Install Dutch nlp modules on Lisa

Paul Huygen <paul.huygen@huygen.nl>

10th November 2014
10:41 h.

Abstract

This is a description and documentation of the installation of the current NLP modules on Lisa, so that they can be used in pipelines.

Contents

1 Introduction

This document describes the current set-up of pipeline that annotates dutch texts in order to extract knowledge. The pipeline has been set up by the Computational Lexicology an Terminology Lab (CLTL [1]) as part of the newsreader [2].

Apart from describing the pipeline set-up, the document actually constructs the pipeline. The described version has been made with an aim to run it on a specific supercomputer (Lisa, Surfsara, Amsterdam [3]), but it can probably be implemented on other unix-like systems without problems.

The installation has been parameterized. The locations and names that you read (and that will be used to build the pipeline) have been read from variables in file `inst.m4` in the nuweb directory.

1.1 List of the modules to be installed

Table 1 lists the modules in the pipeline. The column *source* indicates the origin of the module.

1. `http://wordpress.let.vupr.nl`
2. http://www.newsreader-project.eu
3. https://surfsara.nl/systems/lisa

module	**directory**	**source**	**script**	**Details**
Tokenizer	`ixa-pipe-tok`	EHU	tok	
morphosyntactic parser	`morphosyntactic_parser_nl`	Github	`mor`	
alpinohack	`clean_hack`	This doc.	alpinohack	[4]
NERC	`../modules/jars`	TAR	nerc	
WSD	`svm_wsd`	TAR	wsd	
Onto	`ontotagger`	TAR	onto	
Heidel	`NAF-HeidelTime`	Github	heideltime	
SRL	`vua-srl-nl`	Github	srl	
NED	`ned`	EHU	ned	
Nom. coref	`/dev/null`	None	nomcoref	
Ev. coref	`/dev/null`	None	evcoref	
Opinion miner	`/dev/null`	None	opinimin	
Framenet sem. role label.	`/dev/null`	None	fsrl	

Table 1: List of the modules to be installed. Column description: **directory:** Name of the subdirectory below subdirectory `modules` in which it is installed; **Source:** From where the module has been obtained; **script:** Script to be included in a pipeline.

The modules are obtained in one of the following ways:

1. If possible, the module is directly obtained from an open-source repository like Github.
2. Some modules are available from the dedicated repository on . A username and password are needed to access these modules. This is indicated as EHU.
3. Some modules have not been officially published in a repository or the repositrory is not yet known by the author. These modules have been packed in a tar-ball that can be obtained by the author. This is indicated as TAR.

The modules themselves use other utilities like dependency-taggers and POS taggers. These utilities are listed in table 2.

module	**directory**	**source**	**Details**
KafNafParserPy	`python/KafNafParserPy`	Github	
Alpino	`Alpino`	RUG	
Ticcutils	`ticcutils-0.7`	ILK	
Timbl	`timbl-6.4.6`	ILK	
Treetagger			

Table 2: List of the modules to be installed. Column description: **directory:** Name of the subdirectory below `mod` in which it is installed; **Source:** From where the module has been obtained; **script:** Script to be included in a pipeline.

Table 3 lists the source of the modules and utilities that can be installed from an open source.

module	**source**	**URL**
Tokenizer	Github	https://github.com/ixa-ehu/ixa-pipe-tok.git
Morphosynt. p.	Github	`https://github.com/cltl/morphosyntactic_parser_nl.git`
heideltime.	Github	`https://github.com/cltl/morphosyntactic_parser_nl.git`
Alpino	RUG	`Alpino-x86_64-linux-glibc2.5-20548-sicstus.tar.gz`
Ticcutils	ILK	ticcutils-0.7.tar.gz
Timble	ILK	timbl-6.4.6.tar.gz

Table 3: Sources of the modules

1.2 File-structure of the pipeline

The files that make up the pipeline are organised in set of directories:

nuweb: This directory comntains this document and everything to create the pipeline from the open sources of the modules.

modules: Contains the program code of each module in a subdirectory. Furthermore, it contains a subdirectory `python` for python software-modules, subdirectory `jars` for jar files and subdirectory /usrlocal/ for binaries and libs that are used by modules.

bin: Contains for each of the modules a script that reads NAF input, passes it to the module in the `modules` directory and produces the output on standard out. Furthermore, the subdirectory contains the script `install-modules` that performs the installation, and a script `test` that shows that the pipeline works in a trivial case.

nuweb: Contains this document, the nuweb source that creates the documents and the sources and a Makefile to perform the actions.

⟨ *directories to create* 4a ⟩ ≡

```
        ../modules ◇
```

Fragment defined by 4abcdef, 9h, 27c.
Fragment referenced in 33b.

⟨ *directories to create* 4b ⟩ ≡

```
        ../bin ◇
```

Fragment defined by 4abcdef, 9h, 27c.
Fragment referenced in 33b.

⟨ *directories to create* 4c ⟩ ≡

```
        ../modules/usrlocal ◇
```

Fragment defined by 4abcdef, 9h, 27c.
Fragment referenced in 33b.

⟨ *directories to create* 4d ⟩ ≡

```
        ../modules/usrlocal/bin ◇
```

Fragment defined by 4abcdef, 9h, 27c.
Fragment referenced in 33b.

⟨ *directories to create* 4e ⟩ ≡

```
        ../modules/usrlocal/lib ◇
```

Fragment defined by 4abcdef, 9h, 27c.
Fragment referenced in 33b.

⟨ *directories to create* 4f ⟩ ≡

```
        ../modules/python ../modules/jars ◇
```

Fragment defined by 4abcdef, 9h, 27c.
Fragment referenced in 33b.

Make Python utilities findable with the following macro:

⟨ *set pythonpath* 4g ⟩ ≡

```

        export PYTHONPATH=/mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa/modules/python:$PYTHONPATH
        ◇
```

Fragment referenced in 9f, 15b, 16a.

Similarly, make binaries findable:

⟨ *set local bin directory* 5a ⟩ ≡

```
export PATH=/mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa/modules/usrlocal/bin:$PATH
◇
```

Fragment referenced in 16a.

2 Installation

This section describes how the modules are obtained from their (open-)source and installed.

2.1 Installing vs. updating

When the install-script installs something that has already been installed, it moves the installed module to a temporary location and then tries to install the module from its source. If that is successfull it removes the vormer version of the module, otherwise it moves the old version back.

The following macro's can be used to move or remove modules, provided they are called when the modules directory is the default directory.

⟨ *move module* 5b ⟩ ≡

```
if
 [ -e @1 ]
then
   mv @1 old.@1
fi
◇
```

Fragment referenced in 6a, 8c, 21a.

⟨ *remove old module* 5c ⟩ ≡

```
rm -rf old.@1
◇
```

Fragment referenced in 6a, 8c, 21a.

⟨ *re-instate old module* 5d ⟩ ≡

```
mv old.@1 @1
MESS="Replaced previous version of @1"
⟨ logmess (5e $MESS ) 20e ⟩

◇
```

Fragment referenced in 6a, 8c, 21a.

2.2 Installation from Github

The following macro can be used to install a module from github. It needs as parameters:

1. Name of the module.
2. Name of the root directory.
3. Github URL to clone from.

⟨ *install from github* 6a ⟩ ≡
```
MODNAM=@1
DIRN=@2
GITU=@3
cd /mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa/modules
⟨ move module (6b $DIRN ) 5b ⟩
git clone $GITU
if
  [ $? -gt 0 ]
then
  ⟨ logmess (6c Cannot install current $MODNAM version ) 20e ⟩
  ⟨ re-instate old module (6d $DIRN ) 5d ⟩
else
  ⟨ remove old module (6e $DIRN ) 5c ⟩
fi

◇
```
Fragment referenced in 9b, 11d, 14c, 15d, 17a.

2.3 Installation from the snapshot

For some modules a public repository is not available or not known. They must be installed from a tarball with snapshots that can be obtained from the author. Let us first check whether we have the snapshot and complain if we don't. We expect the file `/mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-L`

⟨ *unpack snapshots or die* 6f ⟩ ≡
```
cd /mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa
if
  [ -e nl_pipeline_snapshots.tgz ]
then
  tar -zxf nl_pipeline_snapshots.tgz
fi
if
  [ ! -e snapshots ]
then
  echo "No module snapshots"
  exit 1
fi
◇
```
Fragment referenced in 7a.

2.4 The installation script

The installation is performed by script `install-modules`

"../bin/install-modules" 7a≡
```
#!/bin/bash
⟨ variables of install-modules 20d ⟩
⟨ unpack snapshots or die 6f ⟩
⟨ install the tokenizer 7c ⟩
⟨ install kafnafparserpy 17a ⟩
⟨ install Alpino 8c ⟩
⟨ install the morphosyntactic parser 9b ⟩
⟨ install the NERC module 10d ⟩
⟨ install the WSD module 11d ⟩
⟨ install the spotlight server 12b ⟩

⟨ install the NED module 12e ⟩
⟨ install the onto module 13c ⟩
⟨ install the heideltime module 14c ⟩
⟨ install the srl module 15d ⟩
⟨ install the treetagger utility 17g, ... ⟩
⟨ install the ticcutils utility 19a ⟩
⟨ install the timbl utility 19b ⟩

◇
```

⟨ *make scripts executable* 7b ⟩ ≡
```
chmod 775  ../bin/install-modules
◇
```
Fragment defined by 7b, 8b, 9g, 10c, 11c, 12a, 13b, 14b, 15c, 16f, 17f.
Fragment referenced in 33c.

2.5 Install tokenizer

2.5.1 Module

The tokenizer is just a jar that has to be run in Java. Although the jar is directly available from `http://ixa2.si.ehu.es/ixa-pipes/download.html`, we prefer to compile the package in order to make this thing ready for reproducible set-ups.

Not yet included in this script is the set-up of an environment to use the specified version of Java (Oracle 1.7) and Maven (3). For now, we assume that it is there. This is a todo item.

To install the tokenizer, we proceed as follows:

1. Clone the source from github into a temporary directory.
2. Compile to produce the jar file with the tokenizer.
3. move the jar file into the jar directory.
4. remove the tempdir with the sourcecode.

⟨ *install the tokenizer* 7c ⟩ ≡
```
tempdir=‘mktemp -d -t tok.XXXXXX‘
cd $tempdir
git clone https://github.com/ixa-ehu/ixa-pipe-tok.git
cd ixa-pipe-tok
mvn clean package
mv target/ixa-pipe-tok-1.5.3.jar /mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa/modules/jar
cd /mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa
◇
```
Fragment referenced in 7a.

2.5.2 Script

The script runs the tokenizerscript.

"../bin/tok" 8a≡
```
#!/bin/bash
ROOT=/mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa
JARFILE=/mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa/modules/jars/ixa-pipe-tok-1.5.3.jar
java -jar $JARFILE tok -l nl --inputkaf
◇
```

⟨ *make scripts executable* 8b ⟩ ≡
```
chmod 775  ../bin/tok
◇
```
Fragment defined by 7b, 8b, 9g, 10c, 11c, 12a, 13b, 14b, 15c, 16f, 17f.
Fragment referenced in 33c.

2.6 Install Alpino

Install Alpino from the website of Gertjan van Noort.

2.6.1 Module

⟨ *install Alpino* 8c ⟩ ≡
```
SUCCES=0
cd /mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa/modules
⟨ move module (8d Alpino ) 5b ⟩
wget http://www.let.rug.nl/vannoord/alp/Alpino/binary/versions/Alpino-x86_64-linux-glibc2.5-20548-sic
SUCCES=$?
if
  [ $SUCCES -eq 0 ]
then
  tar -xzf Alpino-x86_64-linux-glibc2.5-20548-sicstus.tar.gz
  SUCCES=$?
  rm -rf Alpino-x86_64-linux-glibc2.5-20548-sicstus.tar.gz
fi
if
  [ $SUCCES -eq 0 ]
then
  ⟨ logmess (8e Installed Alpino ) 20e ⟩
  ⟨ remove old module (8f Alpino ) 5c ⟩
else
  ⟨ re-instate old module (8g Alpino ) 5d ⟩
fi
◇
```
Fragment referenced in 7a.

Currently, alpino is not used as a pipeline-module on its own, but it is included in other pipeline-modules. Modules that use Alpino should set the following variables:

⟨ *set alpinohome* 9a ⟩ ≡

```
export ALPINO_HOME=/mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa/modules/Alpino
◇
```

Fragment referenced in 9f.
Defines: `ALPINO_HOME` Never used.

2.7 Morphosyntactic parser

2.7.1 Module

⟨ *install the morphosyntactic parser* 9b ⟩ ≡

```
⟨ install from github (9c morphsynparser,9d morphosyntactic_parser_nl,9e https://github.com/cltl/morphosyntac
◇
```

Fragment referenced in 7a.

2.7.2 Script

"../bin/mor" 9f≡

```
#!/bin/bash
ROOT=/mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa
MODDIR=/mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa/modules/morphosyntactic_parser_nl
⟨ set alpinohome 9a ⟩
⟨ set pythonpath 4g ⟩
cat | python $MODDIR/core/morph_syn_parser.py
◇
```

⟨ *make scripts executable* 9g ⟩ ≡

```
chmod 775  ../bin/mor
◇
```

Fragment defined by 7b, 8b, 9g, 10c, 11c, 12a, 13b, 14b, 15c, 16f, 17f.
Fragment referenced in 33c.

2.8 Alpino hack

Install a hack that removes output from Alpino that cannot be interpreted by following modules. It is just a small python script.

2.8.1 Module

⟨ *directories to create* 9h ⟩ ≡

```
../modules/alpinohack ◇
```

Fragment defined by 4abcdef, 9h, 27c.
Fragment referenced in 33b.

"../modules/alpinohack/clean_hack.py" 10a≡
```
#!/usr/bin/python
import sys

input = sys.stdin

output = ''

for line in input:
    line = line.replace('"--','"#')
    line = line.replace('--"','#"')
    output += line

print output

◇
```
Uses: print 27a.

2.8.2 Script

"../bin/alpinohack" 10b≡
```
#!/bin/bash
ROOT=/mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa
HACKDIR=/mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa/modules/alpinohack
cat | python  $HACKDIR/clean_hack.py

◇
```

⟨ *make scripts executable* 10c ⟩ ≡
```
chmod 775  ../bin/alpinohack
◇
```
Fragment defined by 7b, 8b, 9g, 10c, 11c, 12a, 13b, 14b, 15c, 16f, 17f.
Fragment referenced in 33c.

2.9 Named entity recognition (NERC)

2.9.1 Module

We do not (yet) have the source code of the NER module. A snapshot is comprised in a jar library.

⟨ *install the NERC module* 10d ⟩ ≡
```

cp  /mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa/snapshots/nerc/ixa-pipe-nerc-1.1.0.jar /
◇
```
Fragment referenced in 7a.

2.9.2 Script

Unfortunately, this module does not accept the NAF version that the previous module supplies.

⟨ *gawk script to patch NAF for nerc module* 11a ⟩ ≡
```
patchscript='{gsub("wf id=", "wf wid="); gsub("term id=", "term tid="); print}'
◇
```
Fragment referenced in 11b.
Uses: `print` 27a.

`"../bin/nerc"` 11b≡
```
#!/bin/bash
ROOT=/mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa
JARDIR=/mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa/modules/jars
⟨ gawk script to patch NAF for nerc module 11a ⟩
cat | gawk "$patchscript" | java -jar $JARDIR/ixa-pipe-nerc-1.1.0.jar tag
◇
```

⟨ *make scripts executable* 11c ⟩ ≡
```
chmod 775  ../bin/nerc
◇
```
Fragment defined by 7b, 8b, 9g, 10c, 11c, 12a, 13b, 14b, 15c, 16f, 17f.
Fragment referenced in 33c.

2.10 Wordsense-disambiguation

Install WSD from its Github source.

2.10.1 Module

⟨ *install the WSD module* 11d ⟩ ≡
```
⟨ install from github (11e wsd,11f svm_wsd,11g https://github.com/cltl/svm_wsd.git ) 6a ⟩
cd /mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa/modules/svm_wsd
./install_naf.sh
◇
```
Fragment referenced in 7a.

2.10.2 Script

`"../bin/wsd"` 11h≡
```
#!/bin/bash
# WSD -- wrapper for word-sense disambiguation
# 8 Jan 2014 Ruben Izquierdo
# 16 sep 2014 Paul Huygen
ROOT=/mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa
WSDDIR=/mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa/modules/svm_wsd
WSDSCRIPT=dsc_wsd_tagger.py
cat | python $WSDDIR/$WSDSCRIPT --naf
◇
```

⟨ *make scripts executable* 12a ⟩ ≡
```
chmod 775  ../bin/wsd
◇
```
Fragment defined by 7b, 8b, 9g, 10c, 11c, 12a, 13b, 14b, 15c, 16f, 17f.
Fragment referenced in 33c.

2

2.11 Spotlight

Spotlight is not itself a pipeline-module, but it is needed in the NED module. Now I make a shortcut from the snapshot.

⟨ *install the spotlight server* 12b ⟩ ≡
```

cp -r /mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa/snapshots/spotlight  /mnt/kyoto/projec
◇
```
Fragment referenced in 7a.

⟨ *start the spotlight server* 12c ⟩ ≡
```
cd /mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa/modules/spotlight
java -jar -Xmx8g dbpedia-spotlight-0.7-jar-with-dependencies-candidates.jar nl http://localhost:2060/
◇
```
Fragment referenced in 12d.

⟨ *check/start the spotlight server* 12d ⟩ ≡
```
spottasks=‘netstat -an | grep :2060 | wc -l‘
if
  [ $spottasks -eq 0 ]
then
  ⟨ start the spotlight server 12c ⟩
  sleep 180
fi
◇
```
Fragment referenced in 13a.

2.12 NED

The NED module wants to consult the dbpedia spotlight server, so that one has to be installed somewhere. For this moment, let us suppose that it has been installed on localhost.

2.12.1 Module

⟨ *install the* NED *module* 12e ⟩ ≡
```

cp /mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa/snapshots/ned/ixa-pipe-ned-1.0.jar /mnt/k
mkdir -p /mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa/modules/ned
cd /mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa/modules/ned
wget http://ixa2.si.ehu.es/ixa-pipes/models/wikipedia-db.v1.tar.gz
tar -xzf wikipedia-db.v1.tar.gz
◇
```
Fragment referenced in 7a.

2.12.2 Script

"../bin/ned" 13a≡
```
#!/bin/bash
ROOT=/mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa
JARDIR=/mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa/modules/jars
⟨ check/start the spotlight server 12d ⟩
cat | java -jar $JARDIR/ixa-pipe-ned-1.0.jar  -p 2060 -e candidates -i /mnt/kyoto/projecten/pipelines
◇
```

⟨ *make scripts executable* 13b ⟩ ≡
```
chmod 775  ../bin/ned
◇
```
Fragment defined by 7b, 8b, 9g, 10c, 11c, 12a, 13b, 14b, 15c, 16f, 17f.
Fragment referenced in 33c.

2.13 Ontotagger

We do not yet have a source-repository of the Ontotagger module. Therefore, install from a snapshot on Lisa.

2.13.1 Module

⟨ *install the onto module* 13c ⟩ ≡
```

cp -r /mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa/snapshots/ontotagger /mnt/kyoto/projec
◇
```
Fragment referenced in 7a.

2.13.2 Script

"../bin/onto" 14a≡
```
#!/bin/bash
ROOT=/mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa
ONTODIR=/mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa/modules/ontotagger
JARDIR=$ONTODIR/lib
RESOURCESDIR=$ONTODIR/resources
PREDICATEMATRIX="$RESOURCESDIR/PredicateMatrix.v1.1/PredicateMatrix.v1.1.role.nl-1.merged"
GRAMMATICALWORDS="$RESOURCESDIR/grammaticals/Grammatical-words.nl"
TMPFIL=‘mktemp -t stap6.XXXXXX‘
cat >$TMPFIL

CLASSPATH=$JARDIR/ontotagger-1.0-jar-with-dependencies.jar
JAVASCRIPT=eu.kyotoproject.main.KafPredicateMatrixTagger


JAVA_ARGS="--mappings \"fn;pb;nb\" "
JAVA_ARGS="$JAVA_ARGS  --key odwn-eq"
JAVA_ARGS="$JAVA_ARGS  --version 1.1"
JAVA_ARGS="$JAVA_ARGS  --predicate-matrix $PREDICATEMATRIX"
JAVA_ARGS="$JAVA_ARGS  --grammatical-words $GRAMMATICALWORDS"
JAVA_ARGS="$JAVA_ARGS  --naf-file $TMPFIL"
java -Xmx1812m -cp $CLASSPATH $JAVASCRIPT $JAVA_ARGS

rm -rf $TMPFIL

◇
```

⟨ *make scripts executable* 14b ⟩ ≡
```
chmod 775  ../bin/onto
◇
```
Fragment defined by 7b, 8b, 9g, 10c, 11c, 12a, 13b, 14b, 15c, 16f, 17f.
Fragment referenced in 33c.

2.14 Heideltime

2.14.1 Module

⟨ *install the heideltime module* 14c ⟩ ≡

⟨ *install from github* (14d `heideltime`,14e `NAF-HeidelTime`,14f `git@github.com:PaulHuygen/NAF-HeidelTime.git`) 6
⟨ *adapt heideltime's config.props* 15a ⟩

◇

Fragment referenced in 7a.

⟨ *adapt heideltime's config.props* 15a ⟩ ≡
```
CONFIL=NAF-HeidelTime/config.props
tempfil=`mktemp -t heideltmp.XXXXXX`
mv $CONFIL $tempfil
MODDIR=/mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa/modules
TREETAGDIR=treetagger
AWKCOMMAND='/^treeTaggerHome/ {$0="treeTaggerHome = /mnt/kyoto/projecten/pipelines/dutch-nlp-modules-
gawk "$AWKCOMMAND" $tempfil >$CONFIL
◇
```
Fragment referenced in 14c.
Uses: `print` 27a.

2.14.2 Script

"`../bin/heideltime`" 15b≡
```
#!/bin/bash
ROOT=/mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa
HEIDELDIR=/mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa/modules/NAF-HeidelTime
TEMPDIR=`mktemp -t -d heideltmp.XXXXXX`
cd $HEIDELDIR
⟨ set pythonpath 4g ⟩
iconv -t utf-8//IGNORE | python $HEIDELDIR/HeidelTime_NafKaf.py $HEIDELDIR/heideltime-standalone/ $TE
◇
```

⟨ *make scripts executable* 15c ⟩ ≡
```
chmod 775  ../bin/heideltime
◇
```
Fragment defined by 7b, 8b, 9g, 10c, 11c, 12a, 13b, 14b, 15c, 16f, 17f.
Fragment referenced in 33c.

2.15 Semantic Role labelling

2.15.1 Module

⟨ *install the srl module* 15d ⟩ ≡

⟨ *install from github* (15e `srl`,15f `vua-srl-nl`,15g `https://github.com/newsreader/vua-srl-nl.git`) 6a ⟩
◇

Fragment referenced in 7a.

2.15.2 Script

First:

1. set the correct environment. The module needs python and timble.
2. create a tempdir and in that dir a file to store the input and a (SCV) file with the feature-vector.

"../bin/srl" 16a≡
```
#!/bin/bash
ROOT=/mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa
SRLDIR=/mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa/modules/vua-srl-nl
TEMPDIR=`mktemp -d -t SRLTMP.XXXXXX`
cd $SRLDIR
⟨ set local bin directory 5a ⟩
⟨ set pythonpath 4g ⟩
INPUTFILE=$TEMPDIR/inputfile
FEATUREVECTOR=$TEMPDIR/csvfile
TIMBLOUTPUTFILE=$TEMPDIR/timblpredictions
◇
```
File defined by 16abcde.

Create a feature-vector.

"../bin/srl" 16b≡
```
cat | tee  $INPUTFILE | python nafAlpinoToSRLFeatures.py > $FEATUREVECTOR
◇
```
File defined by 16abcde.

Run the trained model on the feature-vector.

"../bin/srl" 16c≡
```

timbl -mO:I1,2,3,4 -i e-mags_mags_press_newspapers.wgt -t $FEATUREVECTOR -o $TIMBLOUTPUTFILE >/dev/nu
◇
```
File defined by 16abcde.

Insert the SRL values into the NAF file.

"../bin/srl" 16d≡
```
python timblToAlpinoNAF.py $INPUTFILE $TIMBLOUTPUTFILE
◇
```
File defined by 16abcde.

Clean up.

"../bin/srl" 16e≡
```
rm -rf $TEMPDIR
◇
```
File defined by 16abcde.

⟨ *make scripts executable* 16f ⟩ ≡
```
chmod 775  ../bin/srl
◇
```
Fragment defined by 7b, 8b, 9g, 10c, 11c, 12a, 13b, 14b, 15c, 16f, 17f.
Fragment referenced in 33c.

2.16 KafNafParserPy

Several modules use KafNafParserpy to read and write NAF files.

2.16.1 Module

⟨ *install kafnafparserpy* 17a ⟩ ≡

⟨ *install from github* (17b `kafnafparserpy`,17c `python/KafNafParserPy`,17d `https://github.com/cltl/KafNafParserP`

◇

Fragment referenced in 7a.

3 Utilities

3.1 Test script

The following script pushes a single sentence through the modules of the pipeline.

"../bin/test" 17e≡

```
#!/bin/bash
ROOT=/mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa
BIND=$ROOT/bin
echo "De hond eet jus." | $BIND/tok | $BIND/mor | \
$BIND/alpinohack | $BIND/nerc  | $BIND/wsd | \
$BIND/onto  > $ROOT/test.onto
cat $ROOT/test.onto | $BIND/heideltime  > $ROOT/test.heidel
cat $ROOT/test.heidel | $BIND/srl  > $ROOT/test.srl
cat $ROOT/test.srl | $BIND/srl  > $ROOT/test.srl
```

◇

⟨ *make scripts executable* 17f ⟩ ≡

```
chmod 775  ../bin/test
```

◇

Fragment defined by 7b, 8b, 9g, 10c, 11c, 12a, 13b, 14b, 15c, 16f, 17f.
Fragment referenced in 33c.

3.2 Treetagger

3.2.1 Module

Installation goes as follows (See Treetagger's homepage:

1. Download and unpack the treetagger tarball. This generates the subdirectories `bin`, `cmd` and `doc`
2. Download and unpack the tagger-scripts tarball

The location where treetager comes from and the location where it is going to reside:

⟨ *install the treetagger utility* 17g ⟩ ≡

```
TREETAGDIR=treetagger
TREETAG_BASIS_URL=http://www.cis.uni-muenchen.de/%7Eschmid/tools/TreeTagger/data/
TREETAGURL=http://www.cis.uni-muenchen.de/%7Eschmid/tools/TreeTagger/data/
```

◇

Fragment defined by 17g, 18abcde.
Fragment referenced in 7a.

The source tarball, scripts and the installation-script:

⟨ *install the treetagger utility* 18a ⟩ ≡
```
TREETAGSRC=tree-tagger-linux-3.2.tar.gz
TREETAGSCRIPTS=tagger-scripts.tar.gz
TREETAG_INSTALLSCRIPT=install-tagger.sh
◇
```
Fragment defined by 17g, 18abcde.
Fragment referenced in 7a.

Parametersets:

⟨ *install the treetagger utility* 18b ⟩ ≡
```
DUTCHPARS_UTF_GZ=dutch-par-linux-3.2-utf8.bin.gz
DUTCH_TAGSET=dutch-tagset.txt
DUTCHPARS_2_GZ=dutch2-par-linux-3.2-utf8.bin.gz
◇
```
Fragment defined by 17g, 18abcde.
Fragment referenced in 7a.

Download everything in the target directory:

⟨ *install the treetagger utility* 18c ⟩ ≡
```

mkdir -p /mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa/modules/$TREETAGDIR
cd /mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa/modules/$TREETAGDIR
wget $TREETAGURL/$TREETAGSRC
wget $TREETAGURL/$TREETAGSCRIPTS
wget $TREETAGURL/$TREETAG_INSTALLSCRIPT
wget $TREETAGURL/$DUTCHPARS_UTF_GZ
wget $TREETAGURL/$DUTCH_TAGSET
wget $TREETAGURL/$DUTCHPARS_2_GZ
◇
```
Fragment defined by 17g, 18abcde.
Fragment referenced in 7a.

Run the install-script:

⟨ *install the treetagger utility* 18d ⟩ ≡
```
chmod 775 $TREETAG_INSTALLSCRIPT
./$TREETAG_INSTALLSCRIPT
◇
```
Fragment defined by 17g, 18abcde.
Fragment referenced in 7a.

Remove the tarballs:

⟨ *install the treetagger utility* 18e ⟩ ≡
```
rm $TREETAGSRC
rm $TREETAGSCRIPTS
rm $TREETAG_INSTALLSCRIPT
rm $DUTCHPARS_UTF_GZ
rm $DUTCH_TAGSET
rm $DUTCHPARS_2_GZ
◇
```
Fragment defined by 17g, 18abcde.
Fragment referenced in 7a.

3.3 Timbl and ticcutils

3.3.1 Module

Timbl and ticcutils are installed from their source-tarballs. The installation is not (yet?) completely reproducibe because it uses the currently available c-compiler. Installation involves:

1. Download the tarball in a temporary directory.
2. Unpack the tarball.
3. cd to the unpacked directory and perform `./configure`, `make` and `make install`. Note the argument that causes the files to be installed in the `usrlocal` subdirectory of the modules directory.

⟨ *install the ticcutils utility* 19a ⟩ ≡

```
URL=http://software.ticc.uvt.nl/ticcutils-0.7.tar.gz
TARB=ticcutils-0.7.tar.gz
DIR=ticcutils-0.7
```

⟨ *unpack ticcutils or timbl* 20a ⟩
◇

Fragment referenced in 7a.

⟨ *install the timbl utility* 19b ⟩ ≡

```
URL=http://software.ticc.uvt.nl/timbl-6.4.6.tar.gz
TARB=timbl-6.4.6.tar.gz
DIR=timbl-6.4.6
```

⟨ *unpack ticcutils or timbl* 20a ⟩
◇

Fragment referenced in 7a.

⟨ *unpack ticcutils or timbl* 20a ⟩ ≡
```
SUCCES=0
ticbeldir=‘mktemp -t -d tickbel.XXXXXX‘
cd $ticbeldir
wget $URL
SUCCES=$?
if
  [ $SUCCES -eq 0 ]
then
  tar -xzf $TARB
  SUCCES=$?
  rm -rf $TARB
fi
if
  [ $SUCCES -eq 0 ]
then
  cd $DIR
  ./configure --prefix=/mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa/modules/usrlocal
  make
  make install
fi
cd /mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa
rm -rf $ticbeldir
if
  [ $SUCCES -eq 0 ]
then
  ⟨ logmess (20b Installed $DIR ) 20e ⟩
else
  ⟨ logmess (20c NOT installed $DIR ) 20e ⟩
fi
◇
```
Fragment referenced in 19ab.

3.4 Logging

Write log messages to standard out if variable LOGLEVEL is equal to 1.

⟨ *variables of install-modules* 20d ⟩ ≡
```
LOGLEVEL=1
◇
```
Fragment referenced in 7a.

⟨ *logmess* 20e ⟩ ≡
```
if
 [ $LOGLEVEL -gt 0 ]
then
 echo @1
fi
◇
```
Fragment referenced in 5d, 6a, 8c, 20a, 21a.

3.5 Misc

Install a module from a tarball: The macro expects the following three variables to be present:

URL: The URL tfrom where the taball can be downloaded.
TARB: The name of the tarball.
DIR; Name of the directory for the module.

Arg 1: URL; Arg 2: tarball; Arg 3: directory.

```
⟨ install from tarball 21a ⟩ ≡
        SUCCES=0
        cd /mnt/kyoto/projecten/pipelines/dutch-nlp-modules-on-Lisa/modules
        ⟨ move module (21b $DIR ) 5b ⟩
        wget $URL
        SUCCES=$?
        if
          [ $SUCCES -eq 0 ]
        then
          tar -xzf $TARB
          SUCCES=$?
          rm -rf $TARB
        fi
        if
          [ $SUCCES -eq 0 ]
        then
          ⟨ logmess (21c Installed $DIR ) 20e ⟩
          ⟨ remove old module (21d $DIR ) 5c ⟩
        else
          ⟨ re-instate old module (21e $DIR ) 5d ⟩
        fi
        ◇
```

Fragment never referenced.

A How to read and translate this document

This document is an example of *literate programming* [?]. It contains the code of all sorts of scripts and programs, combined with explaining texts. In this document the literate programming tool `nuweb` is used, that is currently available from Sourceforge (URL:`nuweb.sourceforge.net`). The advantages of Nuweb are, that it can be used for every programming language and scripting language, that it can contain multiple program sources and that it is very simple.

A.1 Read this document

The document contains *code scraps* that are collected into output files. An output file (e.g. `output.fil`) shows up in the text as follows:

```
"output.fil" 4a ≡
        # output.fil
        < a macro 4b >
        < another macro 4c >
        ◇
```

The above construction contains text for the file. It is labelled with a code (in this case 4a) The constructions between the < and > brackets are macro's, placeholders for texts that can be found in other places of the document. The test for a macro is found in constructions that look like:

< a macro 4b > ≡

```
    This is a scrap of code inside the macro.
    It is concatenated with other scraps inside the
    macro. The concatenated scraps replace
    the invocation of the macro.
```

```
Macro defined by 4b, 87e
Macro referenced in 4a
```

Macro’s can be defined on different places. They can contain other macroÂ´s.

< a scrap 87e > ≡

```
    This is another scrap in the macro. It is
    concatenated to the text of scrap 4b.
    This scrap contains another macro:
    < another macro 45b >
```

```
Macro defined by 4b, 87e
Macro referenced in 4a
```

A.2 Process the document

The raw document is named `a_dutch-nlp-modules-on-Lisa.w`. Figure 1 shows pathways to

Figure 1: Translation of the raw code of this document into printable/viewable documents and into program sources. The figure shows the pathways and the main files involved.

translate it into printable/viewable documents and to extract the program sources. Table 4 lists

Tool	Source	Description
gawk	`www.gnu.org/software/gawk/`	text-processing scripting language
M4	`www.gnu.org/software/m4/`	Gnu macro processor
nuweb	`nuweb.sourceforge.net`	Literate programming tool
tex	`www.ctan.org`	Typesetting system
tex4ht	`www.ctan.org`	Convert TEX documents into `xml`/`html`

Table 4: Tools to translate this document into readable code and to extract the program sources

the tools that are needed for a translation. Most of the tools (except Nuweb) are available on a well-equipped Linux system.

A.3 Translate and run

This chapter assembles the Makefile for this project.

`"Makefile"` 22≡

⟨ *default target* 23a ⟩

⟨ *parameters in Makefile* 23c, … ⟩

⟨ *impliciete make regels* 26a, … ⟩
⟨ *expliciete make regels* 23e, … ⟩
⟨ *make targets* 27a, … ⟩
◇

The default target of make is `all`.

⟨ *default target* 23a ⟩ ≡
```
all : ⟨ all targets 23b ⟩
.PHONY : all

◇
```
Fragment referenced in 22.
Defines: `all` Never used, `PHONY` 26b.

One of the targets is certainly the PDF version of this document.

⟨ *all targets* 23b ⟩ ≡
```
dutch-nlp-modules-on-Lisa.pdf◇
```
Fragment referenced in 23a.
Uses: `pdf` 27a.

We use many suffixes that were not known by the C-programmers who constructed the `make` utility. Add these suffixes to the list.

⟨ *parameters in Makefile* 23c ⟩ ≡
```
.SUFFIXES: .pdf .w .tex .html .aux .log .php

◇
```
Fragment defined by 23cd, 25ab, 27d, 30b, 33a.
Fragment referenced in 22.
Defines: `SUFFIXES` Never used.
Uses: `pdf` 27a.

A.4 Get Nuweb

An annoying problem is, that this program uses nuweb, a utility that is seldom installed on a computer. Therefore, we are going to install that first if it is not present. Unfortunately, nuweb is hosted on sourceforge and it is difficult to achieve automatic downloading from that repository. Therefore I copied one of the versions on a location from where it can be downloaded with a script.

⟨ *parameters in Makefile* 23d ⟩ ≡
```
NUWEB=../bin/nuweb
◇
```
Fragment defined by 23cd, 25ab, 27d, 30b, 33a.
Fragment referenced in 22.
Defines: `NUWEB` 23e, 28abc, 29b, 31c, 32c, 33c.
Uses: `nuweb` 29b.

⟨ *expliciete make regels* 23e ⟩ ≡
```
$(NUWEB): ../nuweb-1.58
        cd ../nuweb-1.58 && make nuweb
        cp ../nuweb-1.58/nuweb $(NUWEB)

◇
```
Fragment defined by 23e, 24abc, 26b, 28a, 30de, 31ab.
Fragment referenced in 22.
Uses: `NUWEB` 23d, `nuweb` 29b.

⟨ *expliciete make regels* 24a ⟩ ≡
```
../nuweb-1.58:
        cd .. && wget http://kyoto.let.vu.nl/~huygen/nuweb-1.58.tgz
        cd .. &&  tar -xzf nuweb-1.58.tgz

◇
```
Fragment defined by 23e, 24abc, 26b, 28a, 30de, 31ab.
Fragment referenced in 22.
Uses: `nuweb` 29b.

A.5 Pre-processing

To make usable things from the raw input `a_dutch-nlp-modules-on-Lisa.w`, do the following:

1. Process `$` characters.
2. Run the m4 pre-processor.
3. Run nuweb.

This results in a LaTeX file, that can be converted into a PDF or a HTML document, and in the program sources and scripts.

A.5.1 Process 'dollar' characters

Many "intelligent" TeX editors (e.g. the auctex utility of Emacs) handle `$` characters as special, to switch into mathematics mode. This is irritating in program texts, that often contain `$` characters as well. Therefore, we make a stub, that translates the two-character sequence `\$` into the single `$` character.

⟨ *expliciete make regels* 24b ⟩ ≡
```
m4_dutch-nlp-modules-on-Lisa.w : a_dutch-nlp-modules-on-Lisa.w
        gawk '{if(match($$0, "@%")) {printf("%s", substr($$0,1,RSTART-1))} else print}' a_dutch-nlp-m
          | gawk '{gsub(/[\\][\$$]/, "$$");print}'  > m4_dutch-nlp-modules-on-Lisa.w

◇
```
Fragment defined by 23e, 24abc, 26b, 28a, 30de, 31ab.
Fragment referenced in 22.
Uses: `print` 27a.

A.5.2 Run the M4 pre-processor

⟨ *expliciete make regels* 24c ⟩ ≡
```
dutch-nlp-modules-on-Lisa.w : m4_dutch-nlp-modules-on-Lisa.w inst.m4
        m4 -P m4_dutch-nlp-modules-on-Lisa.w > dutch-nlp-modules-on-Lisa.w

◇
```
Fragment defined by 23e, 24abc, 26b, 28a, 30de, 31ab.
Fragment referenced in 22.

A.6 Typeset this document

Enable the following:

1. Create a PDF document.
2. Print the typeset document.

3. View the typeset document with a viewer.
4. Create a HTMLdocument.

In the three items, a typeset PDF document is required or it is the requirement itself.

A.6.1 Figures

This document contains figures that have been made by `xfig`. Post-process the figures to enable inclusion in this document.

The list of figures to be included:

⟨ *parameters in Makefile* 25a ⟩ ≡

```
FIGFILES=fileschema
```

◇

Fragment defined by 23cd, 25ab, 27d, 30b, 33a.
Fragment referenced in 22.
Defines: `FIGFILES` 25b, 30b.

We use the package `figlatex` to include the pictures. This package expects two files with extensions `.pdftex` and `.pdftex_t` for `pdflatex` and two files with extensions `.pstex` and `.pstex_t` for the `latex/dvips` combination. Probably tex4ht uses the latter two formats too.

Make lists of the graphical files that have to be present for latex/pdflatex:

⟨ *parameters in Makefile* 25b ⟩ ≡

```
FIGFILENAMES=$(foreach fil,$(FIGFILES), $(fil).fig)
PDFT_NAMES=$(foreach fil,$(FIGFILES), $(fil).pdftex_t)
PDF_FIG_NAMES=$(foreach fil,$(FIGFILES), $(fil).pdftex)
PST_NAMES=$(foreach fil,$(FIGFILES), $(fil).pstex_t)
PS_FIG_NAMES=$(foreach fil,$(FIGFILES), $(fil).pstex)
```

◇

Fragment defined by 23cd, 25ab, 27d, 30b, 33a.
Fragment referenced in 22.
Defines: `FIGFILENAMES` Never used, `PDFT_NAMES` 27b, `PDF_FIG_NAMES` 27b, `PST_NAMES` Never used, `PS_FIG_NAMES` Never used.
Uses: `FIGFILES` 25a.

Create the graph files with program `fig2dev`:

⟨ *impliciete make regels* 26a ⟩ ≡
```
%.eps: %.fig
        fig2dev -L eps $< > $@

%.pstex: %.fig
        fig2dev -L pstex $< > $@

.PRECIOUS : %.pstex
%.pstex_t: %.fig %.pstex
        fig2dev -L pstex_t -p $*.pstex $< > $@

%.pdftex: %.fig
        fig2dev -L pdftex $< > $@

.PRECIOUS : %.pdftex
%.pdftex_t: %.fig %.pstex
        fig2dev -L pdftex_t -p $*.pdftex $< > $@

◇
```
Fragment defined by 26a, 27b, 30c.
Fragment referenced in 22.
Defines: `fig2dev` Never used.

A.6.2 Bibliography

To keep this document portable, create a portable bibliography file. It works as follows: This document refers in the |bibliography| statement to the local bib-file `dutch-nlp-modules-on-Lisa.bib`. To create this file, copy the auxiliary file to another file `auxfil.aux`, but replace the argument of the command `\bibdata{dutch-nlp-modules-on-Lisa}` to the names of the bibliography files that contain the actual references (they should exist on the computer on which you try this). This procedure should only be performed on the computer of the author. Therefore, it is dependent of a binary file on his computer.

⟨ *expliciete make regels* 26b ⟩ ≡
```
bibfile : dutch-nlp-modules-on-Lisa.aux /home/paul/bin/mkportbib
        /home/paul/bin/mkportbib dutch-nlp-modules-on-Lisa litprog

.PHONY : bibfile
◇
```
Fragment defined by 23e, 24abc, 26b, 28a, 30de, 31ab.
Fragment referenced in 22.
Uses: `PHONY` 23a.

A.6.3 Create a printable/viewable document

Make a PDF document for printing and viewing.

⟨ *make targets* 27a ⟩ ≡
```
pdf : dutch-nlp-modules-on-Lisa.pdf

print : dutch-nlp-modules-on-Lisa.pdf
        lpr dutch-nlp-modules-on-Lisa.pdf

view : dutch-nlp-modules-on-Lisa.pdf
        evince dutch-nlp-modules-on-Lisa.pdf

◇
```
Fragment defined by 27a, 30a, 33bc.
Fragment referenced in 22.
Defines: `pdf` 23bc, 27b, `print` 10a, 11a, 15a, 24b, `view` Never used.

Create the PDF document. This may involve multiple runs of nuweb, the LaTeX processor and the bibTeX processor, and depends on the state of the `aux` file that the LaTeX processor creates as a by-product. Therefore, this is performed in a separate script, `w2pdf`.

The w2pdf script The three processors nuweb, LaTeX and bibTeX are intertwined. LaTeX and bibTeX create parameters or change the value of parameters, and write them in an auxiliary file. The other processors may need those values to produce the correct output. The LaTeX processor may even need the parameters in a second run. Therefore, consider the creation of the (PDF) document finished when none of the processors causes the auxiliary file to change. This is performed by a shell script `w2pdf`.

Note, that in the following `make` construct, the implicit rule `.w.pdf` is not used. It turned out, that make did not calculate the dependencies correctly when I did use this rule.

⟨ *impliciete make regels* 27b ⟩ ≡
```
%.pdf : %.w $(W2PDF)  $(PDF_FIG_NAMES) $(PDFT_NAMES)
        chmod 775 $(W2PDF)
        $(W2PDF) $*

◇
```
Fragment defined by 26a, 27b, 30c.
Fragment referenced in 22.
Uses: `pdf` 27a, `PDFT_NAMES` 25b, `PDF_FIG_NAMES` 25b.

The following is an ugly fix of an unsolved problem. Currently I develop this thing, while it resides on a remote computer that is connected via the `sshfs` filesystem. On my home computer I cannot run executables on this system, but on my work-computer I can. Therefore, place the following script on a local directory.

⟨ *directories to create* 27c ⟩ ≡
```
../nuweb/bin ◇
```
Fragment defined by 4abcdef, 9h, 27c.
Fragment referenced in 33b.
Uses: `nuweb` 29b.

⟨ *parameters in Makefile* 27d ⟩ ≡
```
W2PDF=../nuweb/bin/w2pdf
◇
```
Fragment defined by 23cd, 25ab, 27d, 30b, 33a.
Fragment referenced in 22.
Uses: `nuweb` 29b.

⟨ *expliciete make regels* 28a ⟩ ≡
```
$(W2PDF) : dutch-nlp-modules-on-Lisa.w $(NUWEB)
        $(NUWEB) dutch-nlp-modules-on-Lisa.w
◇
```
Fragment defined by 23e, 24abc, 26b, 28a, 30de, 31ab.
Fragment referenced in 22.
Uses: `NUWEB` 23d.

`"../nuweb/bin/w2pdf"` 28b≡
```
#!/bin/bash
# w2pdf -- compile a nuweb file
# usage: w2pdf [filename]
# 20141110 at 1041h: Generated by nuweb from a_dutch-nlp-modules-on-Lisa.w
NUWEB=/usr/local/bin/nuweb

LATEXCOMPILER=pdflatex
⟨ filenames in nuweb compile script 28d ⟩
⟨ compile nuweb 28c ⟩

◇
```
Uses: `NUWEB` 23d, `nuweb` 29b.

The script retains a copy of the latest version of the auxiliary file. Then it runs the four processors nuweb, LaTeX, MakeIndex and bibTeX, until they do not change the auxiliary file or the index.

⟨ *compile nuweb* 28c ⟩ ≡
```
NUWEB=/usr/local/bin/nuweb

⟨ run the processors until the aux file remains unchanged 29c ⟩
⟨ remove the copy of the aux file 29a ⟩
◇
```
Fragment referenced in 28b.
Uses: `NUWEB` 23d, `nuweb` 29b.

The user provides the name of the nuweb file as argument. Strip the extension (e.g. `.w`) from the filename and create the names of the LaTeX file (ends with `.tex`), the auxiliary file (ends with `.aux`) and the copy of the auxiliary file (add `old.` as a prefix to the auxiliary filename).

⟨ *filenames in nuweb compile script* 28d ⟩ ≡
```
nufil=$1
trunk=${1%%.*}
texfil=${trunk}.tex
auxfil=${trunk}.aux
oldaux=old.${trunk}.aux
indexfil=${trunk}.idx
oldindexfil=old.${trunk}.idx
◇
```
Fragment referenced in 28b.
Defines: `auxfil` 29c, 32ab, `indexfil` 29c, 32a, `nufil` 29b, 32ac, `oldaux` 29ac, 32ab, `oldindexfil` 29c, 32a, `texfil` 29b, 32ac, `trunk` 29b, 32acd.

Remove the old copy if it is no longer needed.

⟨ *remove the copy of the aux file* 29a ⟩ ≡
```
rm $oldaux
◇
```
Fragment referenced in 28c, 31d.
Uses: `oldaux` 28d, 32a.

Run the three processors. Do not use the option `-o` (to suppres generation of program sources) for nuweb, because `w2pdf` must be kept up to date as well.

⟨ *run the three processors* 29b ⟩ ≡
```
$NUWEB $nufil
$LATEXCOMPILER $texfil
makeindex $trunk
bibtex $trunk
◇
```
Fragment referenced in 29c.
Defines: `bibtex` 32cd, `makeindex` 32cd, `nuweb` 23de, 24a, 27cd, 28bc, 30a, 31bc.
Uses: `nufil` 28d, 32a, `NUWEB` 23d, `texfil` 28d, 32a, `trunk` 28d, 32a.

Repeat to copy the auxiliary file and the index file and run the processors until the auxiliary file and the index file are equal to their copies. However, since I have not yet been able to test the `aux` file and the `idx` in the same test statement, currently only the `aux` file is tested.

It turns out, that sometimes a strange loop occurs in which the `aux` file will keep to change. Therefore, with a counter we prevent the loop to occur more than 10 times.

⟨ *run the processors until the aux file remains unchanged* 29c ⟩ ≡
```
LOOPCOUNTER=0
while
  ! cmp -s $auxfil $oldaux
do
  if [ -e $auxfil ]
  then
   cp $auxfil $oldaux
  fi
  if [ -e $indexfil ]
  then
   cp $indexfil $oldindexfil
  fi
  ⟨ run the three processors 29b ⟩
  if [ $LOOPCOUNTER -ge 10 ]
  then
    cp $auxfil $oldaux
  fi;
done
◇
```
Fragment referenced in 28c.
Uses: `auxfil` 28d, 32a, `indexfil` 28d, `oldaux` 28d, 32a, `oldindexfil` 28d.

A.6.4 Create HTML files

HTML is easier to read on-line than a PDF document that was made for printing. We use `tex4ht` to generate HTML code. An advantage of this system is, that we can include figures in the same way as we do for `pdflatex`.

Nuweb creates a LaTeX file that is suitable for `latex2html` if the source file has `.hw` as suffix instead of `.w`. However, this feature is not compatible with tex4ht.

Make html file:

⟨ *make targets* 30a ⟩ ≡
```
html : ../nuweb/html/dutch-nlp-modules-on-Lisa.html

◇
```
Fragment defined by 27a, 30a, 33bc.
Fragment referenced in 22.
Uses: `nuweb` 29b.

The HTML file depends on its source file and the graphics files.

Make lists of the graphics files and copy them.

⟨ *parameters in Makefile* 30b ⟩ ≡
```
HTML_PS_FIG_NAMES=$(foreach fil,$(FIGFILES), m4_htmldocdir/$(fil).pstex)
HTML_PST_NAMES=$(foreach fil,$(FIGFILES), m4_htmldocdir/$(fil).pstex_t)
◇
```
Fragment defined by 23cd, 25ab, 27d, 30b, 33a.
Fragment referenced in 22.
Uses: `FIGFILES` 25a.

⟨ *impliciete make regels* 30c ⟩ ≡
```
m4_htmldocdir/%.pstex : %.pstex
        cp  $< $@

m4_htmldocdir/%.pstex_t : %.pstex_t
        cp  $< $@

◇
```
Fragment defined by 26a, 27b, 30c.
Fragment referenced in 22.

Copy the nuweb file into the html directory.

⟨ *expliciete make regels* 30d ⟩ ≡
```
m4_htmlsource : dutch-nlp-modules-on-Lisa.w
        cp  dutch-nlp-modules-on-Lisa.w m4_htmlsource

◇
```
Fragment defined by 23e, 24abc, 26b, 28a, 30de, 31ab.
Fragment referenced in 22.

We also need a file with the same name as the documentstyle and suffix `.4ht`. Just copy the file `report.4ht` from the tex4ht distribution. Currently this seems to work.

⟨ *expliciete make regels* 30e ⟩ ≡
```
m4_4htfildest : m4_4htfilsource
        cp m4_4htfilsource m4_4htfildest

◇
```
Fragment defined by 23e, 24abc, 26b, 28a, 30de, 31ab.
Fragment referenced in 22.

Copy the bibliography.

⟨ *expliciete make regels* 31a ⟩ ≡
```
m4_htmlbibfil : m4_nuwebdir/dutch-nlp-modules-on-Lisa.bib
        cp m4_nuwebdir/dutch-nlp-modules-on-Lisa.bib m4_htmlbibfil


◇
```
Fragment defined by 23e, 24abc, 26b, 28a, 30de, 31ab.
Fragment referenced in 22.

Make a dvi file with `w2html` and then run `htlatex`.

⟨ *expliciete make regels* 31b ⟩ ≡
```

../nuweb/html/dutch-nlp-modules-on-Lisa.html : m4_htmlsource m4_4htfildest $(HTML_PS_FIG_NAMES) $(HTM
        cp w2html ../bin
        cd ../bin && chmod 775 w2html
        cd m4_htmldocdir && ../bin/w2html dutch-nlp-modules-on-Lisa.w


◇
```
Fragment defined by 23e, 24abc, 26b, 28a, 30de, 31ab.
Fragment referenced in 22.
Uses: `nuweb` 29b.

Create a script that performs the translation.

`"w2html"` 31c≡
```
#!/bin/bash
# w2html -- make a html file from a nuweb file
# usage: w2html [filename]
#  [filename]: Name of the nuweb source file.
`#' m4_header
echo "translate " $1 >w2html.log
NUWEB=/usr/local/bin/nuweb

⟨ filenames in w2html 32a ⟩

⟨ perform the task of w2html 31d ⟩


◇
```
Uses: `NUWEB` 23d, `nuweb` 29b.

The script is very much like the `w2pdf` script, but at this moment I have still difficulties to compile the source smoothly into HTML and that is why I make a separate file and do not recycle parts from the other file. However, the file works similar.

⟨ *perform the task of w2html* 31d ⟩ ≡
```
⟨ run the html processors until the aux file remains unchanged 32b ⟩
⟨ remove the copy of the aux file 29a ⟩
◇
```
Fragment referenced in 31c.

The user provides the name of the nuweb file as argument. Strip the extension (e.g. `.w`) from the filename and create the names of the LaTeX file (ends with `.tex`), the auxiliary file (ends with `.aux`) and the copy of the auxiliary file (add `old.` as a prefix to the auxiliary filename).

⟨ *filenames in w2html* 32a ⟩ ≡

```
nufil=$1
trunk=${1%%.*}
texfil=${trunk}.tex
auxfil=${trunk}.aux
oldaux=old.${trunk}.aux
indexfil=${trunk}.idx
oldindexfil=old.${trunk}.idx
◇
```

Fragment referenced in 31c.
Defines: `auxfil` 28d, 29c, 32b, `nufil` 28d, 29b, 32c, `oldaux` 28d, 29ac, 32b, `texfil` 28d, 29b, 32c, `trunk` 28d, 29b, 32cd.
Uses: `indexfil` 28d, `oldindexfil` 28d.

⟨ *run the html processors until the aux file remains unchanged* 32b ⟩ ≡

```
while
  ! cmp -s $auxfil $oldaux
do
  if [ -e $auxfil ]
  then
   cp $auxfil $oldaux
  fi
  ⟨ run the html processors 32c ⟩
done
⟨ run tex4ht 32d ⟩

◇
```

Fragment referenced in 31d.
Uses: `auxfil` 28d, 32a, `oldaux` 28d, 32a.

To work for HTML, nuweb *must* be run with the `-n` option, because there are no page numbers.

⟨ *run the html processors* 32c ⟩ ≡

```
$NUWEB -o -n $nufil
latex $texfil
makeindex $trunk
bibtex $trunk
htlatex $trunk
◇
```

Fragment referenced in 32b.
Uses: `bibtex` 29b, `makeindex` 29b, `nufil` 28d, 32a, `NUWEB` 23d, `texfil` 28d, 32a, `trunk` 28d, 32a.

When the compilation has been satisfied, run makeindex in a special way, run bibtex again (I don't know why this is necessary) and then run htlatex another time.

⟨ *run tex4ht* 32d ⟩ ≡

```
tex '\def\filename{{dutch-nlp-modules-on-Lisa}{idx}{4dx}{ind}} \input idxmake.4ht'
makeindex -o $trunk.ind $trunk.4dx
bibtex $trunk
htlatex $trunk
◇
```

Fragment referenced in 32b.
Uses: `bibtex` 29b, `makeindex` 29b, `trunk` 28d, 32a.

create the program sources Run nuweb, but suppress the creation of the LaTeX documentation. Nuweb creates only sources that do not yet exist or that have been modified. Therefore make does not have to check this. However, "make" has to create the directories for the sources if they do not yet exist. So, let's create the directories first.

⟨ *parameters in Makefile* 33a ⟩ ≡
```
        MKDIR = mkdir -p

        ◇
```
Fragment defined by 23cd, 25ab, 27d, 30b, 33a.
Fragment referenced in 22.
Defines: `MKDIR` 33b.

⟨ *make targets* 33b ⟩ ≡
```
        DIRS = ⟨ directories to create 4a, ... ⟩

        $(DIRS) :
                $(MKDIR) $@

        ◇
```
Fragment defined by 27a, 30a, 33bc.
Fragment referenced in 22.
Defines: `DIRS` 33c.
Uses: `MKDIR` 33a.

⟨ *make targets* 33c ⟩ ≡
```
        sources : dutch-nlp-modules-on-Lisa.w $(DIRS) $(NUWEB)
                $(NUWEB) dutch-nlp-modules-on-Lisa.w
                ⟨ make scripts executable 7b, ... ⟩

        ◇
```
Fragment defined by 27a, 30a, 33bc.
Fragment referenced in 22.
Uses: `DIRS` 33b, `NUWEB` 23d.

B References

B.1 Literature

B.2 URL's

Nuweb: `nuweb.sourceforge.net`
Apache Velocity: `m4_velocityURL`
Velocitytools: `m4_velocitytoolsURL`
Parameterparser tool: `m4_parameterparserdocURL`
Cookietool: `m4_cookietooldocURL`
VelocityView: `m4_velocityviewURL`
VelocityLayoutServlet: `m4_velocitylayoutservletURL`
Jetty: `m4_jettycodehausURL`
UserBase javadoc: `m4_userbasejavadocURL`
VU corpus Management development site: `http://code.google.com/p/vucom`

C Indexes

C.1 Filenames

C.2 Macro's

C.3 Variables