnTypeInfoToString, 71
- comTermDepth, 71
- comTermTypeInfoToString, 71
- comTranslate, 71
- comTranslateAbs, 72
- comTranslateArithmetics, 72
- comTranslateBetween, 72
- comTranslateBinaryConstraints, 72
- comTranslateBottomPeek, 72
- comTranslateChunkHead, 72
- comTranslateConnected, 73
- comTranslateConnexion, 73
- comTranslateConstraint, 73
- comTranslateDirection, 73
- comTranslateDistance, 73
- comTranslateEquation, 74
- comTranslateExists, 74
- comTranslateFormula, 74
- comTranslateFunction, 74
- comTranslateGuard, 74
- comTranslateHas, 74
- comTranslateHeight, 75
- comTranslateIs, 75
- comTranslateLexemNodeAccess, 75
- comTranslateLexicalAccess, 75
- comTranslateLookup, 75
- comTranslateMatch, 76
- comTranslateMinMax, 76
- comTranslateNumber, 76
- comTranslateParens, 76
- comTranslateParent, 76
- comTranslatePhrasequotes, 76
- comTranslatePredicate, 77
- comTranslatePrint, 77
- comTranslatePts, 77
- comTranslateQuotes, 77
- comTranslateStartStop, 77
- comTranslateString, 77
- comTranslateSubsumes, 78
- comTranslateTerm, 78
- comTranslateTopPeek, 78
- comTranslateUnaryConstraints, 78
- comTranslateUnder, 78
- comTranslateUnEquation, 79
- comValueTypeToString, 79
- comWriteDeclarations, 79
- comWriteError, 79
- comWriteFinitFunction, 79
- comWriteFunctions, 79
- comWriteHeader, 80
- comWriteInitFunction, 80
- comWriteWarning, 80
- FINIT_GRAMMAR, 62
- INIT_GRAMMAR, 62
- MakeInfo, 62
- makeInfo, 80
- MakeInfoStruct, 62
- ReturnTypeInfo, 62
- RTAVNode, 63
- RTBoolean, 62
- RTConjunction, 63
- RTDisjunction, 63
- RTError, 63
- RTGraphemNode, 63
- RTLexemNode, 63
- RTLexemPosition, 63
- RTLList, 63
- RTNoError, 63
- RTNumber, 62
- RTPeek, 63
- RTString, 63
- Compiler - compile a constraint grammar, 59
- CompilerStruct, 172
 - Compiler, 62
- CompilerStruct
 - ccFile, 172
 - ccFileName, 172
 - connexions, 172
 - currentConstraint, 173
 - currentFormula, 173

- directions, 173
- indent, 173
- indentString, 173
- indentStrings, 173
- maxLookupStrings, 173
- maxValues, 173
- needsIndent, 174
- objFileName, 174
- soFileName, 174
- strings, 174
- translateOnly, 174
- comPredicateDepth
 - Compiler, 69
- comPrint
 - Compiler, 69
- comPrintln
 - Compiler, 70
- computeNoOfPathsFromStart
 - Lexemgraph, 129
- computeNoOfPathsToEnd
 - Lexemgraph, 129
- comRegisterString
 - Compiler, 70
- comReturnTypeInfoOfFormula
 - Compiler, 70
- comReturnTypeInfoOfFunction
 - Compiler, 70
- comReturnTypeInfoOfPredicate
 - Compiler, 71
- comReturnTypeInfoOfTerm
 - Compiler, 71
- comReturnTypeInfoToString
 - Compiler, 71
- comTermDepth
 - Compiler, 71
- comTermTypeInfoToString
 - Compiler, 71
- comTranslate
 - Compiler, 71
- comTranslateAbs
 - Compiler, 72
- comTranslateArithmetics
 - Compiler, 72
- comTranslateBetween
 - Compiler, 72
- comTranslateBinaryConstraints
 - Compiler, 72
- comTranslateBottomPeek
 - Compiler, 72
- comTranslateChunkHead
 - Compiler, 72
- comTranslateConnected
 - Compiler, 73
- comTranslateConnexion
 - Compiler, 73
- comTranslateConstraint
 - Compiler, 73
- comTranslateDirection
 - Compiler, 73
- comTranslateDistance
 - Compiler, 73
- comTranslateEquation
 - Compiler, 74
- comTranslateExists
 - Compiler, 74
- comTranslateFormula
 - Compiler, 74
- comTranslateFunction
 - Compiler, 74
- comTranslateGuard
 - Compiler, 74
- comTranslateHas
 - Compiler, 74
- comTranslateHeight
 - Compiler, 75
- comTranslateIs
 - Compiler, 75
- comTranslateLexemNodeAccess
 - Compiler, 75
- comTranslateLexicalAccess
 - Compiler, 75
- comTranslateLookup
 - Compiler, 75
- comTranslateMatch
 - Compiler, 76
- comTranslateMinMax
 - Compiler, 76
- comTranslateNumber
 - Compiler, 76
- comTranslateParens
 - Compiler, 76
- comTranslateParent
 - Compiler, 76
- comTranslatePhrasequotes
 - Compiler, 76
- comTranslatePredicate
 - Compiler, 77
- comTranslatePrint
 - Compiler, 77
- comTranslatePts
 - Compiler, 77
- comTranslateQuotes
 - Compiler, 77
- comTranslateStartStop
 - Compiler, 77
- comTranslateString
 - Compiler, 77
- comTranslateSubsumes

- Compiler, 78
- comTranslateTerm
 - Compiler, 78
- comTranslateTopPeek
 - Compiler, 78
- comTranslateUnaryConstraints
 - Compiler, 78
- comTranslateUnder
 - Compiler, 78
- comTranslateUnEquation
 - Compiler, 79
- comValueTypeToString
 - Compiler, 79
- comWriteDeclarations
 - Compiler, 79
- comWriteError
 - Compiler, 79
- comWriteFinitFunction
 - Compiler, 79
- comWriteFunctions
 - Compiler, 79
- comWriteHeader
 - Compiler, 80
- comWriteInitFunction
 - Compiler, 80
- comWriteWarning
 - Compiler, 80
- connexions
 - CompilerStruct, 172
- constraint
 - ConstraintViolationStruct, 182
- constraint_completion_function
 - Command, 54
- ConstraintEdge
 - Constraintnet, 83
- ConstraintEdgeStruct, 175
 - Constraintnet, 83
- ConstraintEdgeStruct
 - counter, 175
 - isMarked, 175
 - m, 175
 - nextSupport, 175
 - prevSupport, 176
 - reverse, 176
 - s, 176
 - scores, 176
 - start, 176
 - stop, 176
- ConstraintNet
 - Constraintnet, 83
- Constraintnet
 - cnBuild, 84
 - cnBuildEdges, 84
 - cnBuildFinal, 85
 - cnBuildInit, 85
 - cnBuildIter, 85
 - cnBuildLevelValues, 86
 - cnBuildLv, 86
 - cnBuildNodes, 86
 - cnBuildTriple, 87
 - cnBuildUpdateArcs, 87
 - cnCallback, 88
 - cnCompareViolation, 88
 - cnConnectedByArc, 88
 - cnCounter, 97
 - cnDelete, 88
 - cnDeleteAllLVs, 89
 - cnDeleteBinding, 89
 - cnDeleteEdge, 89
 - cnDeleteNode, 89
 - cnEdgesFlag, 97, 98
 - CnEdgesType, 84
 - cnFindNet, 90
 - cnFindNode, 90
 - cnGetGraphemNodeFromArc, 90
 - cnGetLattice, 90
 - cnInitialize, 90
 - cnIsEndNode, 91
 - cnIsStartNode, 91
 - cnMostRecentlyCreatedNet, 98
 - cnNodeComparePrio, 91
 - cnNodeCompareSmallest, 91
 - cnOptimizeNet, 91
 - cnOptimizeNode, 92
 - cnPrint, 92
 - cnPrintActiveLVs, 93
 - cnPrintEdge, 93
 - cnPrintInfo, 93
 - cnPrintNode, 93
 - cnPrintParses, 94
 - cnRenew, 94
 - cnShowDeletedFlag, 98
 - cnSortLVs, 94
 - cnSortNodes, 94
 - cnSortNodesMethod, 98
 - cnTag, 94
 - cnUnaryPruning, 95
 - cnUnaryPruningCompare, 95
 - cnUnaryPruningFraction, 99
 - cnUndeleteAllLVs, 95
 - cnUseNonSpec, 99
 - ConstraintEdge, 83
 - ConstraintEdgeStruct, 83
 - ConstraintNet, 83
 - ConstraintNode, 83
 - ConstraintViolation, 83
 - countValidValues, 95
 - cvAnalyse, 95

- cvClone, 96
- cvCompare, 96
- cvCompareNatural, 96
- cvContains, 96
- cvDelete, 97
- cvNew, 97
- cvPrint, 97
- NodeBinding, 83
- Constraintnet - maintainance of constraint nets, 81
- ConstraintNetStruct, 177
- ConstraintNetStruct
 - cache, 177
 - edges, 177
 - evalBinary, 177
 - evalUnary, 178
 - id, 178
 - isBuilt, 178
 - lexemgraph, 178
 - lvTotals, 178
 - nodes, 178
 - parses, 178
 - searchagenda, 179
 - statUnary, 179
 - totalNumberOfValues, 179
 - values, 179
- ConstraintNode
 - Constraintnet, 83
- ConstraintNodeStruct, 180
- ConstraintNodeStruct
 - gn, 180
 - level, 180
 - net, 180
 - noValidValues, 180
 - totalNumberOfValues, 181
 - values, 181
- ConstraintViolation
 - Constraintnet, 83
- ConstraintViolationStruct, 182
- ConstraintViolationStruct
 - constraint, 182
 - lv1, 182
 - lv2, 182
 - nodeBindingIndex1, 182
 - nodeBindingIndex2, 183
 - penalty, 183
- count
 - HookStruct, 189
 - ScoreCacheStruct, 202
- countChunks
 - Chunker, 33
- counter
 - ConstraintEdgeStruct, 175
- countValidValues
 - Constraintnet, 95
- currentConstraint
 - CompilerStruct, 173
- currentFormula
 - CompilerStruct, 173
- cvAnalyse
 - Constraintnet, 95
- cvClone
 - Constraintnet, 96
- cvCompare
 - Constraintnet, 96
- cvCompareNatural
 - Constraintnet, 96
- cvContains
 - Constraintnet, 96
- cvDelete
 - Constraintnet, 97
- cvNew
 - Constraintnet, 97
- cvPrint
 - Constraintnet, 97
- data
 - HookResultStruct, 187
 - ScoreCacheStruct, 202
- database
 - Cd gdb, 25
- dbAge
 - Cd gdb, 25
- dbAvailable
 - Cd gdb, 22
- dbClose
 - Cd gdb, 22
- dbEntry
 - Cd gdb, 22
- dbFileName
 - Cd gdb, 25
- dbFinalize
 - Cd gdb, 22
- dbGetEntries
 - Cd gdb, 22
- dbIndexName
 - Cd gdb, 25
- dbInitialize
 - Cd gdb, 23
- dbLoad
 - Cd gdb, 23
- dbLoadAll
 - Cd gdb, 23
- dbLoadEntries
 - Cd gdb, 23
- dbOpen
 - Cd gdb, 24
- dbOpenCd gFile

- Cd gdb, 24
- dbOpenIndexFile
 - Cd gdb, 24
- DEBUG_SCORECACHE
 - Scache, 142
- DefaultChunker
 - Chunker, 28
- directions
 - CompilerStruct, 173
- distance
 - LexemGraphStruct, 192
- doc
 - Command, 170
- done
 - Cd gdb, 25
- edges
 - ConstraintNetStruct, 177
- embedChunk
 - Chunker, 33
- entries
 - ScoreMatrixStruct, 204
- entryStruct, 184
- entryStruct
 - key, 184
 - oBegin, 184
 - oEnd, 184
- Eval
 - bAdd, 102
 - bAddBadness, 102
 - Badness, 102
 - bClone, 102
 - bCompare, 103
 - bCopy, 103
 - bDelete, 103
 - bEqual, 103
 - bestBadness, 103
 - bNew, 103
 - bPrint, 103
 - bSubtract, 104
 - bSubtractBadness, 104
 - evalBinary, 104
 - evalBinaryConstraint, 105
 - evalConstraint, 105
 - evalCurrentConstraint, 110
 - evalCurrentFormula, 111
 - evalEvaluationMethod, 111
 - evalFinalize, 106
 - evalFormula, 106
 - evalInContext, 107
 - evalInitialize, 107
 - EvalMethodType, 102
 - evalPeekValueMethod, 112
 - evalSloppySubsumesWarnings, 112
 - evalTerm, 107
 - evalUnary, 109
 - evalUnaryConstraint, 109
 - evalValidateEvalMethod, 109
 - has_cache, 112
 - lock_counter, 113
 - lock_tree, 109
 - lock_width, 113
 - peekValue, 110
 - significantlyGreater, 110
 - static_string_chunk_end, 113
 - static_string_chunk_start, 113
 - static_string_chunk_type, 114
 - static_string_from, 114
 - static_string_id, 114
 - static_string_info, 114
 - static_string_to, 114, 115
 - static_string_word, 115
 - unlock_tree, 110
 - worstBadness, 110
- Eval - routines to evaluate constraint formulas, 100
- evalBinary
 - ConstraintNetStruct, 177
 - Eval, 104
- evalBinaryConstraint
 - Eval, 105
- EvalChunker
 - Chunker, 28
- evalChunker
 - Chunker, 33
- evalConstraint
 - Eval, 105
- evalCurrentConstraint
 - Eval, 110
- evalCurrentFormula
 - Eval, 111
- evalEvaluationMethod
 - Eval, 111
- evalFinalize
 - Eval, 106
- evalFormula
 - Eval, 106
- evalHookHandle
 - HookBindings, 156
- evalInContext
 - Eval, 107
- evalInitialize
 - Eval, 107
- EvalMethodType
 - Eval, 102
- evalPeekValueMethod
 - Eval, 112
- evalSloppySubsumesWarnings

- Eval, [112](#)
- evalTerm
 - Eval, [107](#)
- evalUnary
 - ConstraintNetStruct, [178](#)
 - Eval, [109](#)
- evalUnaryConstraint
 - Eval, [109](#)
- evalValidateEvalMethod
 - Eval, [109](#)
- expire
 - Timer, [152](#)
- FakeChunker
 - Chunker, [28](#)
- FALSE
 - Cdg, [12](#)
- findChunk
 - Chunker, [34](#)
- findGrapheme
 - Chunker, [34](#)
- FINIT_GRAMMAR
 - Compiler, [62](#)
- flag
 - SMEEntryStruct, [206](#)
- frob_method_completion_function
 - Command, [54](#)
- from
 - ChunkStruct, [168](#)
- function
 - Command, [170](#)
 - HookStruct, [189](#)
- getCategories
 - Chunker, [34](#)
- getCategory
 - Chunker, [35](#)
- getChunks
 - Chunker, [35](#)
- getFakeChunks
 - Chunker, [35](#)
- getFakeChunksAt
 - Chunker, [35](#)
- getFakeChunkType
 - Chunker, [36](#)
- getHookCmd
 - HookBindings, [156](#)
- getNextArgument
 - Command, [55](#)
- getsHookHandle
 - HookBindings, [156](#)
- glsInteractionHookHandle
 - HookBindings, [157](#)
- gn
 - ConstraintNodeStruct, [180](#)
- gnClone
 - Lexemgraph, [129](#)
- grapheme
 - LexemNodeStruct, [194](#)
- GraphemNode
 - Cdg, [15](#)
- graphemnodes
 - LexemGraphStruct, [192](#)
- GraphemNodeStruct, [185](#)
 - Cdg, [15](#)
- GraphemNodeStruct
 - arc, [185](#)
 - chunk, [185](#)
 - lexemes, [186](#)
 - lexemgraph, [186](#)
 - live, [186](#)
 - no, [186](#)
- hard
 - BadnessStruct, [163](#)
- has_cache
 - Eval, [112](#)
- head
 - ChunkStruct, [168](#)
- hierarchy_completion_function
 - Command, [55](#)
- hits
 - ScoreCacheStruct, [202](#)
- hkCallback
 - HookCore, [124](#)
- hkFinalize
 - HookCore, [124](#)
- hkFindNoOffHook
 - HookCore, [124](#)
- hkHooks
 - HookCore, [125](#)
- hkInitialize
 - HookCore, [124](#)
- hkInterp
 - HookBindings, [159](#)
- hkResult
 - HookBindings, [160](#)
- hkValidate
 - HookCore, [125](#)
- hkVerbosity
 - HookCore, [125](#)
- Hook
 - HookCore, [122](#)
- hook
 - HookResultStruct, [187](#)
- Hook - A callback system, [127](#)
- HOOK_CNBUILDNODES
 - HookCore, [120](#)

- hook_completion_function
 - Command, 55
- HOOK_EVAL
 - HookCore, 121
- HOOK_FLUSH
 - HookCore, 121
- HOOK_GETS
 - HookCore, 121
- HOOK_GLSINTERACTION
 - HookCore, 121
- HOOK_ICINTERACTION
 - HookCore, 121
- HOOK_NSSEARCH
 - HookCore, 121
- HOOK_PARTIALRESULT
 - HookCore, 121
- HOOK_PRINTF
 - HookCore, 121
- HOOK_PROGRESS
 - HookCore, 122
- HOOK_RESET
 - HookCore, 122
- HookBindings
 - evalHookHandle, 156
 - getHookCmd, 156
 - getsHookHandle, 156
 - glsInteractionHookHandle, 157
 - hkInterp, 159
 - hkResult, 160
 - hooker_init, 157
 - HookResult, 156
 - HookResultType, 156
 - ICinteractionHookHandle, 157
 - initHookResult, 157
 - logFlush, 157
 - Logger, 156
 - logger, 160
 - loggerSize, 158
 - logPrintf, 158
 - logWrite, 158
 - logWriteChar, 158
 - netsearchHookHandle, 158
 - partialResultHookHandle, 158
 - progressHookHandle, 159
 - resetHookHandle, 159
 - setHookCmd, 159
 - tclHookHandle, 159
 - TclResultType, 156
- HookBindings - Adaptor to the callback system, 155
- HookCore
 - CDG_DEBUG, 118
 - CDG_DEFAULT, 118
 - CDG_ERROR, 118
 - CDG_EVAL, 119
 - CDG_HINT, 119
 - CDG_HOOK, 119
 - CDG_INFO, 119
 - CDG_PROFILE, 119
 - CDG_PROGRESS, 120
 - CDG_PROLOG, 120
 - CDG_SEARCHRESULT, 120
 - CDG_WARNING, 120
 - CDG_XML, 120
 - cdgExecHook, 122
 - cdgFlush, 123
 - cdgGetString, 123
 - cdgPrintf, 123
 - hkCallback, 124
 - hkFinalize, 124
 - hkFindNoOfHook, 124
 - hkHooks, 125
 - hkInitialize, 124
 - hkValidate, 125
 - hkVerbosity, 125
 - Hook, 122
 - HOOK_CNBUILDNODES, 120
 - HOOK_EVAL, 121
 - HOOK_FLUSH, 121
 - HOOK_GETS, 121
 - HOOK_GLSINTERACTION, 121
 - HOOK_ICINTERACTION, 121
 - HOOK_NSSEARCH, 121
 - HOOK_PARTIALRESULT, 121
 - HOOK_PRINTF, 121
 - HOOK_PROGRESS, 122
 - HOOK_RESET, 122
 - HookFunction, 122
- HookCore - C Part in the core system, 116
- hooker_init
 - HookBindings, 157
- HookFunction
 - HookCore, 122
- HookResult
 - HookBindings, 156
- HookResultStruct, 187
- HookResultStruct
 - data, 187
 - hook, 187
 - intResult, 187
 - stringResult, 187
 - type, 187
- HookResultType
 - HookBindings, 156
- HookStruct, 189
- HookStruct
 - active, 189
 - cmd, 189

- count, [189](#)
- function, [189](#)
- name, [190](#)
- no, [190](#)
- ICinteractionHookHandle
 - HookBindings, [157](#)
- id
 - ConstraintNetStruct, [178](#)
- includes
 - MakeInfoStruct, [197](#)
- indent
 - CompilerStruct, [173](#)
- indentString
 - CompilerStruct, [173](#)
- indentStrings
 - CompilerStruct, [173](#)
- indexOfPair
 - Scache, [142](#)
- INIT_GRAMMAR
 - Compiler, [62](#)
- initChunker
 - Chunker, [36](#)
- initFakeChunker
 - Chunker, [36](#)
- initHookResult
 - HookBindings, [157](#)
- initRealChunker
 - Chunker, [37](#)
- interface_cell
 - Command, [57](#)
- interface_commands
 - Command, [57](#)
- interface_completion
 - Command, [55](#)
- interface_hashiterator
 - Command, [57](#)
- interface_index
 - Command, [58](#)
- interface_length
 - Command, [58](#)
- interface_list
 - Command, [58](#)
- intResult
 - HookResultStruct, [187](#)
- isBuilt
 - ConstraintNetStruct, [178](#)
- isDeletedNode
 - LexemGraphStruct, [192](#)
- isMarked
 - ConstraintEdgeStruct, [175](#)
- key
 - entryStruct, [184](#)
- lattice
 - LexemGraphStruct, [192](#)
- ld
 - MakeInfoStruct, [198](#)
- ldFlags
 - MakeInfoStruct, [198](#)
- ldLibs
 - MakeInfoStruct, [198](#)
- level
 - ConstraintNodeStruct, [180](#)
- level_completion_function
 - Command, [55](#)
- levelsort_completion_function
 - Command, [55](#)
- lexem
 - LexemNodeStruct, [194](#)
- lexemes
 - GraphemNodeStruct, [186](#)
- Lexemgraph
 - computeNoOfPathsFromStart, [129](#)
 - computeNoOfPathsToEnd, [129](#)
 - gnClone, [129](#)
 - lgAreDeletableNodes, [130](#)
 - lgAreDeletedNodes, [130](#)
 - lgClone, [130](#)
 - lgCompactLVs, [141](#)
 - lgCompatibleNodes, [130](#)
 - lgCompatibleSets, [131](#)
 - lgComputeDistances, [131](#)
 - lgComputeNoOfPaths, [131](#)
 - lgContains, [132](#)
 - lgCopySelection, [132](#)
 - lgCopyTagScores, [132](#)
 - lgDelete, [132](#)
 - lgDeleteNode, [133](#)
 - lgDeleteNodes, [133](#)
 - lgDistanceOfNodes, [133](#)
 - lgForbiddenBy, [133](#)
 - lgInitialize, [134](#)
 - lgIntersectingSets, [134](#)
 - lgIsDeletedNode, [134](#)
 - lgIsEndNode, [134](#)
 - lgIsStartNode, [134](#)
 - lgLexemeInLexemNodeList, [135](#)
 - lgMakePath, [135](#)
 - lgMayModify, [135](#)
 - lgMember, [135](#)
 - lgMostProbablePath, [136](#)
 - lgNew, [136](#)
 - lgNewFinal, [136](#)
 - lgNewInit, [137](#)
 - lgNewIter, [137](#)
 - lgOverlap, [138](#)
 - lgPartitions, [138](#)

- lgPrint, [138](#)
- lgPrintNode, [139](#)
- lgQueryCat, [139](#)
- lgRequireLexeme, [139](#)
- lgRequireLexemes, [139](#)
- lgSimultaneous, [139](#)
- lgSpuriousUppercase, [140](#)
- lgSubset, [140](#)
- lgUpdateArcs, [140](#)
- lgWidth, [140](#)
- lexemgraph
 - ConstraintNetStruct, [178](#)
 - GraphemNodeStruct, [186](#)
 - LexemNodeStruct, [195](#)
- Lexemgraph - maintainance of lexem graphs, [128](#)
- LexemGraphStruct, [191](#)
- LexemGraphStruct
 - chunks, [192](#)
 - distance, [192](#)
 - graphemnodes, [192](#)
 - isDeletedNode, [192](#)
 - lattice, [192](#)
 - max, [192](#)
 - min, [193](#)
 - nodes, [193](#)
 - noOfPaths, [193](#)
 - noOfPathsFromStart, [193](#)
 - noOfPathsToEnd, [193](#)
 - tags, [193](#)
- LexemNode
 - Cdg, [16](#)
- LexemNodeStruct, [194](#)
 - Cdg, [16](#)
- LexemNodeStruct
 - arc, [194](#)
 - grapheme, [194](#)
 - lexem, [194](#)
 - lexemgraph, [195](#)
 - limit, [195](#)
 - no, [195](#)
 - tagscore, [195](#)
- lexicon_completion_function
 - Command, [55](#)
- lg
 - ChunkerStruct, [165](#)
- lgAreDeletableNodes
 - Lexemgraph, [130](#)
- lgAreDeletedNodes
 - Lexemgraph, [130](#)
- lgClone
 - Lexemgraph, [130](#)
- lgCompactLVs
 - Lexemgraph, [141](#)
- lgCompatibleNodes
 - Lexemgraph, [130](#)
- lgCompatibleSets
 - Lexemgraph, [131](#)
- lgComputeDistances
 - Lexemgraph, [131](#)
- lgComputeNoOfPaths
 - Lexemgraph, [131](#)
- lgContains
 - Lexemgraph, [132](#)
- lgCopySelection
 - Lexemgraph, [132](#)
- lgCopyTagScores
 - Lexemgraph, [132](#)
- lgDelete
 - Lexemgraph, [132](#)
- lgDeleteNode
 - Lexemgraph, [133](#)
- lgDeleteNodes
 - Lexemgraph, [133](#)
- lgDistanceOfNodes
 - Lexemgraph, [133](#)
- lgForbiddenBy
 - Lexemgraph, [133](#)
- lgInitialize
 - Lexemgraph, [134](#)
- lgIntersectingSets
 - Lexemgraph, [134](#)
- lgIsDeletedNode
 - Lexemgraph, [134](#)
- lgIsEndNode
 - Lexemgraph, [134](#)
- lgIsStartNode
 - Lexemgraph, [134](#)
- lgLexemeInLexemNodeList
 - Lexemgraph, [135](#)
- lgMakePath
 - Lexemgraph, [135](#)
- lgMayModify
 - Lexemgraph, [135](#)
- lgMember
 - Lexemgraph, [135](#)
- lgMostProbablePath
 - Lexemgraph, [136](#)
- lgNew
 - Lexemgraph, [136](#)
- lgNewFinal
 - Lexemgraph, [136](#)
- lgNewInit
 - Lexemgraph, [137](#)
- lgNewIter
 - Lexemgraph, [137](#)
- lgOverlap
 - Lexemgraph, [138](#)

- lgPartitions
 - Lexemgraph, 138
- lgPrint
 - Lexemgraph, 138
- lgPrintNode
 - Lexemgraph, 139
- lgQueryCat
 - Lexemgraph, 139
- lgRequireLexeme
 - Lexemgraph, 139
- lgRequireLexemes
 - Lexemgraph, 139
- lgSimultaneous
 - Lexemgraph, 139
- lgSpuriousUppercase
 - Lexemgraph, 140
- lgSubset
 - Lexemgraph, 140
- lgUpdateArcs
 - Lexemgraph, 140
- lgWidth
 - Lexemgraph, 140
- limit
 - LexemNodeStruct, 195
- live
 - GraphemNodeStruct, 186
- lock_counter
 - Eval, 113
- lock_tree
 - Eval, 109
- lock_width
 - Eval, 113
- logFlush
 - HookBindings, 157
- Logger
 - HookBindings, 156
- logger
 - HookBindings, 160
- loggerSize
 - HookBindings, 158
- LoggerStruct, 196
- LoggerStruct
 - buffer, 196
 - maxsize, 196
 - size, 196
- logPrintf
 - HookBindings, 158
- logWrite
 - HookBindings, 158
- logWriteChar
 - HookBindings, 158
- lv1
 - ConstraintViolationStruct, 182
- lv2
 - ConstraintViolationStruct, 182
- lvTotals
 - ConstraintNetStruct, 178
- m
 - ConstraintEdgeStruct, 175
- mainlevel
 - ChunkerStruct, 166
- make_rl_string
 - Command, 56
- MakeInfo
 - Compiler, 62
- makeInfo
 - Compiler, 80
- MakeInfoStruct, 197
 - Compiler, 62
- MakeInfoStruct
 - cc, 197
 - cFlags, 197
 - includes, 197
 - ld, 198
 - ldFlags, 198
 - ldLibs, 198
- max
 - Cdg, 13
 - LexemGraphStruct, 192
- maxLookupStrings
 - CompilerStruct, 173
- maxsize
 - LoggerStruct, 196
- maxValues
 - CompilerStruct, 173
- memberA
 - MyPrivateTypeStruct, 200
 - Skel, 150
- memberB
 - MyPrivateTypeStruct, 200
 - Skel, 150
- mergeChunk
 - Chunker, 37
- min
 - Cdg, 13
 - LexemGraphStruct, 193
- mode
 - ChunkerStruct, 166
- myExportedFunction
 - Skel, 149
- MyExportedType
 - Skel, 149
- MyExportedTypeStruct, 199
- myExportedVariable
 - Skel, 150
- MyMAXLONG
 - Timer, 151

- myPrivateFunction
 - Skel, 149
- MyPrivateType
 - Skel, 149
- MyPrivateTypeStruct, 200
- MyPrivateTypeStruct
 - memberA, 200
 - memberB, 200
- myPrivateVariable
 - Skel, 150
- name
 - Command, 170
 - HookStruct, 190
- NChunk
 - Chunker, 29
- needsIndent
 - CompilerStruct, 174
- net
 - ConstraintNodeStruct, 180
- net_completion_function
 - Command, 56
- netsearchHookHandle
 - HookBindings, 158
- newChunk
 - Chunker, 37
- newDbEntry
 - Cd gdb, 24
- nextSupport
 - ConstraintEdgeStruct, 175
- no
 - BadnessStruct, 163
 - GraphemNodeStruct, 186
 - HookStruct, 190
 - LexemNodeStruct, 195
- NoChunk
 - Chunker, 29
- node
 - NodeBindingStruct, 201
- NodeBinding
 - Constraintnet, 83
- nodeBindingIndex1
 - ConstraintViolationStruct, 182
- nodeBindingIndex2
 - ConstraintViolationStruct, 183
- NodeBindingStruct, 201
- NodeBindingStruct
 - node, 201
 - value, 201
- nodes
 - ChunkStruct, 168
 - ConstraintNetStruct, 178
 - LexemGraphStruct, 193
- noOfPaths
 - LexemGraphStruct, 193
- noOfPathsFromStart
 - LexemGraphStruct, 193
- noOfPathsToEnd
 - LexemGraphStruct, 193
- noRounds
 - ApproverStruct, 161
- noValidValues
 - ConstraintNodeStruct, 180
- nrLevels
 - ChunkerStruct, 166
- nrWords
 - ChunkerStruct, 166
- NULL
 - Cdg, 13
- oBegin
 - entryStruct, 184
- objFileName
 - CompilerStruct, 174
- oEnd
 - entryStruct, 184
- parent
 - ChunkStruct, 169
- parse
 - ChunkerStruct, 166
- parse_completion_function
 - Command, 56
- parseGetCategory
 - Chunker, 38
- parseGetGrapheme
 - Chunker, 38
- parseGetLabel
 - Chunker, 38
- parseGetLevelValue
 - Chunker, 38
- parseGetModifiee
 - Chunker, 39
- parseGetRoots
 - Chunker, 39
- parses
 - ConstraintNetStruct, 178
- partialResultHookHandle
 - HookBindings, 158
- PChunk
 - Chunker, 29
- peekValue
 - Eval, 110
- penalty
 - ConstraintViolationStruct, 183
- pid
 - ChunkerStruct, 166
- pipe1

- ChunkerStruct, 166
- pipe2
 - ChunkerStruct, 166
- postProcessChunks
 - Chunker, 39
- prevSupport
 - ConstraintEdgeStruct, 176
- printChunk
 - Chunker, 39
- progressHookHandle
 - HookBindings, 159
- RealChunker
 - Chunker, 28
- resetChunker
 - Chunker, 39
- resetHookHandle
 - HookBindings, 159
- ReturnType
 - Compiler, 62
- reverse
 - ConstraintEdgeStruct, 176
- rows
 - ScoreMatrixStruct, 204
- RTAVNode
 - Compiler, 63
- RTBoolean
 - Compiler, 62
- RTConjunction
 - Compiler, 63
- RTDisjunction
 - Compiler, 63
- RTErrror
 - Compiler, 63
- RTGraphemNode
 - Compiler, 63
- RTLexemNode
 - Compiler, 63
- RTLexemPosition
 - Compiler, 63
- RTLlist
 - Compiler, 63
- RTNoError
 - Compiler, 63
- RTNumber
 - Compiler, 62
- RTPeek
 - Compiler, 63
- RTString
 - Compiler, 63
- s
 - ConstraintEdgeStruct, 176
- same
 - Command, 170
- Scache
 - DEBUG_SCORECACHE, 142
 - indexOfPair, 142
 - scDelete, 143
 - scGetScore, 143
 - scInitialize, 143
 - scNew, 143
 - ScoreCache, 143
 - scSetScore, 144
 - scUseCache, 144
- Scache - Cache structures for binary LV scores, 142
- scDelete
 - Scache, 143
- scGetScore
 - Scache, 143
- scInitialize
 - Scache, 143
- scNew
 - Scache, 143
- score
 - SMAEntryStruct, 206
- ScoreCache
 - Scache, 143
- ScoreCacheStruct, 202
- ScoreCacheStruct
 - capacity, 202
 - count, 202
 - data, 202
 - hits, 202
 - size, 203
- ScoreMatrix
 - Scorematrix, 145
- Scorematrix
 - ScoreMatrix, 145
 - ScoreMatrixStruct, 145
 - smDelete, 146
 - SMAEntry, 145
 - SMAEntryStruct, 146
 - smGetFlag, 146
 - smGetScore, 146
 - smNew, 146
 - smSetAllFlags, 147
 - smSetFlag, 147
 - smSetScore, 147
- Scorematrix - The matrix of scores appearing in each ConstraintEdge, 145
- ScoreMatrixStruct, 204
 - Scorematrix, 145
- ScoreMatrixStruct
 - cols, 204
 - entries, 204
 - rows, 204

- scores
 - ConstraintEdgeStruct, 176
- scSetScore
 - Scache, 144
- scUseCache
 - Scache, 144
- search_method_completion_function
 - Command, 56
- searchagenda
 - ConstraintNetStruct, 179
- section_completion_function
 - Command, 56
- set_completion_function
 - Command, 57
- setHookCmd
 - HookBindings, 159
- significantlyGreater
 - Eval, 110
- size
 - LoggerStruct, 196
 - ScoreCacheStruct, 203
- Skel
 - memberA, 150
 - memberB, 150
 - myExportedFunction, 149
 - MyExportedType, 149
 - myExportedVariable, 150
 - myPrivateFunction, 149
 - MyPrivateType, 149
 - myPrivateVariable, 150
 - SOMEMACRO, 149
- Skel - A Skeleton Module, 148
- smDelete
 - Scorematrix, 146
- SMEntry
 - Scorematrix, 145
- SMEntryStruct, 206
 - Scorematrix, 146
- SMEntryStruct
 - flag, 206
 - score, 206
- smGetFlag
 - Scorematrix, 146
- smGetScore
 - Scorematrix, 146
- smNew
 - Scorematrix, 146
- smSetAllFlags
 - Scorematrix, 147
- smSetFlag
 - Scorematrix, 147
- smSetScore
 - Scorematrix, 147
- soFileName
 - CompilerStruct, 174
- soft
 - BadnessStruct, 163
- SOMEMACRO
 - Skel, 149
- start
 - ConstraintEdgeStruct, 176
- STARTUPMSG
 - Command, 45
- static_string_chunk_end
 - Eval, 113
- static_string_chunk_start
 - Eval, 113
- static_string_chunk_type
 - Eval, 114
- static_string_from
 - Eval, 114
- static_string_id
 - Eval, 114
- static_string_info
 - Eval, 114
- static_string_to
 - Eval, 114, 115
- static_string_word
 - Eval, 115
- statUnary
 - ConstraintNetStruct, 179
- stop
 - ConstraintEdgeStruct, 176
- stringResult
 - HookResultStruct, 187
- strings
 - CompilerStruct, 174
- subChunks
 - ChunkStruct, 169
- tags
 - LexemGraphStruct, 193
- tagscore
 - LexemNodeStruct, 195
- tclHookHandle
 - HookBindings, 159
- TclResultType
 - HookBindings, 156
- terminateChild
 - Chunker, 40
- Timer
 - expire, 152
 - MyMAXLONG, 151
 - Timer, 152
 - timerClockTicks, 154
 - timerElapsed, 152
 - timerElapsedFast, 152
 - timerExpired, 154

- TimerFast, [152](#)
- timerFree, [152](#)
- timerInitialize, [153](#)
- timerNew, [153](#)
- timerSetAlarm, [153](#)
- timerStart, [153](#)
- timerStartFast, [154](#)
- timerStopAlarm, [154](#)
- timer
 - ApproverStruct, [161](#)
- Timer - timekeeping functions, [151](#)
- timerClockTicks
 - Timer, [154](#)
- timerElapsed
 - Timer, [152](#)
- timerElapsedFast
 - Timer, [152](#)
- timerExpired
 - Timer, [154](#)
- TimerFast
 - Timer, [152](#)
- timerFree
 - Timer, [152](#)
- timerInitialize
 - Timer, [153](#)
- timerNew
 - Timer, [153](#)
- timerSetAlarm
 - Timer, [153](#)
- timerStart
 - Timer, [153](#)
- timerStartFast
 - Timer, [154](#)
- timerStopAlarm
 - Timer, [154](#)
- tmpFilename
 - Cdgdb, [25](#)
- to
 - ChunkStruct, [169](#)
- totalCompiledBinaryTime
 - ApproverStruct, [162](#)
- totalCompiledUnaryTime
 - ApproverStruct, [162](#)
- totalInterpretedBinaryTime
 - ApproverStruct, [162](#)
- totalInterpretedUnaryTime
 - ApproverStruct, [162](#)
- totalNumberOfValues
 - ConstraintNetStruct, [179](#)
 - ConstraintNodeStruct, [181](#)
- translateOnly
 - CompilerStruct, [174](#)
- TRUE
 - Cdg, [14](#)
- type
 - ChunkStruct, [169](#)
 - HookResultStruct, [187](#)
- UnknownChunk
 - Chunker, [29](#)
- unlock_tree
 - Eval, [110](#)
- value
 - NodeBindingStruct, [201](#)
- values
 - ConstraintNetStruct, [179](#)
 - ConstraintNodeStruct, [181](#)
- VChunk
 - Chunker, [29](#)
- word_completion_function
 - Command, [57](#)
- wordgraph_completion_function
 - Command, [57](#)
- worstBadness
 - Eval, [110](#)