Alpage Linguistic Processing Chain for French

Isabelle Cabrera <Isabelle.Cabrera@inria.fr>

May 20, 2008

Contents

1 Introduction

Alpage project team maintains a full linguistic processing chain for French based on:

sxpipe a pre-parsing chain for French including segmentation, spelling corrections, lexicon lookup, named entities detections, ...

DyALog a parser compiler and logic programming enviromnent

frmg a French Meta Grammar

Lefff a French Morphological and Syntactic Lexicon

You can try our online demo, which is based on the latest stable version of the chain: `http://alpage.inria.fr/parserdemo`.

2 Alpage Linguistic Processing Chain

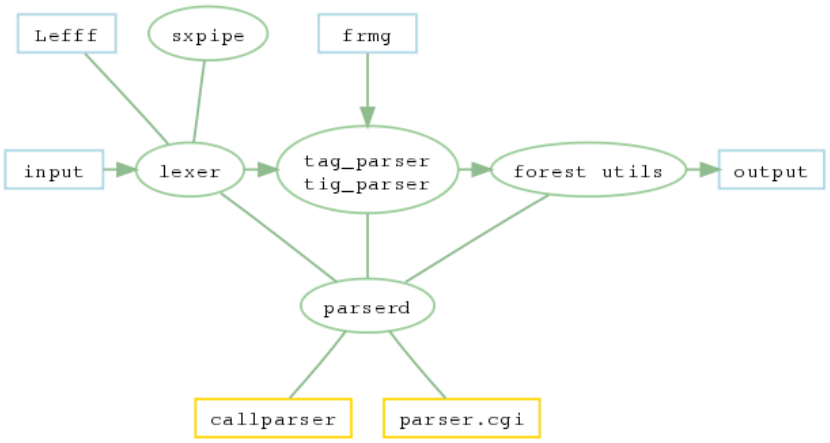

Figure 1: Alpage Linguistic Processing Chain

Figure 1 shows a simplified view of Alpage Linguistic Processing Chain. The **lexer** uses **sxpipe** (a full linguistic preprocessing chain) to preprocess the input (sentence, file of sentences) and combines sxpipe output with lexical information retrieved from **Lefff** (French syntactic lexicon). The lexer output can then be processed by the **TAG parser** (based on metagrammar **frmg**). The forest produced by the parser can be transformed into many formats with **forest_utils**. The user can interact with the server of parsers (parserd), or directly call each component. parserd has two clients:

callparser: a command which can take many options.

parser.pl a web interface.

The easiest way is to interact with parserd with command callparser.

In the next section you will find a tutorial regarding the global use of ALPC. The last section gives more detailed information about the use of each component of the chain.

3 Installation

The easiest way to install Alpage Linguistic Processing Chain is to use **Alpage Installer (alpi)**. **alpi** is a Perl script to help users install locally software developped by Alpage project team. **alpi** works in three steps. This is the process For each package of the chain:

1. download sources
2. detect and install dependencies
3. install package

alpi home page `http://alpage.inria.fr/alpi.en.html`

FAQ `http://alpage.inria.fr/alpifaq.en.html`

alpi mailing list `https://gforge.inria.fr/mail/?group_id=481`

3.1 Download Alpage Installer

Download and unzip alpi: `http://alpage.inria.fr/~cabrera/download/alpi.pl.zip`

3.2 What you need

This is what you need before using alpi.

- connection to Internet :) (because the script retrieves the packages through FTP or Subversion)
- Perl >= 5.8
- AppConfig
- IPC::Run
- pkg-config 0.15.0
- g++
- perlcc: for Lefff (perlcc is part of Mandriva rpm perl-devel)
- recode: for parserd (callparser)...
- xsltproc: for frmg

- bison > 2.3: for DyALog
- flex: for DyALog
- telnet: for parserd

If you retrieve the sources from the Subversion repository:

- Subversion
- autoconf >= 2.60
- automake >= 1.10
- libtool >= 1.5
- makeinfo: for DyALog (makeinfo is part of Mandriva rpm texinfo)

Optional but recommended dependencies:

- ImageMagick: to display a graph of dependencies (command display)
- dillo: to display HTML streams produced by callparser (package parserd)
- uDraw (formerly known as daVinci): view Meta-Grammars hierarchies with mgviewer (package mgtools)

3.3 User installation

```
./alpi.pl
```

This command installs the Linguistic Processing Chain downloading the tar.gz releases through FTP.

Note that you should not be root to launch **alpi** because the chain is installed locally. In future development it will be possible to install as root. By default the chain is installed under `$HOME/exportbuild` (`/home/toto/exportbuild/` if your homedir is `/home/toto`).

3.4 Developper installation

This installation is only recommend for developpers. Installing the chain from the Subversion repository means installing an unstable version!

1. You don't have an INRIA GForge account. You can retrieve the sources with anonymous checkout.

```
./alpi.pl --svn=anonymous
```

Note: when retrieving the sources anonymously, you will not be able to contribute to the code, and the few packages that belong to private projects will be downloaded by FTP.

2. You have an INRIA GForge account and are registered to our GForge projects. (See information about option svn below for more details.)

```
./alpi.pl --svn=mylogin
```

When installation is finished, **alpi** sets up your environment variables so that you can use Alpage software directly. You can set up that environment yourself typing :

```
source ${PREFIX}/sbin/setenv.sh
```

This environment is set for that session only. In order to keep it permanently, you need to add the content of setenv.sh in the file .bashrc (in your homedir).

It is strongly recommended to look at file alpi.log (which is in same directory as **alpi** script) to search for errors, because **alpi** does not explicitly mention them!

3.5 Options

config=***file*** to read a specific config file (default *alpi.conf*)

prefix=***path*** (default */home/toto/exportbuild*)

vardir=***path*** (default */tmp/toto/exportbuild*)

port|p=<**number**> (default 8999)

user|u=<**user**>

group|g=<**group**>

svn=<**gforge_login**> to download sources from the svn repository on Gforge

package|pkg=<**package**> to download a specific package

skippkg=<**package**> to skip installation of specified package

skipdep=<**package**> to skip installation of all dependencies

linux32 to force 32bit emulation mode on 64bit machines

force|f

dependencies|dep to list the dependencies needed to run this script

version|v to print versions of Alpage packages

project to print Gforge project name of each package

info information about packages installed by alpi

log=<**log file**> (default alpi.log)

test run packages tests (default no)

-config

You can write a small configuration script called alpi.conf and placed in the same directory as alpi.pl. This script could look like this:

svn= mylogin prefix= /home/toto/somedir

Then you only need to type ./alpi.pl to run the script. If you want to add an option, place it in this configuration file.

-prefix

Everything is installed under the path specified with option prefix [the default is ${HOME}/exportbuild].

-user and -group

These options are useful if you install parserd which provides a server of parsers. This server will run under user and group. When the values are not provided by the user, the script retrieves user name under which it runs, and the first group the user belongs to.

-svn

alpi can either download the latest stable releases from FTP, or retrieve the latest sources from the Subversion repository (svn) on Inria Gforge. The stable version is retrieved by default.

To retrieve the last sources from the Subversion repository, there are two possibilities.

1. Get the sources anonymously:

```
./alpi.pl --svn=anonymous
```

2. If you are a developper among our projects on INRIA GForge, provide your gforge login:

```
./alpi --svn=mylogin
```

If you do not have a GForge account yet, you can create one filling the following form: `https://gforge.inria.fr/account/register.php`. Then you should contact the author of alpi to get access to the sources (some are public, other private). You will be registered in our GForge projects Alpage Linguistic Workbench, DyALog, Metagrammar Toolkit, Syntax and Alexina.

Then you need to set up ssh (this section comes from the GForge FAQ `http://siteadmin.gforge.inria.fr/FAQ.html#Q6`):

Generate a pair of rsa public/private keys:

```
ssh-keygen -t rsa
```

Paste your public key (/.ssh/id_rsa.pub) in the gforge website. To do this, you need to go to your account and then go to the Account maintainance tab. At the bottom of the Account maintainance tab, you should see a Shell Account

Information section which contains an [Edit keys] link. Paste your public key(s) in the empty field below and click the Update button.

! A common problem is that the Shell Account Information field does not appear in your account page. This usually happens because you do not belong to any gforge project.

! Please, be aware that uploading your ssh public key on the server will not allow you to connect to the server immediately through ssh. To do so, you will need to wait at most 24h. If your connection is impossible 24h later, please, contact the server administrators.

Finally you should set up a ssh agent, so that you will not be prompted to type your passphrase each time **alpi** accesses the Subversion repository:

```
eval ‘ssh-agent‘
cp /etc/ssh/ssh_config .ssh/config # ForwardAgent variable is set to‘‘yes’’
ssh-add
```

The ssh-agent is started in the beginning of an X-session or a login session.

-package|pkg

To install a specific package: package=somepkg (or pkg=somepkg) Examples:

```
./alpi --package=DyALog
./alpi --pkg=DyALog --pkg=frmg
```

To install a special revision of the Subversion repository for a package, type:

```
./alpi --pkg=<package name>-r<revision number>
```

For example, get revision 1144 of package sxpipe:

```
./alpi --pkg=sxpipe-r1144 --svn=mylogin
./alpi --pkg=sxpipe-r1144
```

Note that if you have a GForge account and forget to provide it, the checkout will be anonymous.

-skippkg

You can specify which packages you don’t want to be installed.

```
./alpi --skippkg=biomg
```

-linux32

Package DyALog is not yet compatible with 64b machines. If you have a 64b machine, the solution is to force 32bit emulation mode, activating option -linux32. Command linux32 is mandatory.

Type ’uname -m’ to check if you have a 64b machine.

-force

force causes the script to start from 'autoreconf' instead of 'make' when the sources are retrieved from the Subversion repository. When the sources are retrieved by FTP, the tarball is downloaded and decompressed again.

-test

With this option **alpi** will run tests before installing any package ('make check' for packages using autotools, 'make test' for Perl packages). This is recommended if you are a developper.

3.6 Install the web interface

You can install a web interface such as our online demo: `http://alpage.inria.fr/parserdemo`. alpi does not fully install this interface but provides all the required files.

Note that you need to be root to configure the server. We assume that you know what you are doing.

3.6.1 What you need

You need an apache server and apache-mod_perl, an embedded Perl interpreter for the apache web server.

3.6.2 Configure apache server

Then you need to create a configuration file called, for instance, www-parserd.conf, which should be under /etc/httpd/conf/webapps.d/ and contain something like:

```
<IfModule mod_perl.c>
    Alias /perl/ "${PREFIX}/var/www/perl/"
    PerlModule  ModPerl::Registry
    <Directory "${PREFIX}/var/www/perl/">
        <FilesMatch "\.(pl|cgi)$">
          SetHandler perl-script
          PerlHandler  ModPerl::Registry
          PerlSetEnv PERL5LIB ${PREFIX}/lib/perl5/site_perl:
${PREFIX}/lib/perl5:${PREFIX}/lib/perl5/site_perl/5.8.8:
${PREFIX}/lib/perl/5.8.8/auto:${PREFIX}/share/automake-1.9:
${PREFIX}/share/perl/5.8.8:${PREFIX}/lib/perl/5.8.8:
${PREFIX}/lib/perl:${PREFIX}/lib/perl/5.8:${PREFIX}/share/perl/5.8:
/usr/lib/perl5/vendor_perl/5.8.8
          PerlSetEnv DOTCMD /usr/bin/dot
          PerlSetEnv XLTPROC /usr/bin/xsltproc
          PerlSendHeader On
        </FilesMatch>
        Options ExecCGI -Indexes MultiViews
        Order deny,allow
```

```
        Allow from all
    </Directory>
</IfModule>
```

where ${PREFIX} is the path to your local installation of the linguistic processing chain.

To know the content of the variable PERL5LIB, you should first set the right environment by typing at a command prompt:

```
source /home/toto/exportbuild/sbin/setenv.sh
```

Then type:

```
echo $PERL5LIB
```

The Alias (line 2) is necessary if you have installed parserd with alpi, meaning locally. In such case, parser.pl is typically under ${PREFIX}/var/www/perl/ instead of /var/www/perl/. This Alias should not be overriden by the one present in the file 75_mod_perl.conf, which should be under /etc/httpd/modules.d/. (If you don't have 75_mod_perl.conf, it probably means that apache-mod_perl is not installed on your machine. You must install it.) This file typically contains these two lines:

```
Alias /perl/ /var/www/perl/
Alias /cgi-perl/ /var/www/perl/
```

You should comment these lines.

The last thing you need to do is to restart the apache server, typing:

```
/etc/init.d/httpd reload
```

Finally open a web browser and go to `http://localhost/perl/parser.pl`. It should work :)

4 Tutorial

4.1 Setting up the environment

To use the chain directly, you first need to setup the environment variables typing:

```
source ${PREFIX}/sbin/setenv.sh
```

`${PREFIX}` is where the processing chain is installed (default location is `${HOME}/exportbuild`). This command only works for the current session. In order to keep this environment permanently, you may wish to include the content of `setenv.sh` in your `.bash_profile`.

4.2 Using the parsers through the server of parsers

The server of parserss **parserd** may be started and stopped by the following commands:

```
${PREFIX}/sbin/parserd_setenv start
${PREFIX}/sbin/parserd_setenv stop
```

The server of parsers can be accessed through the command `callparser`. To use directly the command `callparser`, do not forget to set up your environment (see previous section).

To display a graphical view of dependencies (Graphviz and ImageMagick must be installed on your machine):

```
echo "il mange une pomme" | callparser -in - -d dep
```

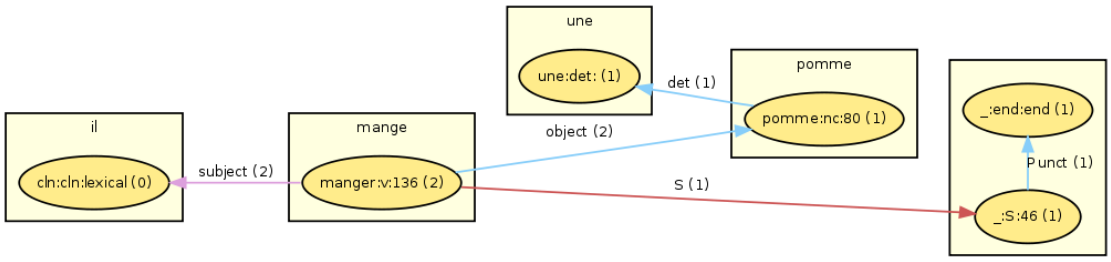

Figure 2: Graphical view of dependencies

To get an XML representation of dependencies:

```
echo "il mange une pomme" | callparser -in - -xmldep
```

To get ambiguity and time statistics:

```
echo "il mange une pomme" | callparser -in - -time -stats
```

To run a sentence file (one sentence per line):

```
cat <file> | callparser -in - -time -stats
```

To save a collection of parsed sentences in directory `mycoll` with basename `sentence`:

```
cat <file> | callparser -in - -collsave -xmldep -collbase sentence -colldir mycoll
```

`callparser` has many options to visualize the parsing output (as HTML, XML, raw, dot, ...). To find more information regarding `callparser` usage, please type:

```
perldoc callparser
```

4.3 Using the parsers directly

If you don't wish to use the server of parsers (parserd), you can type the commands directly. To produce a forest, try the following command line. (Of course, do not type \, this is a single command line.)

```
echo "il mange une pomme" | easy.pl |\
tig_parser ${PREFIX}/share/frmg/small_header.tag - -forest
```

The input sentence is processed by the lexer (easy.pl), which calls the preprocessing chain (sxpipe) and retrieves lemma information from the lexicon (Lefff). The out put is then parsed by the parser (tig_parser).

To get a graphical view of the forest, you can use the forest converter:

```
echo "il mange une pomme" | easy.pl |\
tig_parser ${PREFIX}/share/frmg/small_header.tag - -forest |\
forest_convert.pl -f lp -t dep | dot -Tgif | display
```

-f (from) is the option for input format, here lp (DyALog forest format). -t (to) is the output format, here dep. Option dep produces dot code which can be processed to display an image. You need Graphviz to process the dot output, and ImageMagick to display the image.

You will get something like figure 3, where *manger* is the lemma, *v* the grammatical category (verb), *117* the tree number and *(2)*, the number of derivations from this node.

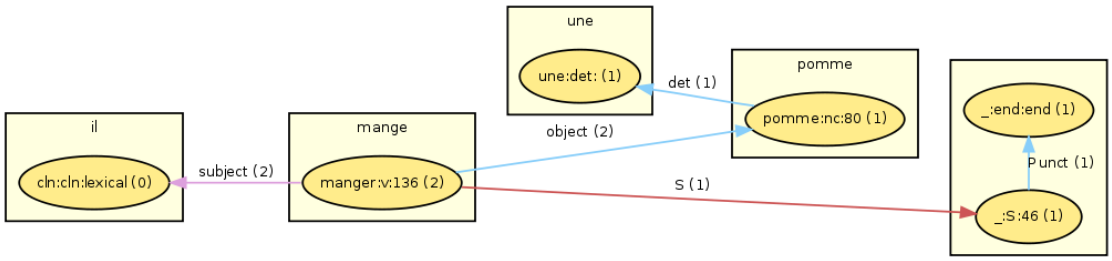

Figure 3: Dependencies

`forest_convert.pl` has many conversion options, most of them similar to the ones used by `callparser` (see previous section). For information regarding `forest_convert.pl`, please type:

```
perldoc forest_convert.pl
```

4.4 Using the parsers through a web interface

You can either use our online demo (`http://alpage.inria.fr/parserdemo`) or you can run this same interface on your machine. This requires the installation of an Apache server. Please follow the instructions provided in section 3.6 (Install the web interface).

4.5 Parsing large corpora

To parse large corpora, the solution is to run the parser on a cluster (group of computers). ALPC can be installed on the front node (master) and sentences will be dispatched to the nodes.

More information soon...

4.6 Input

Callparser accepts plain sentences or unfolded dags (direct acyclic graphs). To create an unfolded dag, first create a dag with sxpipe and pipe the resulting dag to dag2udag.

```
echo "Je teste une phrase" | sxpipe | dag2udag
```

4.7 The output formats

The derivation forest produced by the parsers may be displayed in various formats. These outputs can be divided in two categories: dependency (dependency, EASy format) and derivation (xml, tree, grammar). Some formats can be displayed in xml or in a graphical/html view (dependency graph, easyhtml, grammar, tree). The tagger ouput is an additional option.

-lpdep raw representation of shared forest as produced by the parser

-forest xml representation of forest

-display dep graph representation of dependencies

-xmldep print xml representation of dependencies

-grammar html grammar

-easy easy refers to the xml output used at EASy, a French parsing competition.

-easyhtml html representation of easy format

-tagger print tags

4.7.1 EASy format

The idea behind EASy format is not to apply a syntactic theory but to have a more general parse into syntactic groups, so that the outputs from the various parsers in competition can be easily compared and evaluated. For that purpose, the ouput only shows the (possibly) best analysis. The annotated elements are:

- syntactic groups
- syntactic relations :
  - between words
  - between a syntactic group and a word
  - between syntactic groups

4.7.2 Dependencies

We establish a link from word X to word X2 whenever X2 anchors some tree T2 used on some node of some tree T1 anchored by A1.

4.8 Troubleshooting

Please check the FAQ if you encounter any problem:
`http://alpage.inria.fr/alpchainfaq.en.html`.

Still, here are some tips when the parser fails to parse a sentence. There are many reasons why a sentence cannot be parsed. It can be a problem with the grammar `frmg`, with the lexicon `Lefff`, and so on... If you notice any problem, please notify us. The more feedback we have, the more we can improve the processing chain. Still there are some commands you can try to trace or temporarily solve the problem.

Check the output of `easy.pl`:

```
echo "Je teste une phrase" | easy.pl
```

Getting something like `lex => '_uw'` means that there is an unknown word. Or a word might be tagged with the wrong category, for instance `adj` instead of `np`. This means that the word is not in the lexicon. You can fix this by editing `missing.lex` which you can find under `${PREFIX}/share/frmg/`.

To directly check an entry in the lexicon, type:

```
echo "Luc" | lexed -d ${PREFIX}/lib/lefff-frmg -p dico.xlfg consult
```

You will get somehting like:

```
np      [pred='Luc_____1<Suj:(sn)>',cat=np,@loc,@ms]    Luc_____1
```

You can find information about `lexed` usage typing:

```
lexed -usage
```

5 Alpage components

5.1 Lefff, a French syntactic lexicon

Lefff (Lexique des formes fléchies du francais - Lexicon of French inflected forms) is a wide-coverage morphosyntactic and syntactic lexicon, whose architecture relies on properties inheritance, which makes it more compact and more easily maintainable and allows to describe lexical entries independantly from the formalisms it is used for.

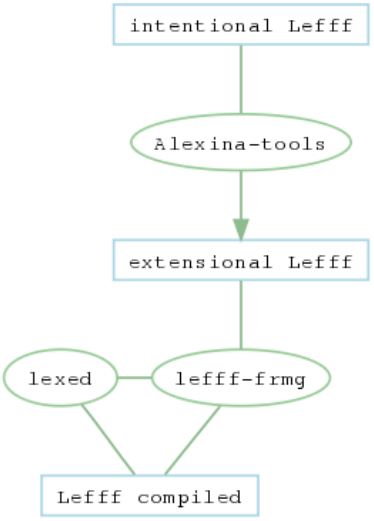

Figure 4: lefff

5.1.1 Description

Lefff is described in an **intentional** way that allows for factorization of information, thanks to a hierarchical inheritance structure. The intentional lexicon is a lexicon of lemmas, whereas the **extensional** lexicon is a lexicon of inflected forms.

Figure 4 shows two steps. First, Lefff is compiled into an extensional version thanks to Alexina-tools. Then it is compiled into an automata with lefff-frmg and lexed to work with the parsers based on the metagrammar frmg.

5.1.2 Access the lexicon

The lexicon Lefff can be accessed with the lexicalizer Lexed:

```
echo "industrie" | lexed -d ${PREFIX}/lib/lefff-frmg -p dico.xlfg consult
```

You will get (the output stands on a single line):

```
100     nc
[pred='industrie_____1<Objde:(de-sinf|de-sn),Objà:(à-sinf)>',
cat=nc,@fs]        industrie_____1Default  fs      %default
```

`${PREFIX}/lib/lefff-frmg` is where the compiled Lefff (`dico.xlfg.fsa`) is installed.

The extensional files installed by Lefff can be found under : `${PREFIX}/share/lefff/`.

5.1.3 More information

- Lefff is described in this article: `http://alpage.inria.fr/biblio?key=LREC06:Lefff`.

- Home page : `http://alpage.inria.fr/~sagot/lefff.html`
- INRIA GForge project page : `http://alexina.gforge.inria.fr`

5.2 sxpipe, a preprocessing chain

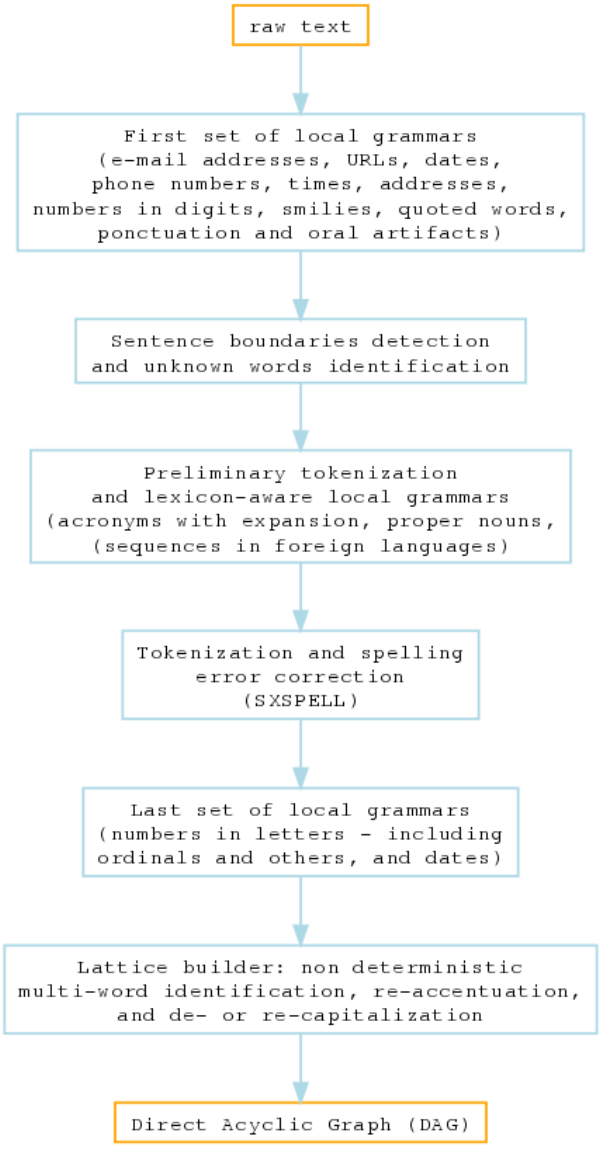

Figure 5: sxpipe

5.2.1 Description

sxpipe is a full-featured architecture to preprocess text before parsing. It converts raw noisy corpus into a Direct Acyclic Graph (**DAG**) that will be used

as input by the parser. It includes sequentially named-entity recognition, tokenization and sentence boundaries detection, lexicon-aware named-entity recognition, spelling correction, and non-deterministic multi-words processing, re-accentuation and de-/re- capitalization.

Though our system currently deals with French language, almost all components are in fact language-independent, and the others can be straightforwardly adapted to almost any inflectional language. sxpipe versions for Polish and Slovak are currently being developed, and very preliminary versions can already be tested. The output is a DAG of lexical entries.

The sxpipe version for French is briefly described in a paper available here: `http://alpage.inria.fr/biblio?key=sagot05:LT`

5.2.2 Usage

To get a DAG (Direct Acyclic Graph), try the following command:

```
echo "Il mange une pomme." | sxpipe
```

You will get something like:

```
({<F id="E1F1">Il</F>} _Uw | {<F id="E1F1">Il</F>} il) {<F id="E1F2">mange</F>} mange {
d="E1F3">une</F>} une {<F id="E1F4">pomme</F>} pomme {<F id="E1F5">.</F>} .
```

For a friendlier output:

```
echo "Il mange une pomme." | sxpipe | dag2udag
```

The output you get is then easier to read and process:

```
## DAG BEGIN
1       {<F id="E1F1">Il</F>}   il      2
2       {<F id="E1F2">mange</F>}        mange   3
3       {<F id="E1F3">une</F>}  une     4
4       {<F id="E1F4">pomme</F>}        pomme   5
5       {<F id="E1F5">.</F>}    .       6
## DAG END
```

When you have several sentences:

```
echo "Il mange une pomme. Elle arrive dans une heure." | sxpipe | dag2udag
```

```
## DAG BEGIN
1       {<F id="E1F1">Il</F>}   il      2
2       {<F id="E1F2">mange</F>}        mange   3
3       {<F id="E1F3">une</F>}  une     4
4       {<F id="E1F4">pomme</F>}        pomme   5
5       {<F id="E1F5">.</F>}    .       6
## DAG END
## DAG BEGIN
```

```
1       {<F id="E2F1">Elle</F> <F id="E2F1">Elle</F>}   elle    2
1       {<F id="E2F1">Elle</F> <F id="E2F1">Elle</F>}   Elle    2
2       {<F id="E2F2">arrive</F>}       arrive  3
3       {<F id="E2F3">dans</F>} dans    4
4       {<F id="E2F4">une</F>}  une     5
5       {<F id="E2F5">heure</F>}        heure   6
6       {<F id="E2F6">.</F>}    .       7
## DAG END
```

Display an image (Graphviz and ImageMagick must be installed):

```
echo "Il mange une pomme" | sxpipe | dag2dot
```

You will get something like figure 6.

Figure 6: dag2dot

5.3 easy.pl, a lexer

easy.pl, provided with package frmg, is a small segmenter tool that can take a sentence or a DAG (Directed Acyclic Graph) as input.

5.3.1 Usage

He is an example with input `il écrira`:

```
$ echo "il écrira" | easy.pl

*** Loading Finite State Automata
"stdin":
*** Load table in memory
*** Searching from stdin
%% Token database generated by
/home/pomerol/alpage/exportbuild/share/frmg/easy.pl
%% Sentence 1

'C'(0,
    lemma{ lex     => il,
           truelex => il,
           lemma   => cln,
           cat     => cln,
           top     => cln{case => nom, gender => masc, number => sg,
person => 3},
```

```
           anchor  => tag_anchor{ name => ht{arg0 => arg{kind => (-),
pcas => (-)}, arg1 => arg{kind => (-), pcas => (-)}, arg2 => arg{kind
=> (-),
pcas => (-)}, refl => (-)}, coanchors => [], equations => [] }
         },
    1
   ).

'C'(1,
    lemma{ lex     => 'écrira',
           truelex => 'écrira',
           lemma   => 'écrire',
           cat     => v,
           top     => v{mode => indicative, number => sg, person => 3,
tense => future},
           anchor  => tag_anchor{ name => ht{arg0 => arg{kind => subj,
pcas => (-)}, arg1 => arg{kind => kind[obj,prepvcomp,scomp,(-)],
pcas => prep[de,(-)]}, arg2 => arg{kind => kind[obj,prepobj,(-)],
pcas => prep['à',(-)]}, imp => (-), refl => (-)}, coanchors => [],
equations => [] }
         },
    2
   ).


'N'(2).
```

easy.pl comes with some options. If the preprocessing chain sxpipe is not installed, you can run easy.pl with option `nosxpipe`:

```
echo "il écrira" | easy.pl --nosxpipe
```

To check available options, pleasy type:

```
perldoc easy.pl
```

5.3.2 Process

easy.pl calls the preprocessing chain sxpipe (see next section) and gets this result:

```
## DAG BEGIN
1       {<F id="E1F1">il</F>}   il      2
2       {<F id="E1F2">écrira</F>}       écrira  3
## DAG END
```

Then it retrieves lemma information from the lexicon Lefff with tool lexed. Here are the results for `il` and `écrira`:

```
100     cln     [pred='cln_____1',case=nom,@3ms]        cln_____1
Default 3ms     %default|100    ilimp  [imp=+,@3ms]     ilimp_____1
                %Default 3ms     %default

100     v
[pred='écrire_____1<Suj:cln|sn,Obj:(cla|de-sinf|scompl|sn),
Objà:(cld|à-sn)>',@CompInd,@CtrlObjàObj,@pers,cat=v,@F3s]
écrire_____1    ThirdSing       F3s     %actif
```

5.4 DyALog

DyALog is an environment to compile and run logic programs and natural language tabular parsers for various grammatical formalisms (DCGs, TAGs, TIGs, RCGs). In Alpage Linguistic Processing Chain, it is used to produce two parsers based on the metagrammar frmg. See next section.

Home page: `http://alpage.inria.fr/dyalog.en.html`

5.5 FRMG, a French metagrammar

The Meta-Grammar is an abstract grammar designed to be compiled into TAG. You can find the original metagrammar in frmg sources:
`${PREFIX}/share/frmg/frgram.smg`

5.6 From the metagrammar to the parsers

The French metagrammar (MG) frmg comes in smg format (simplified metagrammar) and goes through several transformations (see next figure) to produce a parser.

1. frmg is compiled in xml format with tool smg2xml
2. it is converted in a DyALog format mg.pl with mg2dyalog
3. it is compiled with mgcomp (MG compiler) into a XML representation [TAGML] of a Tree Adjoining Grammar [TAG], file extension .tag.xml.
4. the grammar is then converted in a DyALog format (file extension .tag) with tag_converter.
5. then it is compiled into a TAG or hybrid TIG/TAG parser (tag_parser or tig_parser) with dyacc, provided by DyALog.

5.7 The parsers tig_parser and tag_parser

The parsers are generated with DyALog from the metagrammar frmg (See previous section).

Use tig_parser to get a shared derivation forest (option `-forest`):

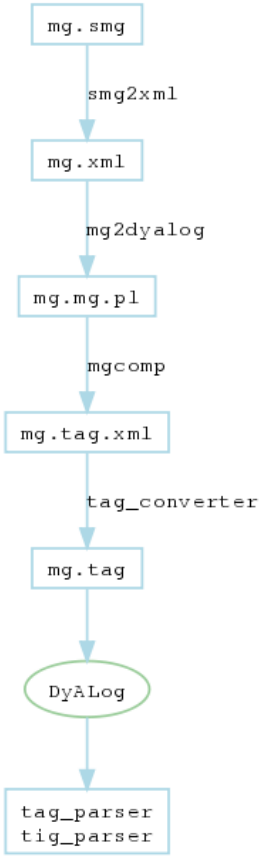

Figure 7: From the metagrammar to the parsers

```
echo "il mange une pomme" | easy.pl |\
tig_parser ${PREFIX}/share/frmg/small_header.tag - -forest
```

To do partial parsing, use option `-robust`:

```
echo "il mange une pomme" | easy.pl |\
tig_parser ${PREFIX}/share/frmg/small_header.tag - -robust
```

Type the following command to list all the options:

```
tig_parser -help
```

It is recommended to use the forest converter to transform the forest into useful formats.

5.8 The forest converter

forest_convert.pl is a perl script provided by package forest_utils. It allows to onvert the forests produced by TAG parsers into many useful formats.

5.8.1 Usage

```
./forest_convert.pl [options]
./forest_convert.pl -i <input forest> -f <from option> -t <to option>
```

where the options are:

- -i <in>
- -o <out>
- -f <from> where <from> can be: lp|rcg|xtag|xml|xmldep
- -t <to> where <to> can be: xml|dep|xmldep|lpdep|stats|tree|grammar|tagger|yesno|line|dump|info

5.9 parserd, a server of parsers

5.9.1 Description

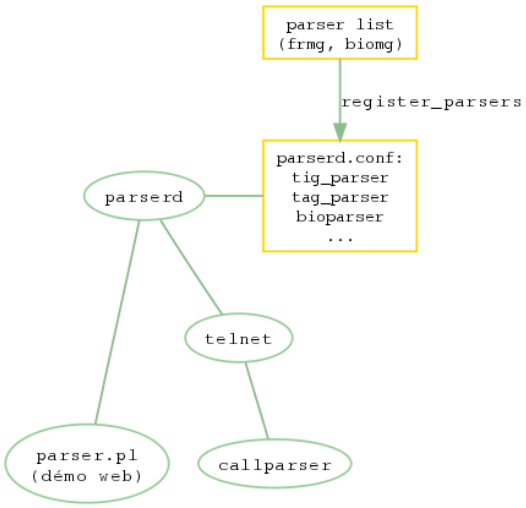

Figure 8: Server of parsers

parserd has two clients, parser.pl and callparser, and there are three possibilities to communicate with the server:

- through the web interface provided by **parser.pl**
- with command **callparser**, which uses telnet to communicate with parserd
- directly through **telnet**

parserd actually registers two parsers provided by **frmg**: **tag_parser** and **tig_parser**. These parsers are registered in parserd configuration file (parserd.conf) with a perl script called `register_parsers`. parserd.conf can be found under `${PREFIX}/etc/`. It contains:

```
port     = 8999    # port under which the server runs
user     = toto    # user of the server
group    = alpage # group of the server
# Name of log file [default = /var/log/parserd.log]
log_file = ${PREFIX}/var/log/parserd.log
# maxclients = 2 # Max number of clients [default = 1]
pid_file = ${PREFIX}/var/run/parserd.pid
path     = ${PREFIX}/bin
path     = /usr/local/bin
path     = /usr/ucb

# Parser list

parsers = frmgel frmgelr frmgtel frmgtelr

[frmgel]
label        = FRMG TAG - Hybrid Strategy - EASY Lexer
cmd          = ${PREFIX}/bin/tag_parser
options      = ${PREFIX}/share/frmg/small_header.tag - -forest
forest       = lp
examples     = ${PREFIX}/share/frmg/tests.txt
lexer        = persistent ${PREFIX}/share/frmg/easy.pl
origin       = frmg-1.0.3
grammar      = /perl/frmg/tree.pl
language     = french

[frmgelr]
label        = FRMG TAG - Hybrid Strategy - EASY Lexer - Robust
cmd          = ${PREFIX}/bin/tag_parser
options      = ${PREFIX}/share/frmg/small_header.tag - -forest -robust
forest       = lp
examples     = ${PREFIX}/share/frmg/tests.txt
lexer        = persistent ${PREFIX}/share/frmg/easy.pl
origin       = frmg-1.0.3
grammar      = /perl/frmg/tree.pl
language     = french

[frmgtel]
label        = FRMG TIG - Hybrid Strategy - EASY Lexer
cmd          = ${PREFIX}/bin/tig_parser
options      = ${PREFIX}/share/frmg/small_header.tag - -forest
forest       = lp
examples     = ${PREFIX}/share/frmg/tests.txt
lexer        = persistent ${PREFIX}/share/frmg/easy.pl
origin       = frmg-1.0.3
grammar      = /perl/frmg/tree.pl
easy         = ${PREFIX}/bin/easyforest
```

```
disambiguate = ${PREFIX}/bin/easyforest -- -depxml
language     = french

[frmgtelr]
label        = FRMG TIG - Hybrid Strategy - EASY Lexer - Robust
cmd          = ${PREFIX}/bin/tig_parser
options      = ${PREFIX}/share/frmg/small_header.tag - -forest -robust
forest       = lp
examples     = ${PREFIX}/share/frmg/tests.txt
lexer        = persistent ${PREFIX}/share/frmg/easy.pl
origin       = frmg-1.0.3
grammar      = /perl/frmg/tree.pl
easy         = ${PREFIX}/bin/easyforest
disambiguate = ${PREFIX}/bin/easyforest -- -depxml
language     = french
```

5.9.2 Usage

To **register parsers** in parserd.conf:

```
register_parsers -a <partial_conf_file> ${PREFIX}/etc/parserd.conf
```

frmg partial conf file contains a list of parsers:

```
# Parser list

parsers = frmgel frmgelr frmgtel frmgtelr

[frmgel]
label        = FRMG TAG - Hybrid Strategy - EASY Lexer
cmd          = ${PREFIX}/bin/tag_parser
options      = ${PREFIX}/share/frmg/small_header.tag - -forest
forest       = lp
examples     = ${PREFIX}/share/frmg/tests.txt
lexer        = persistent ${PREFIX}/share/frmg/easy.pl
origin       = frmg-1.0.3
grammar      = /perl/frmg/tree.pl
language     = french

[frmgelr]
label        = FRMG TAG - Hybrid Strategy - EASY Lexer - Robust
cmd          = ${PREFIX}/bin/tag_parser
options      = ${PREFIX}/share/frmg/small_header.tag - -forest -robust
forest       = lp
examples     = ${PREFIX}/share/frmg/tests.txt
lexer        = persistent ${PREFIX}/share/frmg/easy.pl
origin       = frmg-1.0.3
```

```
grammar      = /perl/frmg/tree.pl
language     = french

[frmgtel]
label        = FRMG TIG - Hybrid Strategy - EASY Lexer
cmd          = ${PREFIX}/bin/tig_parser
options      = ${PREFIX}/share/frmg/small_header.tag - -forest
forest       = lp
examples     = ${PREFIX}/share/frmg/tests.txt
lexer        = persistent ${PREFIX}/share/frmg/easy.pl
origin       = frmg-1.0.3
grammar      = /perl/frmg/tree.pl
easy         = ${PREFIX}/bin/easyforest
disambiguate = ${PREFIX}/bin/easyforest -- -depxml
language     = french

[frmgtelr]
label        = FRMG TIG - Hybrid Strategy - EASY Lexer - Robust
cmd          = ${PREFIX}/bin/tig_parser
options      = ${PREFIX}/share/frmg/small_header.tag - -forest -robust
forest       = lp
examples     = ${PREFIX}/share/frmg/tests.txt
lexer        = persistent ${PREFIX}/share/frmg/easy.pl
origin       = frmg-1.0.3
grammar      = /perl/frmg/tree.pl
easy         = ${PREFIX}/bin/easyforest
disambiguate = ${PREFIX}/bin/easyforest -- -depxml
language     = french
```

To start/stop the server:

```
parserd_setenv start
parserd_setenv stop
```

parserd is provided with two clients: the command **callparser** and the web interface **parser.pl**. callparser usage is described in the tutorial below.

Another way is to directly connect to parserd with **telnet**:

```
telnet localhost <port>
```

The default port is 8999. Then press the return key twice an try:

```
frmgtel il mange une pommme
```

To set a forest format and then print it:

```
set forest dependency
last forest
```

Please type `help` to list commands and options.

5.9.3 Troubleshooting

If you cannot start the server of parser or get the message `'parserd dead but subsys locked'`, remove the lock and try again.

```
rm -f ${PREFIX}/var/lock/subsys/parserd
```

6 Glossary

ALPC: Alpage Linguistic Processing Chain :)

CFG: Context-Free Grammar

CKY: Cocke-Kasami-Younger

DAG: Directed Acyclic Graph

DCG: Definite Clause Grammar

DP: Dynamic Programming

HPSG: Head-driven Phrase Structure Grammar

LFG: Lexical Functional Grammar

LIG: Linear Indexed Grammar

MAF: Morpho-Syntactic Annotation Framework. MAF is a standard proposal developed by ISO TC37SC4 commitee. See www.tc37sc4.org

MG (Meta-Grammar): Introduced by Marie-Hélène Candito in 1999, the Meta-Grammar is a more abstract grammar, designed to be compiled into TAG

RCG: Range Concatenation Grammar

TAG: Tree Adjoining Grammar

TIG: Tree Insertion Grammar