The Regulus Cookbook

Manny Rayner

August 26, 2010

**CENTER FOR THE STUDY
OF LANGUAGE
AND INFORMATION**

To our users
who should nag us even more often

— *M.R.*

Contents

Foreword xxiii

1 Introduction 1
- 1.1 What this book is about 1
- 1.2 What Regulus can do 1
- 1.3 What Regulus can't do 1
- 1.4 How to read this book 1
- 1.5 Regulus literature 1
- 1.6 Online Regulus examples 1

I Getting Started 3

2 Downloading and Installing 5
- 2.1 What you need to run Regulus 5
- 2.2 Installing Cygwin 5
- 2.3 Installing Nuance 5
- 2.4 Installing Sicstus 5
- 2.5 Installing Regulus 5
- 2.6 Installing MedSLT 5

3 Basic Functionality 7
- 3.1 Regulus config files 7
- 3.2 The command-line development environment 7
- 3.3 The window-oriented development environment 7
- 3.4 Loading and running a toy grammar 7
- 3.5 Loading and running a toy dialogue system 7

3.6 Loading and running a toy translation system 7
3.7 Compiling and running a Java-based example 7
3.8 Running Regulus offline 7

II The Development Environment 9

4 The Command Line Development Environment 11
4.1 Overview of the command-line environment 11
4.2 Text input 11
4.3 Logical form input 11
4.4 Speech input 11
4.5 Wavfile input 11
4.6 Command-line help 11

III Grammars and Text Processing 13

5 Basic Grammar Writing 15
5.1 Loading a grammar 15
5.2 A toy grammar 15
5.3 Config files for grammars 15
5.4 Include statements 15
5.5 Grammar declarations 15
5.6 Macros 15
5.7 Macros for lexicon entries 15
5.8 Macros for semantics 15
5.9 Debugging grammars from the command-line 15
5.10 Debugging grammars with the stepper 15
5.11 Converting grammars to other formats 15

6 Parsing and Generation 17
6.1 Different types of parser 17
6.2 The left-corner parser 17
6.3 The DCG parser 17
6.4 The reflective DCG parser 17
6.5 Using Nuance as a parser 17
6.6 Displaying parse trees 17
6.7 Parse preferences 17

6.8 Compiling grammars for generation 17
6.9 Config files for generation 17
6.10 Using generation mode 17
6.11 Generation preferences 17
6.12 Tracing generation 17
6.13 Calling a parser from Prolog 17
6.14 Calling a generator from Prolog 17
6.15 Random generation of sentences from a Regulus grammar 17

7 Grammar Specialisation 19

7.1 A toy specialised grammar 19
7.2 Invoking grammar specialisation 19
7.3 Using a specialised grammar for parsing 19
7.4 Using a specialised grammar for recognition 19
7.5 Using a specialised grammar for generation 19
7.6 Config files for grammar specialisation 19
7.7 Defining operationality criteria 19
7.8 Avoiding multiple top-level rules in specialised grammars 19
7.9 Debugging specialised grammars 19
7.10 Multiple top-level grammars 19
7.11 Including lexicon entries directly 19
7.12 Handling “explosions” in CFG compilation 19
7.13 Ignoring syntactic features 19

8 The English Resource Grammar 21

8.1 Using the English resource grammar 21
8.2 English lexicon macros 21
8.3 Nouns 21
8.4 Proper Names 21
8.5 Adjectives 21
8.6 Verbs 21
8.7 Prepositions 21
8.8 PPs 21

9 Other Resource Grammars 23

9.1 Finding other resource grammars 23

9.2 The Romance resource grammar 23

9.3 The Japanese resource grammar 23

9.4 The Arabic resource grammar 23

9.5 The “Core” resource grammar 23

10 Robust Parsing with Alterf 25

10.1 Overview of Alterf 25

10.2 Surface patterns 25

10.3 LF patterns 25

10.4 Subordinate clause boundary patterns 25

10.5 Training on text data 25

10.6 Training on speech data 25

10.7 Using Alterf at runtime 25

IV Speech Processing 27

11 Speech 29

11.1 Overview of speech functionality 29

11.2 Config files for speech 29

11.3 Compiling Regulus into GSL 29

11.4 Compiling GSL into recognition packages 29

11.5 Probabilistic tuning of recognition packages 29

11.6 Recorded wavfile output 29

11.7 Speech input from the command-line 29

11.8 Speech output from the command-line 29

11.9 A Regulus-friendly interface to `batchrec` 29

11.10 A Regulus-friendly interface to `nl-tool` 29

12 Using Nuance 9 31

12.1 Overview of Nuance 9 31

12.2 Making your grammars non-recursive 31

12.3 Strcat semantics 31

13 Statistical Recognition and Intelligent Help 33
13.1 Overview of statistical recognition and help 33
13.2 Compiling an SLM 33
13.3 Defining Regulus-based SLM classes 33
13.4 Grammar-based generation of SLM training data 33
13.5 Building a help system 33
13.6 Defining help classes 33
13.7 Testing a help system from the command-line 33
13.8 Using a help system in an application 33

V Spoken Dialogue 35

14 Basic Dialogue Processing 37
14.1 Using dialogue mode 37
14.2 A toy dialogue system 37
14.3 Writing an input manager 37
14.4 Writing a dialogue manager 37
14.5 Writing an output manager 37
14.6 Config files for dialogue systems 37

15 Advanced Dialogue Processing 39
15.1 Writing `lf_pattern` rules 39
15.2 Implementing ellipsis resolution rules 39
15.3 Implementing reference resolution rules 39
15.4 Paraphrasing from the dialogue move 39
15.5 Handling non-speech inputs 39
15.6 Setting context in dialogue 39
15.7 Asynchronous dialogue systems 39
15.8 Open mic dialogue systems 39
15.9 The Regulus dialogue server 39
15.10 Regression testing for dialogue systems 39
15.11 Regression testing for multi-modal systems 39
15.12 Using N-best processing in dialogue 39

VI Translation 41

16 Basic Translation Systems 43

16.1 A toy translation system 44
16.2 Config files for translation 44
16.3 Using translation mode 44
16.4 Simple translation rules 44
16.5 Conditional translation rules 44
16.6 Tracing translation 44
16.7 Transfer variables 44
16.8 Macros in translation rules 44
16.9 Reversible translation rules 44
16.10 Using interlingua 44
16.11 Running regression tests in text mode 44
16.12 Getting timing figures for regression tests 44
16.13 Running regression tests in speech mode 44
16.14 Using paraphrases in regression testing 44
16.15 Re-running regression tests in speech mode 44
16.16 Splitting corpora into in- and out-of-coverage 44
16.17 Judging regression tests (Prolog format) 44
16.18 Judging regression tests (CSV format) 44
16.19 Comparing regression test results 44
16.20 Summary of translation commands 44

17 Advanced Translation Systems 45

17.1 Interlingua checking grammars 45
17.2 Interlingua debugging 45
17.3 Interlingua centred development 45
17.4 Backtranslation 45
17.5 Almost Flat Functional Semantics 45
17.6 Simple `role_transfer_rules` 45
17.7 Conditional `role_transfer_rules` 45
17.8 `role_list_transfer_rules` 45
17.9 Generating native script 45
17.10 Generating gloss translations 45
17.11 The translation server 45
17.12 Translating ellipsis 45
17.13 Answer ellipsis 45

17.14 Learning translation rules from examples 45
17.15 Using bidirectional translation mode 45
17.16 Regression testing for bidirectional translation 45
17.17 Using N-best processing in translation 45

VII Command reference 47

18 Regulus Commands 49
18.1 Overview of Commands 49
18.2 `ANSWER_ELLIPSIS_OFF` 50
18.3 `ANSWER_ELLIPSIS_ON` 50
18.4 `BATCH_DIALOGUE Arg1` 50
18.5 `BATCH_DIALOGUE` 50
18.6 `BATCH_DIALOGUE_SPEECH Arg1` 51
18.7 `BATCH_DIALOGUE_SPEECH` 51
18.8 `BATCH_DIALOGUE_SPEECH_AGAIN Arg1` 51
18.9 `BATCH_DIALOGUE_SPEECH_AGAIN` 52
18.10 `BATCH_HELP Arg1` 52
18.11 `BATCH_HELP` 52
18.12 `BIDIRECTIONAL_OFF` 52
18.13 `BIDIRECTIONAL_ON` 52
18.14 `CAT Arg1` 53
18.15 `CHECK_ALTERF_PATTERNS` 53
18.16 `CHECK_BACKTRANSLATION Arg1` 54
18.17 `CHECK_PARAPHRASES` 54
18.18 `CLOSE_DOWN_RECOGNITION` 54
18.19 `COMPACTION` 55
18.20 `COMPILE_ELLIPSIS_PATTERNS` 55
18.21 `COMPILE_HELP` 55
18.22 `DCG` 55
18.23 `DIALOGUE` 56
18.24 `DIALOGUE_SPEAKER` 56
18.25 `DIALOGUE_TIME` 56
18.26 `DOC Arg1` 56
18.27 `DUMP_NBEST_TRAINING_DATA_OFF` 57
18.28 `DUMP_NBEST_TRAINING_DATA_ON` 57

18.29 EBL 57
18.30 EBL_ANALYSIS 58
18.31 EBL_GEMINI 58
18.32 EBL_GENERATION 58
18.33 EBL_GRAMMAR_PROBS 59
18.34 EBL_LOAD 59
18.35 EBL_LOAD_GENERATION Arg1 59
18.36 EBL_LOAD_GENERATION 60
18.37 EBL_MODE 60
18.38 EBL_NUANCE 60
18.39 EBL_POSTPROCESS 61
18.40 EBL_TRAIN 61
18.41 EBL_TREEBANK 61
18.42 ECHO_OFF 61
18.43 ECHO_ON 61
18.44 FEAT Arg1 62
18.45 GEMINI 62
18.46 GENERATE_TRACE_OFF 62
18.47 GENERATE_TRACE_ON 62
18.48 GENERATION 62
18.49 HELP Arg1 63
18.50 HELP 63
18.51 HELP_CONFIG Arg1 63
18.52 HELP_RESPONSE_OFF 64
18.53 HELP_RESPONSE_ON 64
18.54 INCREMENTAL_TREEBANKING_OFF 64
18.55 INCREMENTAL_TREEBANKING_ON 64
18.56 INIT_DIALOGUE Arg1 64
18.57 INIT_DIALOGUE 65
18.58 INTERLINGUA 65
18.59 INTERLINGUA_DEBUGGING_OFF 65
18.60 INTERLINGUA_DEBUGGING_ON 65
18.61 INTERLINGUA_TRACE_OFF 66
18.62 INTERLINGUA_TRACE_ON 66
18.63 KILL_NUANCE_PARSERS 66
18.64 LC 66

18.65 LF_POST_PROCESSING_OFF 66
18.66 LF_POST_PROCESSING_ON 66
18.67 LINE_INFO_OFF 67
18.68 LINE_INFO_ON 67
18.69 LIST_MISSING_HELP_DECLARATIONS 67
18.70 LOAD 68
18.71 LOAD_DEBUG 68
18.72 LOAD_DIALOGUE 68
18.73 LOAD_GENERATION Arg1 68
18.74 LOAD_GENERATION 69
18.75 LOAD_HELP 70
18.76 LOAD_PREFERENCES 70
18.77 LOAD_RECOGNITION Arg1 70
18.78 LOAD_RECOGNITION 70
18.79 LOAD_RECOGNITION_GENERATION 71
18.80 LOAD_SURFACE_PATTERNS 71
18.81 LOAD_TRANSLATE 71
18.82 MAKE_TARGET_GRAMMAR_PROBS_CORPUS Arg1 72
18.83 MAKE_TARGET_SENT_CORPUS 73
18.84 NORMAL_PROCESSING 73
18.85 NO_COMPACTION 73
18.86 NO_ELLIPSIS_PROCESSING 73
18.87 NO_INTERLINGUA 74
18.88 NUANCE 74
18.89 NUANCE_COMPILE 74
18.90 NUANCE_COMPILE_WITH_PCFG 75
18.91 NUANCE_PARSER 75
18.92 PARSE_HISTORY Args 76
18.93 PRINT_TREE_CATEGORIES_OFF 76
18.94 PRINT_TREE_CATEGORIES_ON 76
18.95 PRINT_TREE_SUMMARY_OFF 76
18.96 PRINT_TREE_SUMMARY_ON 76
18.97 RANDOM_GENERATE Arg1 Arg2 78
18.98 RANDOM_GENERATE Arg1 78
18.99 RECOGNISE 78
18.100RELOAD_CFG 78

18.101 SET_BATCH_DIALOGUE_FORMAT Arg1 80
18.102 SET_NBEST_N Arg1 80
18.103 SET_NOTIONAL_SPEAKER Arg1 80
18.104 SET_NOTIONAL_TIME Arg1 81
18.105 SET_REGSERVER_TIMEOUT Arg1 81
18.106 SPLIT_SPEECH_CORPUS Arg1 Arg2 Arg3 Arg4 81
18.107 SPLIT_SPEECH_CORPUS Arg1 Arg2 Arg3 82
18.108 SPLIT_SPEECH_CORPUS training_corpus Arg1 Arg2 Arg3 82
18.109 STEPPER 82
18.110 STORE_TRANSLATION_TARGET_VOCAB Arg1 82
18.111 SURFACE 82
18.112 TRANSLATE 83
18.113 TRANSLATE_CORPUS Arg1 83
18.114 TRANSLATE_CORPUS 83
18.115 TRANSLATE_PARSE_TIMES Arg1 83
18.116 TRANSLATE_PARSE_TIMES 84
18.117 TRANSLATE_SPEECH_CORPUS Arg1 84
18.118 TRANSLATE_SPEECH_CORPUS 84
18.119 TRANSLATE_SPEECH_CORPUS_AGAIN Arg1 84
18.120 TRANSLATE_SPEECH_CORPUS_AGAIN 85
18.121 TRANSLATE_TRACE_OFF 85
18.122 TRANSLATE_TRACE_ON 85
18.123 UNSET_NOTIONAL_SPEAKER 85
18.124 UNSET_NOTIONAL_TIME 85
18.125 UPDATE_DIALOGUE_JUDGEMENTS Arg1 85
18.126 UPDATE_DIALOGUE_JUDGEMENTS 86
18.127 UPDATE_DIALOGUE_JUDGEMENTS_SPEECH Arg1 86
18.128 UPDATE_DIALOGUE_JUDGEMENTS_SPEECH 86
18.129 UPDATE_RECOGNITION_JUDGEMENTS Arg1 87
18.130 UPDATE_RECOGNITION_JUDGEMENTS 87
18.131 UPDATE_TRANSLATION_JUDGEMENTS Arg1 87
18.132 UPDATE_TRANSLATION_JUDGEMENTS 88
18.133 UPDATE_TRANSLATION_JUDGEMENTS_CSV Arg1 88
18.134 UPDATE_TRANSLATION_JUDGEMENTS_CSV 88
18.135 UPDATE_TRANSLATION_JUDGEMENTS_SPEECH Arg1 89
18.136 UPDATE_TRANSLATION_JUDGEMENTS_SPEECH 89

18.137UPDATE_TRANSLATION_JUDGEMENTS_SPEECH_CSV Arg1 89
18.138UPDATE_TRANSLATION_JUDGEMENTS_SPEECH_CSV 90
18.139WAVFILES Arg1 90
18.140WAVFILES 90

19 Config files 93

19.1 Structure of config files 93
19.2 alterf_patterns_file 94
19.3 alterf_sents_file 94
19.4 alterf_treebank_file 94
19.5 analysis_time_limit 94
19.6 answer_config_file 94
19.7 batchrec_trace 94
19.8 batchrec_trace_prolog 94
19.9 batchrec_trace_prolog_with_transcriptions 94
19.10 batchrec_trace_prolog_with_transcriptions(Arg1) 94
19.11 collocation_rules 94
19.12 compiled_collocation_rules 94
19.13 compiled_ellipsis_classes 94
19.14 compiled_from_interlingua_rules 95
19.15 compiled_graphical_orthography_rules 95
19.16 compiled_lf_patterns 95
19.17 compiled_lf_rewrite_rules 95
19.18 compiled_original_script_collocation_rules 95
19.19 compiled_original_script_orthography_rules 95
19.20 compiled_orthography_rules 95
19.21 compiled_recognition_orthography_rules 95
19.22 compiled_surface_constituent_rules 95
19.23 compiled_to_interlingua_rules 95
19.24 compiled_to_source_discourse_rules 95
19.25 compiled_transfer_rules 95
19.26 dcg_grammar 95
19.27 default_compiled_ellipsis_classes 96
19.28 dialogue_batchrec_trace_prolog_with_transcriptions 96
19.29 dialogue_batchrec_trace_prolog_with_transcriptions(Arg1) 96
19.30 dialogue_corpus 96

19.31 dialogue_corpus(Arg1) 96
19.32 dialogue_corpus_judgements 96
19.33 dialogue_corpus_results 96
19.34 dialogue_corpus_results(Arg1) 96
19.35 dialogue_files 96
19.36 dialogue_processing_time_limit 96
19.37 dialogue_rec_params 96
19.38 dialogue_speech_corpus 96
19.39 dialogue_speech_corpus(Arg1) 97
19.40 dialogue_speech_corpus_results 97
19.41 dialogue_speech_corpus_results(Arg1) 97
19.42 discard_lexical_info_in_ebl_training 97
19.43 discriminants 97
19.44 ebl_context_use_threshold 97
19.45 ebl_corpus 97
19.46 ebl_filter_pred 97
19.47 ebl_gemini_grammar 98
19.48 ebl_grammar_probs 98
19.49 ebl_ignore_feats 98
19.50 ebl_ignore_feats_file 98
19.51 ebl_include_lex 98
19.52 ebl_multiple_grammar_decls 98
19.53 ebl_nuance_grammar 98
19.54 ebl_operationality 98
19.55 ebl_rationalised_corpus 98
19.56 ebl_raw_regulus_grammar 99
19.57 ebl_regulus_component_grammar 99
19.58 ebl_regulus_grammar 99
19.59 ebl_regulus_no_binarise_grammar 99
19.60 ebl_treebank 99
19.61 ellipsis_classes 99
19.62 ellipsis_classes_sents_file 99
19.63 ellipsis_classes_treebank_file 99
19.64 filtered_interlingua_declarations_file 99
19.65 from_interlingua_rule_learning_config_file 99
19.66 from_interlingua_rules 99

19.67 from_interlingua_translation_corpus_judgements 100
19.68 gemini_grammar 100
19.69 generation_dcg_grammar 100
19.70 generation_grammar 100
19.71 generation_grammar(Arg1) 100
19.72 generation_incremental_deepening_parameters 100
19.73 generation_module_name 100
19.74 generation_preferences 100
19.75 generation_regulus_grammar 101
19.76 generation_rules 101
19.77 generation_rules(Arg1) 101
19.78 generation_time_limit 101
19.79 global_context 101
19.80 gloss_generation_rules 101
19.81 grammar_probs_data 101
19.82 ignore_subdomain 101
19.83 interlingua_declarations 102
19.84 interlingua_structure 102
19.85 lc_tables_file 102
19.86 lf_patterns 102
19.87 lf_patterns_modules 102
19.88 lf_postproc_pred 102
19.89 macro_expanded_grammar 102
19.90 missing_help_class_decls 102
19.91 nbest_preferences 102
19.92 nbest_training_data_file 102
19.93 no_spaces_in_original_script 102
19.94 nuance_compile_params 103
19.95 nuance_grammar 103
19.96 nuance_grammar_for_compilation 103
19.97 nuance_grammar_for_pcfg_training 103
19.98 nuance_language_pack 103
19.99 nuance_recognition_package 103
19.100only_translate_up_to_interlingua 103
19.101original_script_collocation_rules 103
19.102original_script_encoding 103

19.103original_script_generation_rules 103

19.104original_script_orthography_rules 103

19.105orthography_rules 103

19.106paraphrase_corpus 104

19.107paraphrase_generation_grammar 104

19.108parse_preferences 104

19.109parsing_history_file 104

19.110pcfg_training_output_directory 104

19.111prolog_semantics 104

19.112reflective_dcg_grammar 104

19.113reflective_dcg_grammar_for_generation 104

19.114regulus_grammar 104

19.115regulus_no_sem_decls 104

19.116resolution_preferences 104

19.117role_marked_semantics 104

19.118stanford_dcg_debug_grammar 105

19.119stanford_dcg_grammar 105

19.120strcat_semantics 105

19.121surface_constituent_rules 105

19.122surface_patterns 105

19.123surface_postprocessing 105

19.124tagging_grammar 105

19.125target_model 105

19.126targeted_help_backed_off_corpus_file 105

19.127targeted_help_classes_file 105

19.128targeted_help_corpus_file 106

19.129targeted_help_source_files 106

19.130test_corpus 106

19.131tmp_ebl_operational_file 106

19.132tmp_preds 106

19.133to_interlingua_rule_learning_config_file 106

19.134to_interlingua_rules 106

19.135to_interlingua_translation_corpus_judgements 106

19.136to_source_discourse_rules 106

19.137top_level_cat 106

19.138top_level_generation_cat 106

19.139top_level_generation_feat 107
19.140top_level_generation_pred 107
19.141transfer_rules 107
19.142translate_from_interlingua 107
19.143translation_corpus 107
19.144translation_corpus(Arg1) 107
19.145translation_corpus_judgements 107
19.146translation_corpus_recognition_judgements 108
19.147translation_corpus_results 108
19.148translation_corpus_results(Arg1) 108
19.149translation_corpus_tmp_recognition_judgements 108
19.150translation_corpus_tmp_recognition_judgements(Arg1) 108
19.151translation_rec_params 108
19.152translation_speech_corpus 109
19.153translation_speech_corpus(Arg1) 109
19.154translation_speech_corpus_results 109
19.155translation_speech_corpus_results(Arg1) 109
19.156tts_command 109
19.157wavfile_directory 109
19.158wavfile_preceding_context 109
19.159wavfile_preceding_context(Arg1) 109
19.160wavfile_recording_script 110
19.161wavfiles 110
19.162working_directory 111
19.163working_file_prefix 111

References **113**

Index **115**

Foreword

Buy this fine piece of work, me hearties!

Professor J. Hook
University of Neverland, Far Far Away, PX

Whenever

Acknowledgments

We want to thank Nuance for being wonderful, other people for giving us money, and all that stuff.

1

Introduction

Manny Rayner

1.1 What this book is about

If you build things using Regulus, you need this book. Here's why. Um. Not quite finished yet. Expand this chapter later.

1.2 What Regulus can do

Regulus has been used to build some cool systems. For example, (Rayner et al., 2005, Bouillon et al., 2005).

1.3 What Regulus can't do

1.4 How to read this book

1.5 Regulus literature

1.6 Online Regulus examples

Part I

Getting Started

2

Downloading and Installing

Manny Rayner

2.1 What you need to run Regulus

2.2 Installing Cygwin

2.3 Installing Nuance

2.4 Installing Sicstus

2.5 Installing Regulus

2.6 Installing MedSLT

3

Basic Functionality

Manny Rayner

3.1 Regulus config files

3.2 The command-line development environment

3.3 The window-oriented development environment

3.4 Loading and running a toy grammar

3.5 Loading and running a toy dialogue system

3.6 Loading and running a toy translation system

3.7 Compiling and running a Java-based example

3.8 Running Regulus offline

Part II

The Development Environment

4

The Command Line Development Environment

Manny Rayner

4.1 Overview of the command-line environment

4.2 Text input

4.3 Logical form input

4.4 Speech input

4.5 Wavfile input

4.6 Command-line help

Part III

Grammars and Text Processing

5

Basic Grammar Writing

Manny Rayner

5.1 Loading a grammar
5.2 A toy grammar
5.3 Config files for grammars
5.4 Include statements
5.5 Grammar declarations
5.6 Macros
5.7 Macros for lexicon entries
5.8 Macros for semantics
5.9 Debugging grammars from the command-line
5.9.1 Parsing non-top constituents
5.9.2 Commands for grammar debugging
5.10 Debugging grammars with the stepper
5.11 Converting grammars to other formats

6

Parsing and Generation

Manny Rayner

6.1 Different types of parser

6.2 The left-corner parser

6.3 The DCG parser

6.4 The reflective DCG parser

6.5 Using Nuance as a parser

6.6 Displaying parse trees

6.7 Parse preferences

6.8 Compiling grammars for generation

6.9 Config files for generation

6.10 Using generation mode

6.11 Generation preferences

6.12 Tracing generation

6.13 Calling a parser from Prolog

6.14 Calling a generator from Prolog

6.15 Random generation of sentences from a Regulus grammar

7

Grammar Specialisation

Manny Rayner

7.1 A toy specialised grammar

7.2 Invoking grammar specialisation

7.3 Using a specialised grammar for parsing

7.4 Using a specialised grammar for recognition

7.5 Using a specialised grammar for generation

7.6 Config files for grammar specialisation

7.7 Defining operationality criteria

7.8 Avoiding multiple top-level rules in specialised grammars

7.9 Debugging specialised grammars

7.10 Multiple top-level grammars

7.11 Including lexicon entries directly

7.12 Handling "explosions" in CFG compilation

7.13 Ignoring syntactic features

8

The English Resource Grammar

Manny Rayner

8.1 Using the English resource grammar

8.2 English lexicon macros

8.3 Nouns

8.4 Proper Names

8.5 Adjectives

8.6 Verbs

8.7 Prepositions

8.8 PPs

9

Other Resource Grammars

Manny Rayner

9.1 Finding other resource grammars

9.2 The Romance resource grammar

9.3 The Japanese resource grammar

9.4 The Arabic resource grammar

9.5 The "Core" resource grammar

10

Robust Parsing with Alterf

Manny Rayner

10.1 Overview of Alterf

10.2 Surface patterns

10.3 LF patterns

10.4 Subordinate clause boundary patterns

10.5 Training on text data

10.6 Training on speech data

10.7 Using Alterf at runtime

Part IV

Speech Processing

11

Speech

Manny Rayner

11.1 Overview of speech functionality

11.2 Config files for speech

11.3 Compiling Regulus into GSL

11.4 Compiling GSL into recognition packages

11.5 Probabilistic tuning of recognition packages

11.6 Recorded wavfile output

11.7 Speech input from the command-line

11.8 Speech output from the command-line

11.9 A Regulus-friendly interface to `batchrec`

11.10 A Regulus-friendly interface to `nl-tool`

12

Using Nuance 9

Manny Rayner

12.1 Overview of Nuance 9

12.2 Making your grammars non-recursive

12.3 Strcat semantics

13

Statistical Recognition and Intelligent Help

Manny Rayner

13.1 Overview of statistical recognition and help
13.2 Compiling an SLM
13.3 Defining Regulus-based SLM classes
13.4 Grammar-based generation of SLM training data
13.5 Building a help system
13.6 Defining help classes
13.7 Testing a help system from the command-line
13.8 Using a help system in an application

Part V

Spoken Dialogue

14

Basic Dialogue Processing

Manny Rayner

14.1 Using dialogue mode

14.2 A toy dialogue system

14.3 Writing an input manager

14.4 Writing a dialogue manager

14.5 Writing an output manager

14.6 Config files for dialogue systems

15

Advanced Dialogue Processing

Manny Rayner

15.1 Writing `lf_pattern` rules

15.2 Implementing ellipsis resolution rules

15.3 Implementing reference resolution rules

15.4 Paraphrasing from the dialogue move

15.5 Handling non-speech inputs

15.6 Setting context in dialogue

15.7 Asynchronous dialogue systems

15.8 Open mic dialogue systems

15.9 The Regulus dialogue server

15.10 Regression testing for dialogue systems

15.11 Regression testing for multi-modal systems

15.12 Using N-best processing in dialogue

Part VI

Translation

16

Basic Translation Systems

Manny Rayner

16.1 A toy translation system

16.2 Config files for translation

16.3 Using translation mode

16.4 Simple translation rules

16.5 Conditional translation rules

16.6 Tracing translation

16.7 Transfer variables

16.8 Macros in translation rules

16.9 Reversible translation rules

16.10 Using interlingua

16.11 Running regression tests in text mode

16.12 Getting timing figures for regression tests

16.13 Running regression tests in speech mode

16.14 Using paraphrases in regression testing

16.15 Re-running regression tests in speech mode

16.16 Splitting corpora into in- and out-of-coverage

16.17 Judging regression tests (Prolog format)

16.18 Judging regression tests (CSV format)

16.19 Comparing regression test results

16.20 Summary of translation commands

17

Advanced Translation Systems

Manny Rayner

17.1 Interlingua checking grammars

17.2 Interlingua debugging

17.3 Interlingua centred development

17.4 Backtranslation

17.5 Almost Flat Functional Semantics

17.6 Simple `role_transfer_rules`

17.7 Conditional `role_transfer_rules`

17.8 `role_list_transfer_rules`

17.9 Generating native script

17.10 Generating gloss translations

17.11 The translation server

17.12 Translating ellipsis

17.13 Answer ellipsis

17.14 Learning translation rules from examples

17.15 Using bidirectional translation mode

17.16 Regression testing for bidirectional translation

17.17 Using N-best processing in translation

Part VII

Command reference

18

Regulus Commands

Manny Rayner

18.1 Overview of Commands

18.2 ANSWER_ELLIPSIS_OFF

[Switch off answer ellipsis (default).]

The converse of ANSWER_ELLIPSIS_ON. Relevant to bidirectional translation applications.

See Section 17.13.

18.3 ANSWER_ELLIPSIS_ON

[Switch on answer ellipsis.]

In bidirectional translation applications, it is possible to use “answer ellipsis”: one user asks a question, and non-sentential responses are treated as ellipsis. For example, in an English-to-French translator, the question “Where is the pain?” might be translated as “Où avez-vous mal?” Then, if answer ellipsis is switched on and there are suitable ellipsis declarations loaded, “le soir” might be interpreted as “J’ai mal le soir”.

See Section 17.13.

18.4 BATCH_DIALOGUE Arg1

[Process dialogue corpus with specified ID.]

Parameterised version of BATCH_DIALOGUE. Process the default dialogue mode development corpus, defined by the dialogue_corpus(⟨Arg⟩) config file entry. The output file, defined by the dialogue_corpus_results(⟨Arg⟩) config file entry, contains question marks for dialogue processing steps that have not yet been judged. If these are replaced by valid judgements, currently ’good’, or ’bad’, the new judgements can be incorporated into the dialogue judgements file (defined by the dialogue_corpus_judgements config file entry) using the command UPDATE_DIALOGUE_JUDGEMENTS ⟨Arg⟩.

See Section 15.10.

18.5 BATCH_DIALOGUE

[Process dialogue corpus.]

Process the default dialogue mode development corpus, defined by the dialogue_corpus config file entry. The output file, defined by the dialogue_corpus_results config file entry, contains question marks for dialogue processing steps that have not yet been judged. If these are replaced by valid judgements, currently ’good’, or ’bad’, the new judgements can be incorporated into the dialogue judgements file (defined by the dialogue_corpus_judgements config file entry) using the command UPDATE_DIALOGUE_JUDGEMENTS.

See Section 15.10.

18.6 BATCH_DIALOGUE_SPEECH Arg1

[Process dialogue speech corpus with specified ID.]

Parameterised speech mode version of BATCH_DIALOGUE. Process the default dialogue mode speech corpus, defined by the dialogue_corpus(⟨Arg⟩) config file entry. The output file, defined by the dialogue_speech_corpus_results(⟨Arg⟩) config file entry, contains question marks for dialogue processing steps that have not yet been judged. If these are replaced by valid judgements, currently 'good', or 'bad', the new judgements can be incorporated into the dialogue judgements file (defined by the dialogue_corpus_judgements config file entry) using the command UPDATE_DIALOGUE_JUDGEMENTS_SPEECH ⟨Arg⟩.

See Section 15.10.

18.7 BATCH_DIALOGUE_SPEECH

[Process dialogue speech corpus.]

Speech mode version of BATCH_DIALOGUE. Process the default dialogue mode speech corpus, defined by the dialogue_speech_corpus config file entry. The output file, defined by the dialogue_speechcorpus_results config file entry, contains question marks for dialogue processing steps that have not yet been judged. If these are replaced by valid judgements, currently 'good', or 'bad', the new judgements can be incorporated into the dialogue judgements file (defined by the dialogue_corpus_judgements config file entry) using the command UPDATE_DIALOGUE_JUDGEMENTS_SPEECH.

See Section 15.10.

18.8 BATCH_DIALOGUE_SPEECH_AGAIN Arg1

[Process dialogue speech corpus with specified ID, using recognition results from previous run.]

Like `BATCH_DIALOGUE_SPEECH_AGAIN`, but uses the version of the speech corpus tagged `Arg1`. Input is taken from the transcriptions file specified by the config parameter

```
dialogue_speech_corpus(Arg1)
```

and output is written to the file specified by the config parameter

```
dialogue_speech_corpus_results(Arg1)
```

The config file also needs to define all the entries associated with spoken dialogue applications.

See Section 15.10.

18.9 BATCH_DIALOGUE_SPEECH_AGAIN

[Process dialogue speech corpus, using recognition results from previous run.]

Version of BATCH_DIALOGUE_SPEECH that skips the speech recognition stage, and instead uses stored results from the previous run.

See Section 15.10.

18.10 BATCH_HELP Arg1

[Process help corpus with specified ID.]

Like BATCH_HELP, but input is taken from the file defined by

```
help_corpus(Arg1)
```

and output is written to the file defined by

```
help_corpus_results(Arg1)
```

See Section 13.7.

18.11 BATCH_HELP

[Process help corpus.]

Relevant to applications using targeted help. The corpus defined by the config file parameter

```
help_corpus
```

which should be in `sent(...)` form, is passed through help processing, and the results are written out to the file defined by config file parameter

```
help_corpus_results
```

Help resources must be defined and loaded.

See Section 13.7.

18.12 BIDIRECTIONAL_OFF

[Switch off bidirectional mode (default).]

Relevant to bidirectional translation applications: switches off the mode switched on by BIDIRECTIONAL_ON.

See Section 17.15.

18.13 BIDIRECTIONAL_ON

[Switch on bidirectional_mode.]

Relevant to bidirectional translation applications: switches on a mode where input can be passed to either side of the bidirectional system. Commands for the “question” side are prefaced with “Q:”, e.g.

```
Q: LOAD_TRANSLATE

Q: where is the pain
```

while commands for the 'answer" side are prefaced with "A:", e.g.

```
A: EBL_LOAD

Q: en la cabeza
```

See Section 17.15.

18.14 CAT Arg1

[Display information for specified category.]

When doing grammar development, this command lets you display the list of features associated with a given syntactic category. It requires a grammar to be loaded. Here is an example with the English grammar:

```
>> CAT np
(Display information for specified category)

Features for category "np": [agr,case,conj,def,gapsin,gapsout,
nform,pronoun,sem,sem_n_type,takes_attrib_pp,takes_frequency_pp,
takes_loc_pp,takes_partitive,takes_post_mods,takes_to_pp,
takes_with_pp,wh]
```

See Section 5.9.2.

18.15 CHECK_ALTERF_PATTERNS

[Check the consistency of the current Alterf patterns file.]

Check the consistency of the current Alterf patterns file, defined by the `alterf_patterns_file config` file entry. Records in the file should have the format

```
alterf_pattern(<Pattern>, <Atom>, <Sent>).
```

or

```
alterf_pattern(<Pattern>, <Atom>, <Sent>) :- <Conds>.
```

where `<Pattern>` is the Alterf pattern, `<Atom>` is the semantic atom it corresponds to, `<Sent>` is an example sentence illustrating the patterns, and `<Conds>` are optional Prolog conditions.

The command parses each `<Sent>` using the currently loaded grammar, and checks that the `<Pattern>` matches it. It warns about patterns that fail to match, and print summary statistics.

See Section 15.1.

18.16 CHECK_BACKTRANSLATION Arg1

[Process Lang -⟩ Lang output in Lang -⟩ Int environment to check that back-translations parse.]

Command relevant to interlingua-based translation applications which use backtranslation (cf. Section 17.4). The assumption is that translation is from Source to Interlingua, and backtranslation is thus from Interlingua to Source.

The argument to the command should be an output file produced by doing `TRANSLATE_CORPUS` (cf. Section 18.114) in the Source → Source environment. The command is however run in the Source → Interlingua environment. The intent is to check that the result of backtranslation is an expression which, when parsed in the Source language and translated back into Interlingua, would produce the same Interlingua representation as the one produced by performing the Source → Source translation. Examples which fail to give a match are flagged.

See Section 16.11.

18.17 CHECK_PARAPHRASES

[Check that transcription paraphrases are in coverage.]

Command used for regression testing in speech translation applications, when employing a paraphrase file (cf. Section 16.14). There will typically be some out-of-coverage utterances which are still close enough that they will successfully go through speech understanding. However, since the transcription is out-of-coverage, the scoring routines will have no way to know that the result is incorrect, since the transcription produces no reference output to compare with.

Under these circumstances, it is possible to declare a *paraphrase file*, which associates in-coverage sentences with out-of-coverage transcriptions. A paraphrase file record has the following format:

```
paraphrase(<TranscriptAtom>, <ParaphraseAtom>)
```

where `<TranscriptAtom>` is the transcription and `<ParaphraseAtom>` is the paraphrase.

The command `CHECK_PARAPHRASES` assumes that a grammar is loaded. It parses the `<ParaphraseAtom>` fields in the paraphrase file, and checks that they are all in coverage, issuing warnings for the ones that are not.

See Section 16.14.

18.18 CLOSE_DOWN_RECOGNITION

[Close down recognition resources: license manager, recserver, TTS and regserver.]

If you are doing recognition from the Regulus command-line (cf. Section 4.4), you may want to use this command to close down the relevant processes after you are finished. This command lets you do it, though it currently only works under Windows/Cygwin.

See Section 11.7.

18.19 COMPACTION

[Switch on compaction processing for Regulus to Nuance conversion (default).]

Thie command is mostly included for historical reasons. You should not want to switch off compaction under normal circumstances.

See Section 11.3.

18.20 COMPILE_ELLIPSIS_PATTERNS

[Compile patterns used for ellipsis processing.]

Compile the patterns used for ellipsis processing, which are defined by the ellipsis_classes config file entry. The compiled patterns will be loaded next time you invoke LOAD_TRANSLATE.

See Section 17.12.

18.21 COMPILE_HELP

[Compile material for targeted help.]

This command compiles run-time targeted help resources for a specific language pair in a speech translation application. It expects the following config file parameters to be defined:

- `targeted_help_source_files` A list of the form

  ```
  [use_combined_interlingua_corpus(<SourceLang>, <TargetLang>),
   <CombinedInterlinguaCorpus>]
  ```

  where *langle*`CombinedInterlinguaCorpus`⟩ is a combined interlingua corpus (cf. 17.3) and *langle*`SourceLang`⟩, *langle*`TargetLang`⟩ are identifiers for the source and target languages. The combined interlingua corpus must be created first, and contain the relevant languages.
- `targeted_help_classes_file` A file of targeted help class declarations for the source language.

See Section 13.5.

18.22 DCG

[Use DCG parser.]

The grammar can be parsed using either the left-corner parser (the default) or the DCG parser. The left-corner parser is faster, but the DCG parser can be useful for debugging. In particular, it can be used to parse non-top constituents ; the left-corner parser lacks this capability.

See Section 6.3.

18.23 DIALOGUE

[Do dialogue-style processing on input sentences.]

In this mode, the sentence is parsed using the current parser. If any parses are found, the first one is processed through the code defined by the dialogue_files config file entry.

See Section 4.1.

18.24 DIALOGUE_SPEAKER

[Show notional speaker (if any) currently being used for dialogue processing.]

In dialogue processing applications where words like “I” and “me” are used, it can be important to know the identity of the speaker; for example, in the Calendar application, one could say “When is my next meeting?” or “Am I attending a meeting on Friday?” The `DIALOGUE_SPEAKER` command makes it possible to print the identity of the notional speaker from the command-line. The notional speaker can also be retrieved using the predicate `get_notional_speaker/1` in `PrologLib/utilities.pl`.

See Section 15.6.

18.25 DIALOGUE_TIME

[Show time (notional or real) currently being used for dialogue processing.]

In dialogue processing applications where expressions like “today” or “next week” are used, it is necessary to know the current time. When doing regression testing, it is then useful to be able to set a notional time, so that responses to time-dependent utterances stay stable.

The `DIALOGUE_TIME` command makes it possible to print the notional time from the command-line. The notional time can also be retrieved using the predicate `get_notional_time/1` in `PrologLib/utilities.pl`.

See Section 15.6.

18.26 DOC Arg1

[Print documentation for command or config file entry.]

The DOC command prints out detailed documentation for a command or config file entry, when this is available. For example

```
DOC DIALOGUE_TIME
```

See Section **??**.

18.27 DUMP_NBEST_TRAINING_DATA_OFF

[Don't write out training data when doing batch processing of N-best results (default).]

When doing batch speech translation, using TRANSLATE_SPEECH_CORPUS and allied commands, it is optionally possible to write out N-best training data if the config file parameter nbest_training_data_file is defined. Output is written to the file in question. This command turns off the behaviour.

See Section 15.12.

18.28 DUMP_NBEST_TRAINING_DATA_ON

[Write out training data when doing batch processing of N-best results.]

When doing batch speech translation, using TRANSLATE_SPEECH_CORPUS and allied commands, it is optionally possible to write out N-best training data if the config file parameter nbest_training_data_file is defined. Output is written to the file in question. This command turns on the behaviour.

See Section 15.12.

18.29 EBL

[Do main EBL processing: equivalent to LOAD, EBL_TREEBANK, EBL_TRAIN, EBL_POSTPROCESS, EBL_NUANCE.]

This command does all the processing needed to build a specialised Nuance grammar from scratch. You will need to have defined at least the following config file parameters:

- ebl_corpus The training corpus, which should consist of Prolog records of the form sent(...).
- ebl_operationality The file defining the operationality criteria.
- ebl_nuance_grammar The output file which will contains the final Nuance grammar.

In many applications, you will also want the following:

- ebl_include_lex A file of "include-lex" definitions, which specify lexical entries to be included directly.

- `ebl_regulus_component_grammar` The specialised grammar that will be loaded by the `EBL_LOAD` command, if it is not the default one.
- `ebl_ignore_feats` Features to ignore when performing Regulus-to-Nuance compilation on the specialised grammar.

18.30 EBL_ANALYSIS

[Do main EBL processing, except for creation of Nuance grammar: equivalent to LOAD, EBL_TREEBANK, EBL_TRAIN, EBL_POSTPROCESS.]

This command does all the processing needed to build a specialised Regulus grammar from scratch. You will need to have defined at least the following config file parameters:

- `ebl_corpus` The training corpus, which should consist of Prolog records of the form `sent(...)`.
- `ebl_operationality` The file defining the operationality criteria.

In many applications, you will also want the following:

- `ebl_include_lex` A file of “include-lex” definitions, which specify lexical entries to be included directly.
- `ebl_regulus_component_grammar` The specialised grammar that will be loaded by the `EBL_LOAD` command, if it is not the default one.

See Section 7.2.

18.31 EBL_GEMINI

[Compile current specialised Regulus grammar into Gemini form.]

Compile current specialised Regulus grammar into Gemini form. Same as the `GEMINI` command, but for the specialised grammar. The base name of the Gemini files produced is defined by the `ebl_gemini_grammar` config file entry.

See Section 5.11.

18.32 EBL_GENERATION

[Do main generation EBL processing: equivalent to LOAD, EBL_TREEBANK, EBL_TRAIN, EBL_POSTPROCESS, EBL_LOAD_GENERATION.]

This command does all the processing needed to build a specialised Nuance generation grammar from scratch. You will need to have defined at least the following config file parameters:

- `ebl_corpus` The training corpus, which should consist of Prolog records of the form `sent(...)`.

- ebl_operationality The file defining the operationality criteria.
- generation_grammar The output file which will contain the generation grammar.

In many applications, you will also want the following:

- ebl_include_lex A file of "include-lex" definitions, which specify lexical entries to be included directly.
- ebl_regulus_component_grammar The specialised grammar that will be loaded by the EBL_LOAD command, if it is not the default one.
- generation_incremental_deepening_parameters Settings to control incremental deepening during generation. A typical value is [0, 50, 50].

See Section 7.2.

18.33 EBL_GRAMMAR_PROBS

[Create Nuance grammar probs training set from current EBL training set or grammar_probs_data file.]

Convert the current EBL training set, defined by the ebl_corpus config file entry, into a form that can be used as training data by the Nuance compute-grammar-probs utility. The output training data is placed in the file defined by the ebl_grammar_probs config file entry.

See Section 11.5.

18.34 EBL_LOAD

[Load current specialised Regulus grammar in DCG and left-corner form.]

Load current specialised Regulus grammar in DCG and left-corner form. Same as the LOAD command, but for the specialised grammar.

The specialised grammar is taken from the setting of the ebl_regulus_component_grammar if it is defined.

See Section 7.3.

18.35 EBL_LOAD_GENERATION Arg1

[Compile and load designated version of current specialised Regulus grammar for generation.]

Parameterised version of EBL_LOAD_GENERATION: compile and load the specialised generation grammar for the subdomain tag ⟨SubdomainTag⟩. This will be the file ⟨prefix⟩_specialised_no_binarise_⟨SubdomainTag⟩.regulus, where ⟨prefix⟩ is the value of the config file entry working_file_prefix. The resulting compiled generation grammar is placed in the file defined by the `generation_grammar(⟨SubdomainTag⟩)` config file entry.

Note that `EBL_LOAD_GENERATION ⟨SubdomainTag⟩` places the compiled generation grammar in the same place as `LOAD_GENERATION ⟨SubdomainTag⟩`.

See Section 7.5.

18.36 EBL_LOAD_GENERATION

[Compile and load current specialised Regulus grammar for generation.]

Compile and load the current specialised generation grammar. This will be the file `⟨prefix⟩_specialised_no_binarise_default.regulus`, where `⟨prefix⟩` is the value of the config file entry `working_file_prefix`. The resulting compiled generation grammar is placed in the file defined by the `generation_rules` config file entry.

Note that `EBL_LOAD_GENERATION` places the compiled generation grammar in the same place as LOAD_GENERATION.

See Section 7.5.

18.37 EBL_MODE

[Do EBL processing on input sentences.]

Put the top-loop in a mode where it shows the results of doing EBL-based processing on input sentences. For this to make sense, you need to have loaded a general grammar and have a config file entry for `ebl_operationality`.

When the top loop is in EBL mode, input sentences are parsed and then subjected to EBL generalisation, using the rules in `ebl_operationality`. The file is reloaded each time. The learned rules are printed in schematic form, as they are in the files output by the EBL_POSTPROCESS and related commands. Each rule is paired with the phrase used to induce it.

See Section 4.1.

18.38 EBL_NUANCE

[Compile current specialised Regulus grammar into Nuance GSL form.]

Compile current specialised Regulus grammar into Nuance GSL form. Same as the `NUANCE` command, but for the specialised grammar. The input is the file created by the `EBL_POSTPROCESS command`; the output Nuance GSL grammar is placed in the file defined by the `ebl_nuance_grammar` config file entry.

See Section 7.4.

18.39 EBL_POSTPROCESS

[Postprocess results of EBL training into specialised Regulus grammar.]

Create one or more specialised Regulus grammars out of the results produced by the EBL_TRAIN command. The grammars are created in two forms. The file `⟨prefix⟩_specialised_no_binarise_⟨tag⟩.regulus` is the original one; the file `⟨prefix⟩_specialised_⟨tag⟩.regulus` has been subjected to a binarisation transformation, so that no rule has more than two daughters. The binarised version is the one passed to the `EBL_NUANCE` command, and is also the default file loaded by `EBL_LOAD`.

See Section 7.2.

18.40 EBL_TRAIN

[Do EBL training on current treebank.]

Takes the file built using the `EBL_TREEBANK` command and performs EBL generalisation using operationality criteria defined by `ebl_operationality`. The output needs to be processed further by the `EBL_POSTPROCESS` command.

See Section 7.2.

18.41 EBL_TREEBANK

[Parse all sentences in current EBL training set into treebank form.]

Parse all sentences in current EBL training set, defined by the `ebl_corpus` config file entry, to create a treebank file. Sentences that fail to parse are printed out with warning messages, and a summary statistic is produced at the end of the run. This is very useful for checking where you are with coverage.

See Section 7.2.

18.42 ECHO_OFF

[Don't echo input sentences (default).]

Switch off functionality to echo input text.

See Section 3.8.

18.43 ECHO_ON

[Echo input sentences (normally useful only in batch mode).]

Switch on echoing of input text. In this mode, each input text string (as opposed to Regulus command) is printed out before being processed. This is normally only useful when running in batch mode.

See Section 3.8.

18.44 FEAT Arg1

[Display information for specified feature.]

If the argument is a feature in the currently loaded grammar, print information showing the range of permitted values for that feature. For example:

```
>> FEAT agr
(Display information for specified feature)

Feature values for feature "agr": [[1,2,3],[masc,fem],[sg,pl]]
```

See Section 5.9.2.

18.45 GEMINI

[Compile current Regulus grammar into Gemini form.]

The grammar defined by the parameter `regulus_grammar` is translated into Gemini form. The base name for the output grammar file needs to be specified using the parameter `gemini_grammar`.

See Section 5.11.

18.46 GENERATE_TRACE_OFF

[Switch off generation tracing (default).]

In translation mode, switch off printing of the generation trace.

See Section 6.12.

18.47 GENERATE_TRACE_ON

[Switch on generation tracing.]

In translation mode, switch on printing of generation tracing. Each example processed prints the tree for the generated target, together with preference information. If multiple target sentences are produced, the tree and preference information is printed for each one.

See Section 6.12.

18.48 GENERATION

[Generate from parsed input sentences.]

Run the system in “generation mode”. Each input sentence is analysed. If any parses are found, the first one is generated back using the currently loaded generation grammar, showing all possible generated strings. This is normally used for debugging the generation grammar.

See Section 4.1.

18.49 HELP Arg1

[Print help for commands whose name or description match the string.]

The argument is matched against all commands and their short descriptions, and matching examples are displayed. Matching is not case-sensitive.

Example:

```
>> HELP trace
(Print help for commands whose name or description match the string)

6 commands matching "trace":

GENERATE_TRACE_OFF (Switch off generation tracing (default))
GENERATE_TRACE_ON (Switch on generation tracing)
INTERLINGUA_TRACE_OFF (Switch off interlingua tracing (default))
INTERLINGUA_TRACE_ON (Switch on interlingua tracing)
TRANSLATE_TRACE_OFF (Switch off translation tracing (default))
TRANSLATE_TRACE_ON (Switch on translation tracing)
```

See Section 4.6.

18.50 HELP

[Print help for all commands.]

Print all available commands, together with short descriptions of what they do. It is usually preferable to restrict the search by giving an argument to HELP, since there are many commands.

See Section 4.6.

18.51 HELP_CONFIG Arg1

[Print help for config file entries whose name or description match the string.]

Search for config file entries whose name matches the argument and display them. Matching is not case-sensitive.

Example:

```
>> HELP_CONFIG tree
(Print help for config file entries whose name or description match the

3 config file entries matching "tree":

alterf_treebank_file
ebl_treebank
ellipsis_classes_treebank_file
```

See Section 4.6.

18.52 HELP_RESPONSE_OFF

[Switch off help response in main loop (default off).]

Switch off help processing for top-level inputs. This only makes sense if help resources are loaded.

See Section 13.7.

18.53 HELP_RESPONSE_ON

[Switch on help response in main loop (default off).]

Switch on help processing for top-level inputs. This only makes sense if help resources have been loaded using the `LOAD_HELP` command. In that case, help processing is applied first, and the top 5 help responses are printed together with trace information. After that, normal processing is carried out.

See Section 13.7.

18.54 INCREMENTAL_TREEBANKING_OFF

[Don't try to reuse old treebank material (default on).]

Invoking this command forces the `EBL_TREEBANK` command to reparse the whole of the training corpus. By default, it attempts to reuse old analyses when it considers that they should be reliable.

See Section **??**.

18.55 INCREMENTAL_TREEBANKING_ON

[Try to reuse old treebank material when possible (default on).]

When incremental treebanking is on (default), the `EBL_TREEBANK` command attempts to reuse stored analyses of corpus examples rather than parsing them again. It considers an analysis of a sentence S safe a) if the only grammar rules that have changed since the stored parse was created are lexical ones involving words not occuring in S, b) the config file and the files it includes have not changed, c) the analysis preferences have not changed.

See Section **??**.

18.56 INIT_DIALOGUE Arg1

[Initialise the dialogue state, passing it the given argument.]

In dialogue mode, initialises the dialogue state, passing it the specified argument. For this to make sense, you need to have previously invoked the `LOAD_DIALOGUE` command, and the predicate `initial_dialogue_state/2` needs to be defined. This predicate will be called to obtain the new dialogue state and perform any relevant initialisation. The argument to INIT_DIALOGUE is passed as the first argument, and the state is returned as the second argument.

See Section 14.1.

18.57 INIT_DIALOGUE

[Initialise the dialogue state.]

In dialogue mode, initialises the dialogue state. For this to make sense, you need to have previously invoked the LOAD_DIALOGUE command, and the predicate `initial_dialogue_state/1` needs to be defined. This predicate will be called to obtain the new dialogue state.

See Section 14.1.

18.58 INTERLINGUA

[Perform translation through interlingua.]

Do translation through interlingua, i.e. by first applying source-to-interlingua rules (from the file that `to_interlingua_rules` points to) and then interlingua-to-target rules ((from the file that `from_interlingua_rules` points to). This applies both to interactive processing in translate mode, and to batch processing using commands like TRANSLATE_CORPUS, TRANSLATE_SPEECH_CORPUS and TRANSLATE_SPEECH_CORPUS_AGAIN.

See Section 16.10.

18.59 INTERLINGUA_DEBUGGING_OFF

[Switch off interlingua debugging (default).]

Converse of INTERLINGUA_DEBUGGING_ON.

See Section 17.2.

18.60 INTERLINGUA_DEBUGGING_ON

[Switch on interlingua debugging.]

Switch on interlingua debugging; relevant to translation applications that use an interlingua checking grammar. In this mode, translations that give rise to interlingua that is ill-formed according to the interlingua checking grammar are processed by subjecting the ill-formed interlingua to all possible insertions, deletions and substitutions of a single interlingua element, until either a well-formed variant is discovered or a timeout is exceeded. The first well-formed variant found is displayed.

See Section 17.2.

18.61 INTERLINGUA_TRACE_OFF

[Switch off interlingua tracing (default).]

Converse of INTERLINGUA_TRACE_ON.

18.62 INTERLINGUA_TRACE_ON

[Switch on interlingua tracing.]

Relevant to translation applications using an interlingua checking grammar. If interlingua checking succeeds, print the interlingua checking grammar's analysis tree.

18.63 KILL_NUANCE_PARSERS

[Kill any outstanding nl-tool processes (may be necessary after doing NUANCE_PARSER).]

Every time you invoke the NUANCE_PARSER command, it starts up a new nl-tool process. This command allows you to shut down all outstanding nl-tool processes.

See Section 6.5.

18.64 LC

[Use left-corner parser.]

Sets the current parser back to the default left-corner parser. This is normally used after invoking the DCG or NUANCE_PARSER commands.

See Section 6.2.

18.65 LF_POST_PROCESSING_OFF

[Switch off semantic post-processing of LFs.]

The grammar processing mechanism currently supports three main types of semantics: linear, Almost Flat Functional (AFF) and RIACS. For AFF and RIACS semantics, the original logical form produced by the grammar needs to be post-processed.

This command switches off post-processing of LFs, making it possible to examine the original logical form, and is in particular useful when you suspect a post-processing bug.

See Section 5.9.

18.66 LF_POST_PROCESSING_ON

[Switch on semantic post-processing of LFs (default).]

Converse of LF_POST_PROCESSING_OFF.

See Section 5.9.

18.67 LINE_INFO_OFF

[Don't print line and file info for rules and lex entries in parse trees.]

A typical parse tree printed without line info will look like this:

```
.MAIN
   utterance
      command
      /  verb lex(switch)
      |  onoff null lex(on)
      |  np
      |  /  lex(the)
      |  |  noun lex(light)
      \  \  null
```

See Section 6.6.

18.68 LINE_INFO_ON

[Print line and file info for rules and lex entries in parse trees (default).]

A typical parse tree printed with line info will look like this:

```
.MAIN [TOY1_RULES:1-4]
   utterance [TOY1_RULES:5-9]
      command [TOY1_RULES:10-14]
      /  verb lex(switch) [TOY1_LEXICON:8-10]
      |  onoff null lex(on) [TOY1_LEXICON:24-25]
      |  np [TOY1_RULES:25-29]
      |  /  lex(the)
      |  |  noun lex(light) [TOY1_LEXICON:16-17]
      \  \  null

------------------------------- FILES -------------------------------

TOY1_LEXICON: d:/regulus/examples/toy1/regulus/toy1_lexicon.regulus
TOY1_RULES:   d:/regulus/examples/toy1/regulus/toy1_rules.regulus
```

See Section 6.6.

18.69 LIST_MISSING_HELP_DECLARATIONS

[Write out a list of lexical items that are not listed in targeted help declarations.]

Relevant to applications that use targeted help; a grammar must be loaded, and the parameters `targeted_help_classes_file` and `missing_help_class_decls` need to be defined. The help classes file is searched, and all lexical items in the grammar that are not defined are listed in the missing decls file.

See Section 13.6.

18.70 LOAD

[Load current Regulus grammar in DCG and left-corner form.]

Compile and load the Regulus grammar defined by the `regulus_grammar` config file entry in DCG and left-corner form. If the grammar files and the config file have not been modified since the last invocation of the `LOAD` command, left-corner compilation is not performed, and the stored version of the compiled grammar is used.

If parse preferences and/or nbest preference files are defined, these are also loaded. These files are specified by the parameters `parse_preferences` and `nbest_preferences` respectively, and can be also loaded using the `LOAD_PREFERENCES` command.

See Section 5.1.

18.71 LOAD_DEBUG

[Load current Regulus grammar in DCG and left-corner form, including extra debugging rules in left-corner grammar.]

Compile and load the Regulus grammar defined by the `regulus_grammar` config file entry in DCG and left-corner form, including extra rules useful for grammar debugging. This makes parsing slightly slower.

When the grammar is loaded in this form, a top-level input of the form

```
<CategoryName> <Sentence>
```

is treated as a request to parse ⟨`Sentence`⟩ as an instance of ⟨`CategoryName`⟩, printing out semantic and feature values. 1 shows an example using the Toy1 grammar.

See Section 5.1.

18.72 LOAD_DIALOGUE

[Load dialogue-related files.]

Relevant to dialogue applications: compile the files defined by the `dialogue_files` config file entry. These should at a minimum define the predicates `lf_to_dialogue_move`, `initial_dialogue_state`, `update_dialogue_state` and `abstract_action_to_action` with appropriate arities.

See Section 14.1.

18.73 LOAD_GENERATION Arg1

[Compile and load current generator grammar, and store as designated

```
>> LOAD_DEBUG

(...)

>> np the light
(Parsing with left-corner parser)

Analysis time: 0.00 seconds

Return value: [[device,light]]

Global value: []

Syn features: [sem_np_type=switchable\/dimmable,singplur=sing]

Parse tree:

np [TOY1_RULES:25-29]
/  lex(the)
\  noun lex(light) [TOY1_LEXICON:16-17]

------------------------------- FILES -------------------------------

TOY1_LEXICON: d:/regulus/examples/toy1/regulus/toy1_lexicon.regulus
TOY1_RULES:   d:/regulus/examples/toy1/regulus/toy1_rules.regulus
```

FIGURE 1 Example showing use of `LOAD_DEBUG`

subdomain grammar.]

Compile and load the current generation grammar, defined by the `generation_grammar` config file entry. The resulting compiled generation grammar is placed in the file defined by the `generation_grammar(⟨Arg⟩)` config file entry. This can be useful if you are normally using grammar specialisation to build the generation grammar.

See Section 6.8.

18.74 LOAD_GENERATION

[Compile and load current generator grammar.]

Compile and load the current generation grammar, defined by the `regulus_grammar` or `generation_regulus_grammar` config file entry. The resulting compiled generation grammar is placed in the file defined by the `generation_grammar` config file entry.

See Section 6.8.

18.75 LOAD_HELP

[Load compiled material for targeted help.]

Relevant to applications using targeted help. Loads the help resources previously build by invoking the COMPILE_HELP command.

See Section 13.7.

18.76 LOAD_PREFERENCES

[Load parse and N-best preference files.]

Load parse preferences and/or nbest preference files if they are defined. These files are specified by the parameters parse_preferences and nbest_preferences respectively.

See Section 6.7.

18.77 LOAD_RECOGNITION Arg1

[Load recognition resources: license manager, recserver, TTS and regserver, using specified port for Regserver.]

See Section 11.7.

18.78 LOAD_RECOGNITION

[Load recognition resources: license manager, recserver, TTS and regserver.]

Start Nuance speech resources to enable speech processing from the Regulus command-line. The following are required:

- The file $REGULUS/scripts/run_license.bat needs to exist and contain a valid invocation of the license manager. Typical contents (not a real licence code) might be

  ```
  nlm C:/Nuance/Vocalizer4.0/license.txt abc12-1234-a-ab12
  ```

- One of the parameters translation_rec_params and dialogue_rec_params needs to exist, and have an appropriate value which specifies the recognition package to use and the Nuance parameters to pass the recogniser invocation. A typical value might be

  ```
  [package=callslt_runtime(recogniser), grammar='.MAIN',
  'rec.Pruning=1600', 'rec.DoNBest=TRUE', 'rec.NumNBest=6',
  'rec.ConfidenceRejectionThreshold=0',
  'ep.EndSeconds=1.5']).
  ```

- If the application is to use TTS, an appropriate invocation to start Vocalizer must be supplied as the value of the parameter tts_command. A typical value to start the English version of Vocalizer 4.0 would be

```
'vocalizer -num_channels 1 -voice enhancedlaurie
-voices_from_disk'
```

See Section 11.7.

18.79 LOAD_RECOGNITION_GENERATION

[Compile and load current generator grammar(s) for converting recognition results to other scripts.]

Relevant to applications which use multiple parallel grammars, and need to convert recognition results either to the *original script* or to the *gloss script*. The parallel original script and gloss script grammars first need to be compiled into generation grammar form. They must then be declared using one or both of the parameters `original_script_recognition_generation_rules` and `gloss_recognition_generation_rules`. The command `LOAD_RECOGNITION_GENERATION` loads any parallel generation grammars that may be declared.

See Section 6.8.

18.80 LOAD_SURFACE_PATTERNS

[Load current surface patterns and associated files.]

Relevant to applications which do surface parsing using Alterf; load the Alterf surface pattern files. You can then parse in surface mode using the `SURFACE` command. The following config file entries must be defined:

- `surface_patterns`
- `tagging_grammar`
- `target_model`
- `discriminants`
- `surface_postprocessing`

See Section 10.1.

18.81 LOAD_TRANSLATE

[Load translation-related files.]

Load all translation-related files defined in the currently valid config file. These consist of a subset of the following; the set of files required depends on whether translation is interlingua-based or direct, and whether translation is from source to target, from source to interlingua, or from interlingua to target.

- An interlingua checking grammar compiled into generation form, defined by the `interlingua_structure` config file entry. Required if translation is interlingua-based.

item An interlingua declarations file defined by the `interlingua_declarations` config file entry. Required if translation is interlingua-based.

- One or more to_interlingua rules files defined by the `to_interlingua_rules` config file entry. Required if translation is interlingua-based.
- One or more from_interlingua rules files defined by the `from_interlingua_rules` config file entry. Required if translation is interlingua-based.
- An ellipsis classes file (optional) defined by the `ellipsis_classes` config file entry. If this is defined, you need to compile it first using the `COMPILE_ELLIPSIS_PATTERNS` command.
- A generation grammar file (required, unless translation is from source to interlingua) defined by the `generation_rules` config file entry. This should be the compiled form of a Regulus grammar for the target language. The compiled generation grammar must first be created using the `LOAD_GENERATION` command.
- A generation preferences file (optional) defined by the `generation_preferences` config file entry.
- A collocations file (optional) defined by the `collocation_rules` config file entry.
- An orthography rules file (optional) defined by the `orthography_rules` config file entry.
- One or more transfer rules files defined by the `transfer_rules` config file entry. This is only required for direct (i.e. non-interlingua-based) translation applications.

If the config file entries `wavfile_directory` and `wavfile_recording_script` are defined, implying that output speech will be produced using recorded wavfiles, this command also produces a new version of the file defined by `wavfile_recording_script`.

See Section 16.3.

18.82 `MAKE_TARGET_GRAMMAR_PROBS_CORPUS Arg1`

[Create Nuance grammar probs training set for given grammar from translation output.]

Relevant to translation applications. Take the results held in the file indicated by `translation_corpus_results` and turn them into a training file for Nuance PCFG tuning, using the argument as the relevant Nuance grammar. Put the result in the file indicated by `target_grammar_probs`. Thus, for example, the call

```
MAKE_TARGET_GRAMMAR_PROBS_CORPUS .MAIN
```

will create a file where, for each translation ⟨Sent⟩ in translation_corpus_results, the file target_grammar_probs will contain a record of the form

```
.MAIN <Sent>
```

See Section 11.5.

18.83 MAKE_TARGET_SENT_CORPUS

[Create sent-formatted corpus from translation output.]

Relevant to translation applications. Take the results held in the file indicated by translation_corpus_results and turn them into a sent-formatted file. Put the result in the file indicated by target_sent_corpus. Thus, for example, the call

```
MAKE_TARGET_SENT_CORPUS .MAIN
```

will create a file where, for each translation ⟨Sent⟩ in translation_corpus_results, the file target_sent_corpus will contain a record of the form

```
sent(<Sent>).
```

See Section 11.5.

18.84 NORMAL_PROCESSING

[Do normal processing on input sentences.]

Switch sentence processing in the Regulus top-level back to the default behaviour: the system attempts to parse the sentence using the current parser. If successful, it prints relevant information, in particular the logical form(s), the associated analysis tree(s), and any associated preference information.

See Section 4.1.

18.85 NO_COMPACTION

[Switch off compaction processing for Regulus to Nuance conversion.]

Switch off the grammar compaction step at the end of Regulus-to-Nuance compilation, as for example invoked by the NUANCE command.

You should not normally wish to do this, since compaction is almost always beneficial and is very stable.

See Section 11.3.

18.86 NO_ELLIPSIS_PROCESSING

[Unload any ellipsis processing rules that may be loaded.]

Relevant to translation applications: remove any currently loaded ellipsis rules. These rules will have been compiled by the command COMPILE_ELLIPSIS_PATTERNS and loaded by the command LOAD_TRANSLATE.

See Section 17.12.

18.87 NO_INTERLINGUA

[Perform translation directly, i.e. not through interlingua.]

Applies to translation applications: converse of the command INTERLINGUA, it sets the translation processing mode to perform direct translation from source to target, i.e. not through the interlingua. It follows that the current config file must define a value for the parameter transfer_rules, which should point to a file of direct translation rules.

This applies both to interactive processing when the TRANSLATE command is in effect, and to batch processing using commands like TRANSLATE_CORPUS, TRANSLATE_SPEECH_CORPUS and TRANSLATE_SPEECH_CORPUS_AGAIN.

See Section 16.10.

18.88 NUANCE

[Compile current Regulus grammar into Nuance GSL form.]

Compile current Regulus grammar into Nuance GSL form. You won't be able to use this command in conjunction with a large general grammar, since it currently runs out of memory during compilation — this why we need EBL. The NUANCE command is useful for smaller Regulus grammars, e.g. the Toy1 grammar.

The current Regulus grammar is defined by the regulus_grammar config file entry, and the location of the generated Nuance grammar by the nuance_grammar config file entry.

See Section 11.3.

18.89 NUANCE_COMPILE

[Compile Nuance grammar into recogniser package.]

Compile the generated Nuance grammar, defined by the ebl_nuance_grammar or nuance_grammar config file entry, into a recognition package with the same name. This will be done using the Nuance language pack defined by the nuance_language_pack config file entry and the extra parameters defined by the nuance_compile_params config file entry. Typical values for these parameters are as follows:

```
regulus_config(nuance_language_pack, 'English.America').
regulus_config(nuance_compile_params,
                    ['-auto_pron', '-dont_flatten']).
```

See Section 11.4.

18.90 NUANCE_COMPILE_WITH_PCFG

[Compile Nuance grammar into recogniser package, first doing PCFG training.]

First perform PCFG training on the generated Nuance grammar, defined by the `ebl_nuance_grammar` or `nuance_grammar` config file entry. The training data is taken from the file defined by the `ebl_grammar_probs` config file entry.

Next, compile the PCFG-trained version of the Nuance grammar, produced by the first step, into a recognition package with the same name. This will be done using the Nuance language pack defined by the `nuance_language_pack` config file entry and the extra parameters defined by the `nuance_compile_params` config file entry. Typical values for these parameters are as follows:

```
regulus_config(nuance_language_pack, 'English.America').
regulus_config(nuance_compile_params,
                    ['-auto_pron', '-dont_flatten']).
```

See Section 11.4.

18.91 NUANCE_PARSER

[Start new Nuance nl-tool process and use it as parser.]

Start an `nl-tool` process, and use it to do parsing. Any old `nl-tool` processes are first killed. The current config file needs to include either a `dialogue_rec_params` declaration (for dialogue apps) or a `translation_rec_params` declaration (for speech translation apps); the declaration must contain definitions for `'package'` and `'grammar'`. The following is a typical example of a suitable declaration:

```
regulus_config(dialogue_rec_params,
                    [package=calendar_runtime(recogniser),
                     grammar='.MAIN',
                     'rec.Pruning=1600', 'rec.DoNBest=TRUE',
                     'rec.NumNBest=6']).
```

Notes:

- After `NUANCE_PARSER` is successfully invoked, `nl-tool` is used for ALL parsing, including batch processing with commands like `TRANSLATE_CORPUS` and Prolog calls to `parse_with_current_parser/6`.
- The Nuance parser only returns logical forms, not parse trees.

See Section 6.5.

18.92 PARSE_HISTORY Args

[Show parse history for examples matching specified string.]

Search the parsing history file created by the EBL_MAKE_TREEBANK command to find matching example. The argument is treated as a list of words, which may optionally contain wildcards, and matching is performed at the word (opposed to character) level. Each matching example is printed together with a date-stamp showing when it last produced a parse. Here is a typical invocation:

```
>> PARSE_HISTORY i would * cheese
(Show parse history for examples matching specified string)

--- Read parsing history file (394 records)
d:/call-slt/eng/generatedfiles/callslt_parsing_history.pl

Found 2 records matching pattern

2010-06-01_15-41-03 1 i would like the cheese plate
2010-06-01_15-41-06 1 i would like the macaroni cheese
```

See Section ??.

18.93 PRINT_TREE_CATEGORIES_OFF

[Don't print categories in parse trees (default).]

Converse of PRINT_TREE_CATEGORIES_ON.

See Section 6.6.

18.94 PRINT_TREE_CATEGORIES_ON

[Print categories in parse trees.]

When printing parse trees at top level, also show all the categories in the tree. 2 shows an example using the Toy1 grammar.

See Section 6.6.

18.95 PRINT_TREE_SUMMARY_OFF

[Don't print summary versions of parse trees (default).]

Converse of PRINT_TREE_SUMMARY_ON.

See Section 6.6.

18.96 PRINT_TREE_SUMMARY_ON

[Print summary versions of parse trees.]

```
>> PRINT_TREE_CATEGORIES_ON
(Print categories in parse trees)

--- Performed command PRINT_TREE_CATEGORIES_ON, time = 0.02 seconds

>> switch on the light
(Parsing with left-corner parser)

Analysis time: 0.00 seconds

Return value: [[action,switch],[device,light],
               [onoff,on],[utterance_type,command]]

Global value: []

Syn features: []

Parse tree:

.MAIN [TOY1_RULES:1-4]
   utterance [TOY1_RULES:5-9]
      command [TOY1_RULES:10-14]
      /  verb lex(switch) [TOY1_LEXICON:8-10]
      |  onoff null lex(on) [TOY1_LEXICON:24-25]
      |  np [TOY1_RULES:25-29]
      |  /  lex(the)
      |  |  noun lex(light) [TOY1_LEXICON:16-17]
      \  \  null

------------------------------- FILES -------------------------------

TOY1_LEXICON: d:/regulus/examples/toy1/regulus/toy1_lexicon.regulus
TOY1_RULES:   d:/regulus/examples/toy1/regulus/toy1_rules.regulus

Categories:

[('.MAIN':[]),
 (command:[]),
 (noun:[sem_np_type=switchable,singplur=sing]),
 (np:[sem_np_type=switchable,singplur=sing]),
 (onoff:[]),
 (utterance:[]),
 (verb:[obj_sem_np_type=switchable,singplur=sing,
        vform=imperative,vtype=switch])]
```

FIGURE 2 Example showing use of PRINT_TREE_CATEGORIES_ON

When processing input sentences from the Regulus top-level, print a summary of each parse tree. This is primarily useful if you need to add `tree_includes_structure` or `tree_doesnt_include_structure` constraints in an EBL training corpus. 3 shows an example using the Toy1 grammar.

See Section 6.6.

18.97 RANDOM_GENERATE Arg1 Arg2

[Randomly generate and print the specified number of sentences, with specified maximum depth.]

Like `RANDOM_GENERATE Arg1 Arg2`, but use the second argument to limit the maximum depth of the generated tree.

See Section 6.15.

18.98 RANDOM_GENERATE Arg1

[Randomly generate and print the specified number of sentences.]

Randomly generate valid sentences from the currently loaded grammar. Here is an example using the Toy1 grammar:

```
>> RANDOM_GENERATE 5
(Randomly generate and print the specified number of sentences)
.....
are the fans on
is the fan off
dim the light
switch on the lights
are the lights in the living room in the living room switched off
```

See Section 6.15.

18.99 RECOGNISE

[Take next loop input from live speech.]

Assumed that recognition resources have been loaded using the `LOAD_RECOGNITION` command; uses the current recogniser to perform recognition, then treats the 1-best recognition result as though it had been top-level text input. The `RECOGNISE` command can be used in any top-level mode.

See Section 11.7.

18.100 RELOAD_CFG

[Reload current config file.]

Reload the current Regulus config file, plus any files it may include.

See Section 3.2.

```
>> PRINT_TREE_SUMMARY_ON
(Print summary versions of parse trees)

--- Performed command PRINT_TREE_SUMMARY_ON, time = 0.00 seconds

>> switch on the light
(Parsing with left-corner parser)

Analysis time: 0.00 seconds

Return value: [[action,switch],[device,light],
               [onoff,on],[utterance_type,command]]

Global value: []

Syn features: []

Parse tree:

.MAIN [TOY1_RULES:1-4]
   utterance [TOY1_RULES:5-9]
      command [TOY1_RULES:10-14]
      /  verb lex(switch) [TOY1_LEXICON:8-10]
      |  onoff null lex(on) [TOY1_LEXICON:24-25]
      |  np [TOY1_RULES:25-29]
      |  /  lex(the)
      |  |  noun lex(light) [TOY1_LEXICON:16-17]
      \  \  null

------------------------------- FILES ----------------------------------

TOY1_LEXICON: d:/regulus/examples/toy1/regulus/toy1_lexicon.regulus
TOY1_RULES:   d:/regulus/examples/toy1/regulus/toy1_rules.regulus

Summary:

('.MAIN' <
 [(utterance <
   [command<[verb<lex(switch),
                 onoff<lex(on),
                 np<[lex(the),
                     noun<lex(light),
                     empty_constituent]]])])
```

FIGURE 3 Example showing use of PRINT_TREE_SUMMARY_ON

18.101 SET_BATCH_DIALOGUE_FORMAT Arg1

[Set format for printing batch dialogue results. Default is "normal".]

By default, the output of invoking BATCH_DIALOGUE and similar commands is to print all processing information. This command can be used to select other output formats, or to revert to the default format. The currently supported alternatives are the following:

- normal Default format.
- no_paraphrases Suppress printing of fields related to paraphrasing.
- no_datastructures Suppress printing of all fields related to intermediate datastructures.

See Section ??.

18.102 SET_NBEST_N Arg1

[Set the maximum number of hypotheses used for N-best processing.]

For offline speech processing commands, the parameters translation_rec_params and dialogue_rec_params control, among other things, the number of N-best alternatives generated by Nuance. For example, the value

```
[package=callslt_runtime(recogniser), grammar='.MAIN',
'rec.Pruning=1600', 'rec.DoNBest=TRUE', 'rec.NumNBest=6']).
```

produces a maximum of 6 hypotheses.

This command makes it possible to reduce the number of N-best hyptheses considered by language processing. Evidently, the number cannot be increased.

See Section 15.12.

18.103 SET_NOTIONAL_SPEAKER Arg1

[Set notional name of speaker for dialogue processing..]

In dialogue processing applications where words like "I" and "me" are used, it can be important to know the identity of the speaker; for example, in the Calendar application, one could say "When is my next meeting?" or "Am I attending a meeting on Friday?" When doing regression testing, it is then useful to be able to set a notional speaker, so that responses to time-dependent utterances stay stable.

The SET_NOTIONAL_SPEAKER command makes it possible to set the identity of the notional speaker from the command-line. The argument should be an atom, e.g.

```
SET_NOTIONAL_SPEAKER manny
```

The notional speaker can be retrieved using the predicate `get_notional_time/1` in `PrologLib/utilities.pl`.

See Section 15.6.

18.104 SET_NOTIONAL_TIME Arg1

[Set notional time for dialogue processing. Format = YYYY-MM-DD_-HH-MM-SS, e.g. 2006-12-31_23-59-59.]

In dialogue processing applications where expressions like “today” or “next week” are used, it is necessary to know the current time. When doing regression testing, it is then useful to be able to set a notional time, so that responses to time-dependent utterances stay stable.

The `SET_NOTIONAL_TIME` command makes it possible to set the identity of the notional time from the command-line. The format is YYYY-MM-DD_HH-MM-SS, e.g.

```
SET_NOTIONAL_TIME 2010-08-04_15-17-55
```

The notional time can be retrieved using the predicate `get_notional_time/1` in `PrologLib/utilities.pl`.

See Section 15.6.

18.105 SET_REGSERVER_TIMEOUT Arg1

[Set the time the system waits before starting up the Regserver.]

When recognition resources are started by the `LOAD_RECOGNITION` command, specify the number of seconds to wait before starting the Regserver process. The default value is 60.

See Section 11.7.

18.106 SPLIT_SPEECH_CORPUS Arg1 Arg2 Arg3 Arg4

[Split speech corpus into in-coverage and out-of-coverage pieces with respect to the specified grammar. Arguments: ⟨GrammarAtom⟩, ⟨CorpusId⟩, ⟨InCoverageCorpusId⟩ ⟨OutOfCoverageCorpusId⟩.]

Splits the speech translation corpus output file, defined by the `translation_speech_corpus(⟨Arg2⟩)` config file entry, into

- an in-coverage part defined by a `translation_speech_corpus(⟨Arg3⟩)` config file entry, and
- an out-of-coverage part defined by a `translation_speech_corpus(⟨Arg4⟩)` config file entry.

Coverage is with respect to the top-level grammar `⟨GrammarName⟩` (Arg1), which must be loaded.

Typical call:

```
SPLIT_SPEECH_CORPUS .MAIN corpus2 in_coverage2 out_of_coverage2
```

See Section 16.16.

18.107 SPLIT_SPEECH_CORPUS Arg1 Arg2 Arg3

[Split default speech corpus into in-coverage and out-of-coverage pieces with respect to the specified grammar. Arguments: ⟨GrammarAtom⟩, ⟨InCoverageCorpusId⟩ ⟨OutOfCoverageCorpusId⟩.]

Splits the speech translation corpus output file, defined by the `translation_speech_corpus` config file entry, into

- an in-coverage part defined by a `translation_speech_corpus(⟨Arg2⟩)` config file entry, and
- an out-of-coverage part defined by a `translation_speech_corpus(⟨Arg3⟩)` config file entry.

Coverage is with respect to the top-level grammar ⟨`Arg1`⟩, which must be loaded.

Typical call:

```
SPLIT_SPEECH_CORPUS .MAIN in_coverage out_of_coverage
```

See Section 16.16.

18.108 SPLIT_SPEECH_CORPUS training_corpus Arg1 Arg2 Arg3

[Split speech corpus into in-training and out-of-training pieces with respect to EBL training corpus. Arguments: ⟨FromCorpusId⟩, ⟨InTrainingCorpusId⟩ ⟨OutOfTrainingCorpusId⟩.]

See Section 16.16.

18.109 STEPPER

[Start grammar stepper.]

See Section 5.10.

18.110 STORE_TRANSLATION_TARGET_VOCAB Arg1

[Process Source -⟩ Target output and store target vocabulary items in the predicate regulus_preds:target_vocabulary_item.]

See Section 16.11.

18.111 SURFACE

[Use surface pattern-matching parser.]

This assumes that surface pattern files have been loaded, using the LOAD_SURFACE_PATTERNS command.

See Section 10.1.

18.112 TRANSLATE

[Do translation-style processing on input sentences.]

In this mode, the sentence is parsed using the current parser. If any parses are found, the first one is processed through translation and generation. Translation is performed using interlingual rules if the INTERLINGUA command has been applied, otherwise using direct transfer.

See Section 4.1.

18.113 TRANSLATE_CORPUS Arg1

[Process text translation corpus with specified ID.]

Parameterised version of TRANSLATE_CORPUS. Process the text mode translation corpus with ID ⟨Arg⟩, defined by the parameterised config file entry translation_corpus(⟨Arg⟩). The output file, defined by the parameterised config file entry translation_corpus_results(⟨Arg⟩), contains question marks for translations that have not yet been judged. If these are replaced by valid judgements, currently 'good', 'ok' or 'bad', the new judgements can be incorporated into the translation judgements file (defined by the translation_corpus_judgements config file entry) using the parameterised command UPDATE_TRANSLATION_JUDGEMENTS ⟨Arg⟩.

See Section 16.11.

18.114 TRANSLATE_CORPUS

[Process text translation corpus.]

Process the default text mode translation corpus, defined by the translation_corpus config file entry. The output file, defined by the translation_corpus_results config file entry, contains question marks for translations that have not yet been judged. If these are replaced by valid judgements, currently 'good', 'ok' or 'bad', the new judgements can be incorporated into the translation judgements file (defined by the translation_corpus_judgements config file entry) using the command UPDATE_TRANSLATION_JUDGEMENTS.

See Section 16.11.

18.115 TRANSLATE_PARSE_TIMES Arg1

[Print parse times for latest run on text translation corpus with specified ID.]

See Section 16.12.

18.116 TRANSLATE_PARSE_TIMES

[Print parse times for latest run on text translation corpus.]

See Section 16.12.

18.117 TRANSLATE_SPEECH_CORPUS Arg1

[Process speech translation corpus with specified ID.]

Parameterised version of TRANSLATE_SPEECH_CORPUS. Process speech mode translation corpus, defined by the translation_speech_corpus(⟨Arg⟩) config file entry. The output file, defined by the translation_speech_corpus_results(⟨Arg) config file entry, contains question marks for translations that have not yet been judged. If these are replaced by valid judgements, currently 'good', 'ok' or 'bad', the new judgements can be incorporated into the stored translation judgements file using the command UPDATE_TRANSLATION_JUDGEMENTS_SPEECH ⟨Arg⟩. A second output file, defined by the translation_corpus_tmp_recognition_judgements(⟨Arg⟩) config file entry, contains "blank" recognition judgements: here, the question marks should be replaced with either 'y' (acceptable recognition), or 'n' (unacceptable recognition). Recognition judgements can be updated using the UPDATE_RECOGNITION_JUDGEMENTS ⟨Arg⟩ command.

See Section 16.13.

18.118 TRANSLATE_SPEECH_CORPUS

[Process speech translation corpus.]

Process speech mode translation corpus, defined by the translation_speech_corpus config file entry. The output file, defined by the translation_speech_corpus_results config file entry, contains question marks for translations that have not yet been judged. If these are replaced by valid judgements, currently 'good', 'ok' or 'bad', the new judgements can be incorporated into the stored translation judgements file using the command UPDATE_TRANSLATION_JUDGEMENTS_SPEECH. A second output file, defined by the translation_corpus_tmp_recognition_judgements config file entry, contains "blank" recognition judgements: here, the question marks should be replaced with either 'y' (acceptable recognition), or 'n' (unacceptable recognition). Recognition judgements can be updated using the UPDATE_RECOGNITION_JUDGEMENTS command.

See Section 16.13.

18.119 TRANSLATE_SPEECH_CORPUS_AGAIN Arg1

[Process speech translation corpus with specified ID, using recognition

results from previous run.]

Parameterised version of TRANSLATE_SPEECH_CORPUS_AGAIN. Process speech mode translation corpus, starting from the results saved from the most recent invocation of the TRANSLATE_SPEECH_CORPUS ⟨Arg⟩ command. This is useful if you are testing speech translation performance, but have only changed the translation or generation files. The output files are the same as for the TRANSLATE_SPEECH_CORPUS ⟨Arg⟩ command.

See Section 16.15.

18.120 TRANSLATE_SPEECH_CORPUS_AGAIN

[Process speech translation corpus, using recognition results from previous run.]

Process speech mode translation corpus, starting from the results saved from the most recent invocation of the TRANSLATE_SPEECH_CORPUS command. This is useful if you are testing speech translation performance, but have only changed the translation or generation files. The output files are the same as for the TRANSLATE_SPEECH_CORPUS command.

See Section 16.15.

18.121 TRANSLATE_TRACE_OFF

[Switch off translation tracing (default).]

See Section 16.6.

18.122 TRANSLATE_TRACE_ON

[Switch on translation tracing.]

See Section 16.6.

18.123 UNSET_NOTIONAL_SPEAKER

[Remove setting of notional name for dialogue processing.]

See Section 15.6.

18.124 UNSET_NOTIONAL_TIME

[Use real as opposed to notional time for dialogue processing.]

See Section 15.6.

18.125 UPDATE_DIALOGUE_JUDGEMENTS Arg1

[Update dialogue judgements file with specified ID from annotated dialogue corpus output.]

Parameterised version of UPDATE_DIALOGUE_JUDGEMENTS. Update the dialogue judgements file, defined by the dialogue_corpus_judgements config file entry, from the output of the dialogue corpus output file with ID ⟨Arg⟩, defined by the parameterised config file entry dialogue_corpus_results(⟨Arg⟩). This command should be used after editing the output file produced by the parameterised command BATCH_DIALOGUE ⟨Arg⟩. Editing should replace question marks by valid judgements, currently 'good' or 'bad'.

See Section 15.10.

18.126 UPDATE_DIALOGUE_JUDGEMENTS

[Update dialogue judgements file from annotated dialogue corpus output.]

Update the dialogue judgements file, defined by the dialogue_corpus_judgements config file entry, from the output of the default text dialogue corpus output file, defined by the dialogue_corpus_results config file entry. This command should be used after editing the output file produced by the BATCH_DIALOGUE command. Editing should replace question marks by valid judgements, currently 'good', or 'bad'.

See Section 15.10.

18.127 UPDATE_DIALOGUE_JUDGEMENTS_SPEECH Arg1

[Update dialogue judgements file with specified ID from annotated speech dialogue corpus output.]

Parameterised version of UPDATE_DIALOGUE_JUDGEMENTS_SPEECH. Update the dialogue judgements file, defined by the dialogue_corpus_judgements config file entry, from the output of the dialogue corpus output file with ID ⟨Arg⟩, defined by the parameterised config file entry dialogue_corpus_results(⟨Arg⟩). This command should be used after editing the output file produced by the parameterised command BATCH_DIALOGUES_SPEECH ⟨Arg⟩. Editing should replace question marks by valid judgements, currently 'good' or 'bad'.

See Section 15.10.

18.128 UPDATE_DIALOGUE_JUDGEMENTS_SPEECH

[Update dialogue judgements file from annotated speech dialogue corpus output.]

Update the dialogue judgements file, defined by the dialogue_corpus_judgements config file entry, from the output of the default speech dialogue corpus output file, defined by the dialogue_speech_corpus_results config file entry. This command should be used after editing the output file produced by the BATCH_DIALOGUES_SPEECH command. Editing should replace question marks by valid judgements, currently 'good', or 'bad'.

See Section 15.10.

18.129 UPDATE_RECOGNITION_JUDGEMENTS Arg1

[Update recognition judgements file from temporary translation corpus recognition judgements with specified ID.]

Parameterised version of UPDATE_RECOGNITION_JUDGEMENTS. Update recognition judgements file, defined by the translation_corpus_recognition_judgements config file entry, from the temporary translation corpus recognition judgements file, defined by the translation_corpus_tmp_recognition_judgements(⟨Arg⟩) config file entry and produced by the TRANSLATE_SPEECH_CORPUS ⟨Arg⟩ or TRANSLATE_SPEECH_CORPUS_AGAIN ⟨Arg⟩ commands. This command should be used after editing the temporary translation corpus recognition judgements file. Editing should replace question marks by valid judgements, currently 'y' or 'n'.

See Section 16.17.

18.130 UPDATE_RECOGNITION_JUDGEMENTS

[Update recognition judgements file from temporary translation corpus recognition judgements.]

Update recognition judgements file, defined by the translation_corpus_recognition_judgements config file entry, from the temporary translation corpus recognition judgements file, defined by the translation_corpus_tmp_recognition_judgements config file entry and produced by the TRANSLATE_SPEECH_CORPUS or TRANSLATE_SPEECH_CORPUS_AGAIN commands. This command should be used after editing the temporary translation corpus recognition judgements file. Editing should replace question marks by valid judgements, currently 'y' or 'n'.

See Section 16.17.

18.131 UPDATE_TRANSLATION_JUDGEMENTS Arg1

[Update translation judgements file from annotated translation corpus output with specified ID.]

Parameterised version of UPDATE_TRANSLATION_JUDGEMENTS. Update the translation judgements file, defined by the translation_corpus_judgements config file entry, from the output of the text translation corpus output file with ID ⟨Arg⟩, defined by the parameterised config file entry translation_corpus_results(⟨Arg⟩). This command should be used after editing the output file produced by the parameterised command TRANSLATE_CORPUS ⟨Arg⟩. Editing should replace question marks by valid judgements, currently 'good', 'ok' or 'bad'.

See Section 16.17.

18.132 `UPDATE_TRANSLATION_JUDGEMENTS`

[Update translation judgements file from annotated translation corpus output.]

Update the translation judgements file, defined by the translation_corpus_judgements config file entry, from the output of the default text translation corpus output file, defined by the translation_corpus_results config file entry. This command should be used after editing the output file produced by the TRANSLATE_CORPUS command. Editing should replace question marks by valid judgements, currently 'good', 'ok' or 'bad'.

See Section 16.17.

18.133 `UPDATE_TRANSLATION_JUDGEMENTS_CSV Arg1`

[Update translation judgements file from CSV version of annotated translation corpus output with specified ID.]

Parameterised version of UPDATE_TRANSLATION_JUDGEMENTS_CSV. Update the translation judgements file, defined by the translation_corpus_judgements config file entry, from the output of the text translation corpus output file with ID ⟨Arg⟩, defined by the parameterised config file entry translation_corpus_results(⟨Arg⟩). This command should be used after editing the CSV version of the output file produced by the parameterised command TRANSLATE_CORPUS ⟨Arg⟩. Editing should replace question marks in the first column by valid judgements, currently 'good', 'ok' or 'bad'.

See Section 16.18.

18.134 `UPDATE_TRANSLATION_JUDGEMENTS_CSV`

[Update translation judgements file from CSV version of annotated translation corpus output.]

Update the translation judgements file, defined by the translation_corpus_judgements config file entry, from the output of the default text translation corpus output file, defined by the translation_corpus_results config file entry. This command should be used after editing the CSV version of the output file produced by the TRANSLATE_CORPUS command. Editing should replace question marks in the first column by valid judgements, currently 'good', 'ok' or 'bad'.

See Section 16.18.

18.135 UPDATE_TRANSLATION_JUDGEMENTS_SPEECH Arg1

[Update translation judgements file from annotated speech translation corpus output with specified ID.]

Parameterised version of UPDATE_TRANSLATION_JUDGEMENTS_SPEECH. Update the translation judgements file, defined by the translation_corpus_judgements config file entry, from the output of the speech translation corpus output file, defined by the translation_speech_corpus_results(⟨Arg⟩) config file entry. This command should be used after editing the output file produced by the TRANSLATE_SPEECH_CORPUS ⟨Arg⟩ or TRANSLATE_SPEECH_CORPUS_AGAIN ⟨Arg⟩ command. Editing should replace question marks by valid judgements, currently 'good', 'ok' or 'bad'.

See Section 16.17.

18.136 UPDATE_TRANSLATION_JUDGEMENTS_SPEECH

[Update translation judgements file from annotated speech translation corpus output.]

Update the translation judgements file, defined by the translation_corpus_judgements config file entry, from the output of the speech translation corpus output file, defined by the translation_speech_corpus_results config file entry. This command should be used after editing the output file produced by the TRANSLATE_SPEECH_CORPUS or TRANSLATE_SPEECH_CORPUS_AGAIN command. Editing should replace question marks by valid judgements, currently 'good', 'ok' or 'bad'.

See Section 16.17.

18.137 UPDATE_TRANSLATION_JUDGEMENTS_SPEECH_CSV Arg1

[Update translation judgements file from CSV version of annotated speech translation corpus output with specified ID.]

Parameterised version of UPDATE_TRANSLATION_JUDGEMENTS_SPEECH_CSV. Update the translation judgements file, defined by the translation_corpus_judgements config file entry, from the output of the speech translation corpus output file, defined by the translation_speech_corpus_results(⟨Arg⟩) config file entry. This command should be used after editing the CSV version of the output file produced by the TRANSLATE_SPEECH_CORPUS ⟨Arg⟩ or TRANSLATE_SPEECH_CORPUS_AGAIN ⟨Arg⟩ command. Editing should replace question marks in the first column by valid judgements, currently 'good', 'ok' or 'bad'.

See Section 16.18.

18.138 UPDATE_TRANSLATION_JUDGEMENTS_SPEECH_CSV

[Update translation judgements file from CSV version of annotated speech translation corpus output.]

Update the translation judgements file, defined by the translation_corpus_judgements config file entry, from the output of the speech translation corpus output file, defined by the translation_speech_corpus_results config file entry. This command should be used after editing the CSV version of the output file produced by the TRANSLATE_SPEECH_CORPUS or TRANSLATE_SPEECH_CORPUS_AGAIN command. Editing should replace question marks in the first column by valid judgements, currently 'good', 'ok' or 'bad'.

See Section 16.18.

18.139 WAVFILES Arg1

[Show most recent N wavfiles recorded from speech input at top-level.]

Show the last ⟨Arg⟩ wavfiles recorded using recognition from the top-level (the RECOGNISE command). Each file is displayed together with a timestamp and associated text. The associated text is a transcription if one is available, or the recognition result otherwise.

The files shown by this command can be used as top-level recorded speech input by typing

```
WAVFILE <Wavfile>
```

e.g.

```
WAVFILE c:/Regulus/recorded_wavfiles/2008-04-24_22-36-03/utt05.wav
```

See Section 11.6.

18.140 WAVFILES

[Show wavfiles recorded from speech input at top-level.]

Show wavfiles recorded using recognition from the top-level (the RECOGNISE command). Each file is displayed together with a timestamp and associated text. The associated text is a transcription if one is available, or the recognition result otherwise.

The files shown by this command can be used as top-level recorded speech input by typing

```
WAVFILE <Wavfile>
```

e.g.

```
WAVFILE c:/Regulus/recorded_wavfiles/2008-04-24_22-36-03/utt05.wav
```

See Section 11.6.

19

Config files

Manny Rayner

19.1 Structure of config files

19.2 alterf_patterns_file

Relevant if you are doing Alterf processing with LF patterns. Points to a file containing Alterf LF patterns, that can be tested using the CHECK_ALTERF_PATTERNS command.

19.3 alterf_sents_file

[No documentation yet.]

19.4 alterf_treebank_file

[No documentation yet.]

19.5 analysis_time_limit

[No documentation yet.]

19.6 answer_config_file

[No documentation yet.]

19.7 batchrec_trace

[No documentation yet.]

19.8 batchrec_trace_prolog

[No documentation yet.]

19.9 batchrec_trace_prolog_with_transcriptions

[No documentation yet.]

19.10 batchrec_trace_prolog_with_transcriptions(Arg1)

[No documentation yet.]

19.11 collocation_rules

Relevant to translation applications. Points to a file containing rules for post-transfer collocation processing.

19.12 compiled_collocation_rules

[No documentation yet.]

19.13 compiled_ellipsis_classes

Relevant to translation applications. Points to a file containing the compiled form of the ellipsis processing rules.

19.14 compiled_from_interlingua_rules

[No documentation yet.]

19.15 compiled_graphical_orthography_rules

[No documentation yet.]

19.16 compiled_lf_patterns

[No documentation yet.]

19.17 compiled_lf_rewrite_rules

[No documentation yet.]

19.18 compiled_original_script_collocation_rules

[No documentation yet.]

19.19 compiled_original_script_orthography_rules

[No documentation yet.]

19.20 compiled_orthography_rules

[No documentation yet.]

19.21 compiled_recognition_orthography_rules

[No documentation yet.]

19.22 compiled_surface_constituent_rules

[No documentation yet.]

19.23 compiled_to_interlingua_rules

[No documentation yet.]

19.24 compiled_to_source_discourse_rules

[No documentation yet.]

19.25 compiled_transfer_rules

[No documentation yet.]

19.26 dcg_grammar

[No documentation yet.]

19.27 default_compiled_ellipsis_classes

[No documentation yet.]

19.28 dialogue_batchrec_trace_prolog_with_transcriptions

[No documentation yet.]

19.29 dialogue_batchrec_trace_prolog_with_transcriptions(Arg1)

[No documentation yet.]

19.30 dialogue_corpus

[No documentation yet.]

19.31 dialogue_corpus(Arg1)

[No documentation yet.]

19.32 dialogue_corpus_judgements

[No documentation yet.]

19.33 dialogue_corpus_results

[No documentation yet.]

19.34 dialogue_corpus_results(Arg1)

[No documentation yet.]

19.35 dialogue_files

Relevant to dialogue applications. Points to a list of files defining dialogue processing behaviour.

19.36 dialogue_processing_time_limit

[No documentation yet.]

19.37 dialogue_rec_params

[No documentation yet.]

19.38 dialogue_speech_corpus

[No documentation yet.]

19.39 dialogue_speech_corpus(Arg1)

[No documentation yet.]

19.40 dialogue_speech_corpus_results

[No documentation yet.]

19.41 dialogue_speech_corpus_results(Arg1)

[No documentation yet.]

19.42 discard_lexical_info_in_ebl_training

[No documentation yet.]

19.43 discriminants

Relevant to applications using surface processing. Points to a file of Alterf discriminants.

19.44 ebl_context_use_threshold

Relevant to applications using grammar specialisation. Defines the minimum number of examples of a rule that must be present if the system is to use rule context anti-unification to further constrain that rule.

19.45 ebl_corpus

Points to the file of training examples used as input to the EBL_TREEBANK operation. Intended originally for use for grammar specialisation, but can also be used simply to parse a set of examples to get information about coverage. The format is sent(Atom), so for example a typical line would be

```
sent('switch off the light').
```

(note the closing period).

If the application compiles multiple top-level specialised grammars, the grammars relevant to each example are defined in an optional second argument. For example, if a home control domain had separate grammars for each room, a typical line in the training file might be

```
sent('switch off the light', [bedroom, kitchen, living_room]).
```

19.46 ebl_filter_pred

[No documentation yet.]

19.47 ebl_gemini_grammar

Relevant to applications using grammar specialisation. Specifies the base name of the Gemini files generated by the EBL_GEMINI command.

19.48 ebl_grammar_probs

Convert the current EBL training set, defined by the `ebl_corpus` config file entry, into a form that can be used as training data by the Nuance `compute-grammar-probs utility`. The output training data is placed in the file defined by the `ebl_grammar_probs` config file entry.

19.49 ebl_ignore_feats

Relevant to applications using grammar specialisation. The value should be a list of unification grammar features: these features will be ignored in the specialised grammar. A suitable choice of value can greatly speed up Regulus to Nuance compilation for the specialised grammar.

19.50 ebl_ignore_feats_file

[No documentation yet.]

19.51 ebl_include_lex

Relevant to applications using grammar specialisation. Specifies a file or list of files containing EBL include lex declarations.

19.52 ebl_multiple_grammar_decls

[No documentation yet.]

19.53 ebl_nuance_grammar

Relevant to applications using grammar specialisation. Points to the specialised Nuance GSL grammar file produced by the EBL_NUANCE operation.

19.54 ebl_operationality

Relevant to applications using grammar specialisation. Specifies the operationality criteria.

19.55 ebl_rationalised_corpus

[No documentation yet.]

19.56 ebl_raw_regulus_grammar

[No documentation yet.]

19.57 ebl_regulus_component_grammar

Relevant to applications using grammar specialisation that define multiple top-level specialised grammars. Identifies which specialised Regulus grammar will be loaded by the EBL_LOAD command.

19.58 ebl_regulus_grammar

[No documentation yet.]

19.59 ebl_regulus_no_binarise_grammar

[No documentation yet.]

19.60 ebl_treebank

Parse all sentences in current EBL training set, defined by the `ebl_corpus` config file entry, to create a treebank file. Sentences that fail to parse are printed out with warning messages, and a summary statistic is produced at the end of the run. This is very useful for checking where you are with coverage.

19.61 ellipsis_classes

Relevant to translation applications. Points to a file defining classes of intersubstitutable phrases that can be used in ellipsis processing.

19.62 ellipsis_classes_sents_file

[No documentation yet.]

19.63 ellipsis_classes_treebank_file

[No documentation yet.]

19.64 filtered_interlingua_declarations_file

[No documentation yet.]

19.65 from_interlingua_rule_learning_config_file

[No documentation yet.]

19.66 from_interlingua_rules

Relevant to translation applications. Points to a file, or list of files, containing rules that transfer source language representations into interlingual representations

19.67 `from_interlingua_translation_corpus_judgements`

[No documentation yet.]

19.68 `gemini_grammar`

Specifies the base name of the Gemini files generated by the GEMINI command.

19.69 `generation_dcg_grammar`

[No documentation yet.]

19.70 `generation_grammar`

Relevant to applications that use generation (typically translation applications). Points to the file containing the compiled generation grammar.

19.71 `generation_grammar(Arg1)`

Relevant to applications that use generation (typically translation applications) and also grammar specialisation. Points to the file containing the compiled specialised generation grammar for the subdomain tag ⟨Arg⟩.

19.72 `generation_incremental_deepening_parameters`

Relevant to applications that use generation (typically translation applications). Value should be a list of three positive numbers [⟨Start⟩, ⟨Increment⟩, ⟨Max⟩], such that both ⟨Start⟩ and ⟨Increment⟩ are less than or equal to ⟨Max⟩. Generation uses an iterative deepening algorithm, which initially sets a maximum derivation length of ⟨Start⟩, and increases it in increments of ⟨Increment⟩ until it exceeds ⟨Max⟩.

Default value is [5, 5, 50].

19.73 `generation_module_name`

Relevant to applications that use generation (typically translation applications). Specifies the module name in the compiled generation grammar file. Default is generator.

19.74 `generation_preferences`

Relevant to applications that use generation (typically translation applications). Points to the file containing the generation preference declarations.

19.75 generation_regulus_grammar

Relevant to applications that use generation (typically translation applications). If there is no regulus_grammar entry, points to the Regulus file, or list of Regulus files, that are to be compiled into the generation file.

19.76 generation_rules

Relevant to translation applications. Points to the file containing the generation grammar. Normally this will be a Regulus grammar compiled for generation. The translation code currently assumes that this file will define the module generator, and that the top-level predicate will be of the form

```
generator:generate(Representation, Tree, Words)
```

19.77 generation_rules(Arg1)

[No documentation yet.]

19.78 generation_time_limit

[No documentation yet.]

19.79 global_context

Relevant to translation applications. Defines a value that can be accessed by conditional-dependent transfer rules, if transfer rules are to be shared across several applications defined by multiple config files.

19.80 gloss_generation_rules

[No documentation yet.]

19.81 grammar_probs_data

[No documentation yet.]

19.82 ignore_subdomain

Relevant to applications using grammar specialisation. Sometimes, you will have defined multiple subdomains, but you will only be carrying out development in one of them. In this case, you can speed up EBL training by temporarily adding ignore_subdomain declarations in the config file. An ignore_subdomain declaration has the form

```
regulus_config(ignore_subdomain, <Tag>).
```

The effect is to remove all references to ⟨Tag⟩ when performing training, and not build any specialised grammar for ⟨Tag⟩. You may include any number of ignore_subdomain declarations.

19.83 `interlingua_declarations`

Relevant to translation applications. Points to the file containing the interlingua declarations, which define the constants that may be used at the interlingual representation level.

19.84 `interlingua_structure`

[No documentation yet.]

19.85 `lc_tables_file`

[No documentation yet.]

19.86 `lf_patterns`

Relevant to dialogue applications. Points to a files of LF patterns.

19.87 `lf_patterns_modules`

Relevant to dialogue applications. Value should be a list of modules referenced by the compiled LF patterns.

19.88 `lf_postproc_pred`

Defines a post-processing predicate that is applied after Regulus analysis. If you are using the riacs_sem semantic macros, you must set this parameter to the value riacs_postproc_lf.

19.89 `macro_expanded_grammar`

[No documentation yet.]

19.90 `missing_help_class_decls`

[No documentation yet.]

19.91 `nbest_preferences`

[No documentation yet.]

19.92 `nbest_training_data_file`

[No documentation yet.]

19.93 `no_spaces_in_original_script`

[No documentation yet.]

19.94 `nuance_compile_params`

Specifies a list of extra compilation parameters to be passed to Nuance compilation by the NUANCE_COMPILE command. A typical value is

```
['-auto_pron', '-dont_flatten']
```

19.95 `nuance_grammar`

Points to the Nuance GSL grammar produced by the NUANCE command.

19.96 `nuance_grammar_for_compilation`

[No documentation yet.]

19.97 `nuance_grammar_for_pcfg_training`

[No documentation yet.]

19.98 `nuance_language_pack`

Specifies the Nuance language pack to be used by Nuance compilation in the NUANCE_COMPILE command.

19.99 `nuance_recognition_package`

[No documentation yet.]

19.100 `only_translate_up_to_interlingua`

[No documentation yet.]

19.101 `original_script_collocation_rules`

[No documentation yet.]

19.102 `original_script_encoding`

[No documentation yet.]

19.103 `original_script_generation_rules`

[No documentation yet.]

19.104 `original_script_orthography_rules`

[No documentation yet.]

19.105 `orthography_rules`

Relevant to translation applications. Points to a file containing rules for post-transfer orthography processing.

19.106 paraphrase_corpus

[No documentation yet.]

19.107 paraphrase_generation_grammar

[No documentation yet.]

19.108 parse_preferences

Can be used to define default analysis preferences.

19.109 parsing_history_file

[No documentation yet.]

19.110 pcfg_training_output_directory

[No documentation yet.]

19.111 prolog_semantics

Relevant to generation grammars, in particular paraphrase grammars. If value is 'yes', allow semantic values to be arbitrary Prolog expressions (by default, only Regulus-formatted GSL expressions are allowed).

19.112 reflective_dcg_grammar

[No documentation yet.]

19.113 reflective_dcg_grammar_for_generation

[No documentation yet.]

19.114 regulus_grammar

Points to the Regulus file, or list of Regulus files, that constitute the main grammar.

19.115 regulus_no_sem_decls

Points to a file which removes the sem feature from the main grammar.

19.116 resolution_preferences

[No documentation yet.]

19.117 role_marked_semantics

[No documentation yet.]

19.118 stanford_dcg_debug_grammar

[No documentation yet.]

19.119 stanford_dcg_grammar

[No documentation yet.]

19.120 strcat_semantics

[No documentation yet.]

19.121 surface_constituent_rules

Relevant to applications using surface processing. Points to the surface constituent rules file.

19.122 surface_patterns

Relevant to applications using surface processing. Points to the surface patterns file.

19.123 surface_postprocessing

Relevant to applications using surface processing. Points to a file that defines a post-processing predicate that can be applied to the results of surface processing. The file should define a predicate

```
surface_postprocess(Representation,
                    PostProcessedRepresentation).
```

19.124 tagging_grammar

Relevant to applications using surface processing. Points to a file that defines a tagging grammar, in DCG form. The top-level rule should be of the form

```
tagging_grammar(Item) --> <Body>.
```

19.125 target_model

Relevant to applications using surface processing. Points to a file defining a target model. The file should define the predicates target_atom/1 and target_atom_excludes/2.

19.126 targeted_help_backed_off_corpus_file

[No documentation yet.]

19.127 targeted_help_classes_file

[No documentation yet.]

19.128 `targeted_help_corpus_file`

[No documentation yet.]

19.129 `targeted_help_source_files`

[No documentation yet.]

19.130 `test_corpus`

[No documentation yet.]

19.131 `tmp_ebl_operational_file`

[No documentation yet.]

19.132 `tmp_preds`

[No documentation yet.]

19.133 `to_interlingua_rule_learning_config_file`

[No documentation yet.]

19.134 `to_interlingua_rules`

Relevant to translation applications. Points to a file, or list of files, containing rules that transfer interlingual representations into target language representations.

19.135 `to_interlingua_translation_corpus_judgements`

[No documentation yet.]

19.136 `to_source_discourse_rules`

Relevant to translation applications. Points to a file, or list of files, containing rules that transfer source representations into source discourse representations.

19.137 `top_level_cat`

Defines the top-level category of the grammar.

19.138 `top_level_generation_cat`

Relevant to applications that use generation (typically translation applications). Defines the top-level category of the generation grammar. Default is .MAIN.

19.139 top_level_generation_feat

Relevant to applications that use generation (typically translation applications). Defines the semantic feature in the top-level rule which holds the semantic value. Normally, the rule will be of the form

```
'.MAIN':[gsem=[value=Sem]] --> Body
```

and the value of this parameter will be value (default if not specified).

19.140 top_level_generation_pred

Relevant to applications that use generation (typically translation applications). Defines the top-level category of the generation grammar. For translation applications, the value should be generate (default if not specified).

19.141 transfer_rules

Relevant to translation applications. Points to a file, or list of files, containing rules that transfer source language representations into target language representations.

19.142 translate_from_interlingua

[No documentation yet.]

19.143 translation_corpus

Relevant to translation applications. Points to a file of examples used as input to the TRANSLATE_CORPUS command. The format is sent(Atom), so for example a typical line would be

```
sent('switch off the light').
```

(note the closing period).

19.144 translation_corpus(Arg1)

Relevant to translation applications. Points to a file of examples used as input to the parameterised command TRANSLATE_CORPUS ⟨Arg⟩. The format is sent(Atom), so for example a typical line would be

```
sent('switch off the light').
```

(note the closing period).

19.145 translation_corpus_judgements

Relevant to translation applications. Points to a file of recognition judgements. You should not normally edit this file directly, but update it using the command UPDATE_RECOGNITION_JUDGEMENTS.

19.146 translation_corpus_recognition_judgements

[No documentation yet.]

19.147 translation_corpus_results

Relevant to translation applications. Points to the file containing the result of running the TRANSLATE_CORPUS command. You can then edit this file to update judgements, and incorporate them into the translation_corpus_judgements file by using the command UPDATE_TRANSLATION_JUDGEMENTS.

19.148 translation_corpus_results(Arg1)

Relevant to translation applications. Points to the file containing the result of running the parameterised command TRANSLATE_CORPUS ⟨Arg⟩. You can then edit this file to update judgements, and incorporate them into the translation_corpus_judgements file by using the parameterised command UPDATE_TRANSLATION_JUDGEMENTS ⟨Arg⟩.

19.149 translation_corpus_tmp_recognition_judgements

Relevant to translation applications. Points to the file of new recognition results generated by running the TRANSLATE_SPEECH_CORPUS command. You can then edit this file to update the judgements, and incorporate them into the translation_corpus_recognition_judgements file using the command UPDATE_RECOGNITION_JUDGEMENTS.

19.150 translation_corpus_tmp_recognition_judgements(Arg1)

Relevant to translation applications. Points to the file of new recognition results generated by running the TRANSLATE_SPEECH_CORPUS ⟨Arg⟩ command. You can then edit this file to update the judgements, and incorporate them into the translation_corpus_recognition_judgements file using the command UPDATE_RECOGNITION_JUDGEMENTS ⟨Arg⟩.

19.151 translation_rec_params

Relevant to translation applications. Specifies the list of Nuance parameters that will be used when carrying out recognition for the TRANSLATE_SPEECH_CORPUS command. These parameters must at a minimum specify the recognition package and the top-level Nuance grammar, for example

```
[package=med_runtime(recogniser), grammar='.MAIN']
```

19.152 `translation_speech_corpus`

Relevant to translation applications. Points to a file of examples used as input to the TRANSLATE_SPEECH_CORPUS command. The format is ⟨Wavfile⟩ ⟨Words⟩, so for example a typical line would be

```
C:/Regulus/data/utt03.wav switch off the light
```

19.153 `translation_speech_corpus(Arg1)`

Relevant to translation applications. Points to a file of examples used as input to the TRANSLATE_SPEECH_CORPUS(⟨Arg⟩) command. The format is ⟨Wavfile⟩ ⟨Words⟩, so for example a typical line would be

```
C:/Regulus/data/utt03.wav switch off the light
```

19.154 `translation_speech_corpus_results`

Relevant to translation applications. Points to the file containing the result of running the TRANSLATE_SPEECH_CORPUS command. You can then edit this file to update judgements, and incorporate them into the translation_corpus_judgements file by using the command UPDATE_TRANSLATION_JUDGEMENTS_SPEECH.

19.155 `translation_speech_corpus_results(Arg1)`

Relevant to translation applications. Points to the file containing the result of running the TRANSLATE_SPEECH_CORPUS ⟨Arg⟩ command. You can then edit this file to update judgements, and incorporate them into the translation_corpus_judgements file by using the command UPDATE_TRANSLATION_JUDGEMENTS_SPEECH ⟨Arg⟩.

19.156 `tts_command`

[No documentation yet.]

19.157 `wavfile_directory`

Relevant to translation applications. If output speech is to be produced using recorded wavfiles, points to the directory that holds these files.

19.158 `wavfile_preceding_context`

[No documentation yet.]

19.159 `wavfile_preceding_context(Arg1)`

[No documentation yet.]

19.160 wavfile_recording_script

Relevant to translation applications. If output speech is to be produced using recorded wavfiles, points to an automatically created file that holds a script which can be used to create the missing wavfiles. This script is produced by finding all the lexical items in the file referenced by generation_rules, and creating an entry for every item not already in wavfile_directory. The file is created as part of the processing carried out by the LOAD_TRANSLATE command.

Due to limitations of some operating systems the script contains some latin-1 characters translated to character sequences shown in the table below.

Char	Translates to
á	a1
â	a2
à	a3
ä	a4
å	a5
ç	c1
é	e1
ê	e2
è	e3
ë	e4
æ	e6
ñ	n1
ó	o1
ô	o2
ò	o3
ö	o4
ú	u1
û	u2
ù	u3
ü	u4

19.161 wavfiles

Show wavfiles recorded using recognition from the top-level (the RECOGNISE command). Each file is displayed together with a timestamp and associated text. The associated text is a transcription if one is available, or the recognition result otherwise.

The files shown by this command can be used as top-level recorded speech input by typing

```
WAVFILE <Wavfile>
```

e.g.

```
WAVFILE c:/Regulus/recorded_wavfiles/2008-04-24_22-36-03/utt05.wav
```

19.162 working_directory

Working files will have names starting with this prefix.

19.163 working_file_prefix

[No documentation yet.]

References

Bouillon, P., M. Rayner, N. Chatzichrisafis, B.A. Hockey, M. Santaholma, M. Starlander, Y. Nakao, K. Kanzaki, and H. Isahara. 2005. A generic multi-lingual open source platform for limited-domain medical speech translation. In *In Proceedings of the 10th Conference of the European Association for Machine Translation (EAMT)*. Budapest, Hungary.

Rayner, M., B.A. Hockey, J.M. Renders, N. Chatzichrisafis, and K. Farrell. 2005. A voice enabled procedure browser for the International Space Station. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (interactive poster and demo track)*. Ann Arbor, MI.

Index

later, 1

Regulus
  why, 1