NSM-DALIA command line interface

July 17, 2009

1 Introduction

This file contains a short description of NSM-DALIA command-line interface.

You work with NSM-DALIA by writing commands at the prompt.

Commands are full-stop terminated, and can have arguments between parentheses.

There must be no space between the command name and the open parenthesis.

2 Commands

2.1 cline_interface.pl – NSM-DALIA, command line version

To be done
- `wl/0` implementation
- re-implementation of pw

This module provides aliases to call the main NSM-DALIA predicates, (contained in modules `nsmdalia.pl` and `file_reader.pl`), and the running interface.

l(+*Lang***)** *[det]*

Loads a language module into the program, and sets the current_language flag to the loaded language. From now on, parse commands will be taken to refer to that language as the parsing-from language.

See also nsmdalia:load(*Lang*)

The ”*Lang*” argument is either a language code (without quotation marks), or a full language name (a PROLOG string, between double quotation marks).

The ”load” command works like this: if a compiled module is found in the lang_bin directory, it will be loaded. If not, the directory lang_src will be searched for the source files, compiled and loaded. (The latter procedure is slower).

The user who is not also writing a grammar need only know which languages are available, and what their code is (the ”list” command displays all the installed languages with their code). As the only language available in NSM-DALIA v. 0.8 is English (code: eng), you can load the English grammar with:

```
DALIA> l(eng).
```

or:

```
DALIA> l("English").
```

A note for gramamr writers: please remember that, once a grammar is saved in the lang_bin directory (”s” command), l(*Lang*) will read from that directory, NOT from the source file. If, after you modified your source file, and you are wondering why on earth that sentence still do not parse, though you have fixed the bug in your grammar – that’s because you have loaded the file with ”l”, not with ”b”, so the program does not read the new grammar, but the old compiled one! When you are developing a grammar, better not to compile it (”c” command”) until you are finished. (However, see ”b” command below).

l2(+*Lang***)** *[det]*

This command loads a grammar module, just like l(*Lang*), with the difference that the language loaded is set as **l2** (*second language*). Translation commands like t(Sentence) will translate into l2.

ldb(+*FileName:string***)** *[det]*

Loads a text database. Text is stored in a series of `mtext/3` facts (see `nsmfiles.txt` for details).

b(+*Lang***)** *[det]*

Full form: build(+*Lang*). Loads a language module into the program, and sets the current_language flag to the loaded language. Files are searched for exclusively in the lang_src directory. Useful for grammar writers. The ”*Lang*” argument is a language code or a string (full language name).

sl1(+*Lang***)** *[det]*

Sets *Lang* as the current language.

sl2(+*Lang***)** *[det]*

Full form: set_current_l2(*Lang*).

Used in automatic translation. Once current_lang and current_l2 are set, the ”translation” commands translate NSM sentences from current_lang into current_l2. *Lang* = language code.

sl

Full form: switch_languages.

Useful to switch between various languages when you have loaded more than one grammar. The ”Lang” argument is a language code.

sm(+*MarkUp***)** *[det]*

Full form: set_markup(+*MarkUp*).

MarkUp is a term referring to one of the supported markup schemes (see NSM-`files.txt` and `grammars.txt`).

smf(+*Format***)** *[det]*

Format is one of the terms `line_by_line` or `whole_text`. The NSM-file parser has an option for translating an NSM text and displaying the translation together with the original.

Note that, for example, the rtf markup scheme requires a `line_by_line` option which, however, result in the texts being displayed in a two_column table, one column for the original, the other for the translation, and with corresponding lines aligned.

stt(+*TableId:integer***)** *[det]*

For languages having more than one transcription possibility, sets the transcription table number *TableId* as active.

pw(+*W***)** *[det]*

Full form: parse_and_write_word(*W*).

You won't need this very much, because "ps" can parse single words, as well as whole sentences. When you are developing a grammar, however, this command can be useful, because it also outputs a morpheme split-up of the word. The argument *W* is a double-quoted string, which is analyzed as a word of the last current language set. Ex.

```
DALIA> pw("things").
```

NSM-DALIA answers:

```
Morphemic String: thing-s
ct(n(n), sp(e, e, plur(e), [], something(thing))).
```

ps(+*S***)** *[det]*

Full form: parse_and_write_sentence(*S*). Like "pw", but it parses a whole sentence. Ex.

```
DALIA> ps("something good is happening now").
```

NSM-DALIA answers with the corresponding NSM-PROLOG formula:

```
ct(s, s(e, time(e, now, e), e, e, e,
        p(i(happen),
          [o:sp(e, e, sing(e),[good], something(thing)),
           d:e]),
        e, e)).
```

pst(+*S***)** *[det]*

Full form: parse_and_write_tabular_sentence(*S*,min). Like "ps", but delivers the analysis in tabular fashion.

Prints only the "minimal" table.

```
DALIA> ps("I do something good")
```

NSM-DALIA answers with the corresponding NSM-PROLOG formula:

```
s: _____
. pred: _____
. . v:do
. . a: _____
. . . pers:loc(me)
. . . n:me
. . o: _____
. . . a: _____
. . . . eval: _____
. . . . . a:good
. . . n:something(thing)
```

pstf(+*S***)** *[det]*

Full form: parse_and_write_tabular_sentence(*S*,full). Like ”ps”, but delivers the analysis in tabular fashion.

Prints only the full table.

```
DALIA> ps("I do something good")
```

NSM-DALIA answers with the corresponding NSM-PROLOG formula:

```
s: _____
. c: _____
. . compl:e
. . top:e
. . int:e
. . top2:e
. . foc:e
. . pol:e
. mod: _____
. . speech_act:e
. . eval:e
. . evid:e
. . epist:e
. f: _____
. . top3:e
. . finite:e
. t: _____
. . past:e
. . fut:e
. m: _____
. . irrealis:e
. . necess:e
. . possib:e
. . vol:e
. . oblig:e
. . allow:e
. asp: _____
. . hab:e
. . rep:e
. . freq:e
. . celer:e
. . ant:e
. . term:e
. . cont:e
. . perf:e
. . retro:e
. . pross:e
. . dur:e
. . prog:e
. . prosp:e
. . compl_sg:e
. . compl_pl:e
. vo: _____
. . v_1:e
. . v_2:e
. . v_3:e
. ak: _____
. . celer:e
. . comp:e
. . rep:e
. . freq:e
. pred: _____
. . v:do
. . a: _____
. . . det:e
. . . alt:e
. . . q:e
. . . pers:loc(me)
. . . a: _____
. . . . eval:e
. . . . size:e
. . . . length:e
. . . . height:e
. . . . speed:e
. . . . width:e
. . . . weight:e
. . . . temp:
. . . . age:e
. . . . shape:e
. . . . colour:e
. . . . origin:e
. . . . material:e
. . . dem:e
. . . poss:e
. . . class:e
. . . n:me
. . o: _____
. . . det:e
. . . alt:e
. . . q:e
. . . pers:e
. . . a: _____
. . . . eval: _____
. . . . . int:e
. . . . . a:good
. . . . size:e
. . . . length:e
. . . . height:e
. . . . speed:e
. . . . width:e
. . . . weight:e
. . . . temp:e
. . . . age:e
. . . . shape:e
. . . . colour:e
. . . . origin:e
. . . . material:e
. . . dem:e
. . . poss:e
. . . class:e
. . . n:something(thing)
. . d:e
. . e:e
. . c:e
. . i:e
. . b:e
. . l:e
. . m:e
```

gs(*+LF*) *[det]*

Full form: gen_and_write_sentence(*LF*). The ”*LF*” argument is an NSM-PROLOG formula; the output is the corresponding sentence in the current language (or the formula itself, if NSM-DALIA fails generation). Ex.

```
DALIA> gs(ct(s, s(e, before(now), e, e, e,
                  p(do, [a:sp(e, e, sing(e), [], me),
                         o:sp(e, e, sing(e), [good],
                              something(thing)),
                         d:e, c:e, i:e]),
                  e, e))).
```

Answer:

```
I did something good
```

t(+*Sentence***)** *[det]*

Full form: translate_and_write_sentence(*Sentence*). Translates a sentence from current_language to current_l2. *Sentence* argument is a string, between double quotation marks.

so(+*Filename***)** *[det]*

Full form: set_output_file(*Filename*). Redirects the output to the file named *Filename* (in some operating systems, you will need to give a full name with extension – NSM-DALIA adds none). ”*Filename*” is double-quoted string. Ex.

```
DALIA> so("pippo.txt").
  * Output file set to pippo.txt
```

You will perhaps use this mostly before a ”pf” command.

ro(+*F***)** *[det]*

Used to select an output file if the user wants it to be overwritten by the new data. `so/1` *appends* the output file if it exists, and does not rewrite it, as `ro/1` does.

co

Full form: close_output_file

Users will rarely use this command, because the output file is automatically closed (and the output redirected again to the console) when the parsing of an NSM-input file is finished.

pf(+*F***)** *[det]*

Full form: parse_file(*F*). Parses and outputs a text file with particular tags (see the documentation file ”NSM-`files.txt`”). You can write a whole file of NSM texts in, say, English NSM, and then have it parsed or translated automatically. This will probably be one of the most used commands, when other language modules are available.

tm(+*SWITCH***)** *[det]*

Full form: trace_morphology(*SWITCH*). To switch on verbose mode for morphology parsing, say:

```
tm(1).
```

To switch it off, say

```
tm(0)
```

Verbose modes are useful in grammar development. Turning tracing morphology on, the ”parse” and ”generate” commands will display information about the grammar rules used during the parsing process, in the morphophonemic component of the grammar. This will help you to find out why the morphophonemic component of your grammar is not doing what you intended it to do.

ts(+*SWITCH***)** *[det]*

Full form: trace_syntax(*SWITCH*). ”*SWITCH*” is either ”1” or ”0”. Turns on or off syntax verbose mode for parsing.

tg(+*SWITCH***)** *[det]*

Full form: trace_generation(*SWITCH*). ”*SWITCH*” is either 0 or 1. Turns on/of verbose mode for generation.

s(+*L*, +*FullName***)** *[det]*

See `s/1`

wl(+*Lang***)** *[det]*

Full form: word_list(+*Lang*).

Saves a formatted version of the lexical database (stored with `m/4` predicates), in the current active markup format.

w *[det]*

GNU-ish command, to display the ”WARRANTY” part of the GNU GPL.

c *[det]*

GNU-ish command, a pointer to the ”CONDITION” part of the GNU GPL.

pg *[det]*

Prints a formatted version of the grammar.

run *[det]*

Procedure run calls `init/0`, then enters the main loop. Fails on bugs

.