Before we start...

- Start downloading Moses:

```
wget http://ufal.mff.cuni.cz/~tamchyna/mosesgiza.64bit.tar.gz
```

- Start downloading our “playground” for SMT:

```
wget http://ufal.mff.cuni.cz/eman/download/playground.tar
```

- Slides can be downloaded here:

```
http://ufal.mff.cuni.cz/~tamchyna/mtm14.slides.pdf
```

Experimenting in MT: Moses Toolkit and Eman

Ondřej Bojar, **Aleš Tamchyna**
Institute of Formal and Applied Linguistics
Faculty of Mathematics and Physics
Charles University, Prague

Mon Sept 8, 2014

Outline

- Quick overview of Moses.
- Bird's eye view of (phrase-based) MT.
    - With pointers to Moses repository.
- Experiment management.
    - Motivation.
    - Overview of Eman.
- Run your own experiments.
    - Introduce Eman's features through building a baseline Czech→English MT system.
    - Inspect the pipeline and created models.
    - Try some techniques to improve over the baseline.

Moses Toolkit

- Comprehensive open-source toolkit for SMT
- Core: phrase-based and syntactic decoder

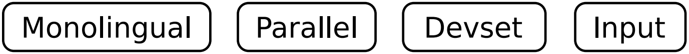
Moses Toolkit

- Comprehensive open-source toolkit for SMT
- Core: phrase-based and syntactic decoder
- Includes many related tools:
    - Data pre-processing:
      cleaning, sentence splitting, tokenization, ...
    - Building models for translation:
      create phrase/rule tables from word-aligned data,
      train language models with KenLM
    - Tuning translation systems (MERT and others)

Moses Toolkit

- Comprehensive open-source toolkit for SMT
- Core: phrase-based and syntactic decoder
- Includes many related tools:
    - Data pre-processing:
      cleaning, sentence splitting, tokenization, ...
    - Building models for translation:
      create phrase/rule tables from word-aligned data,
      train language models with KenLM
    - Tuning translation systems (MERT and others)
- You still need a tool for word alignment:
    - GIZA++, fast_align, ...
- Bundled with its own experiment manager EMS
    - We will use a different one

Bird's Eye View of Phrase-Based MT



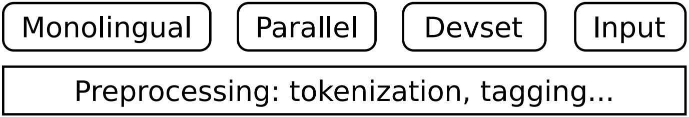
Monolingual

Parallel

Devset

Input

Bird's Eye View of Phrase-Based MT



Monolingual

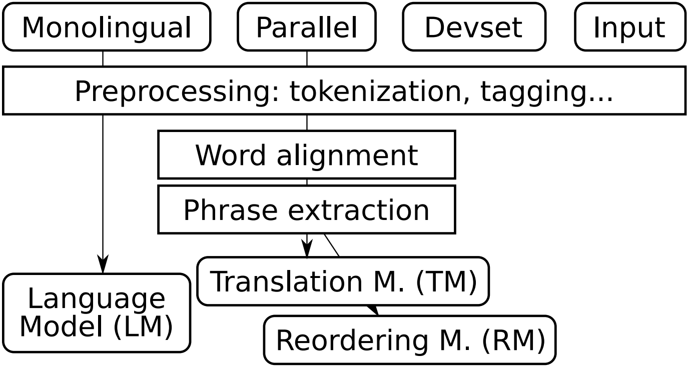
Parallel

Devset

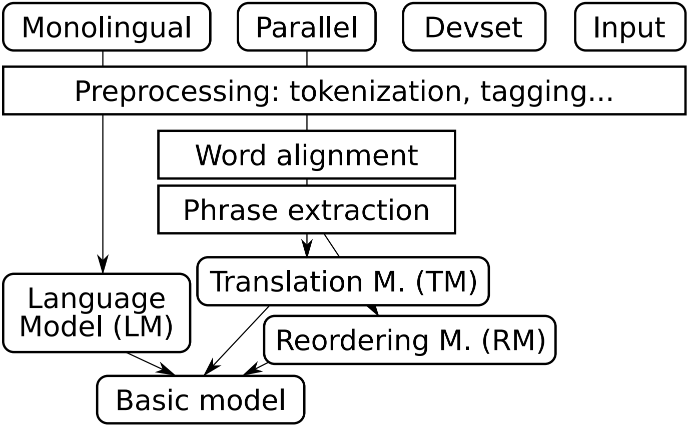
Input

Preprocessing: tokenization, tagging...

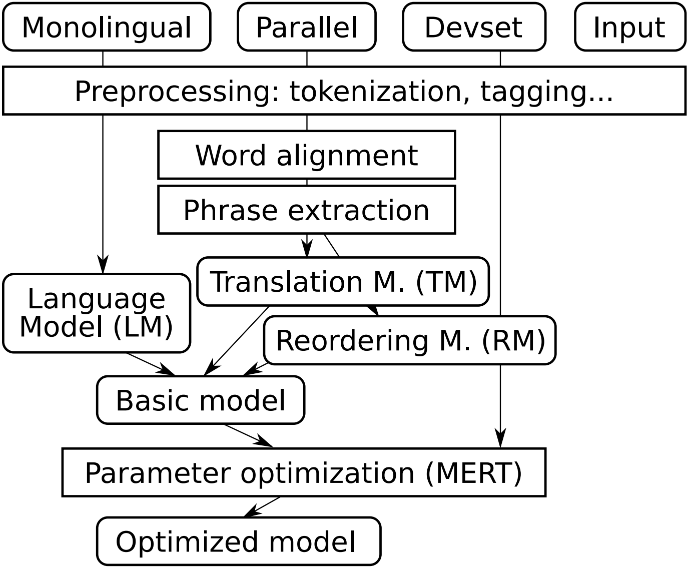
Bird's Eye View of Phrase-Based MT



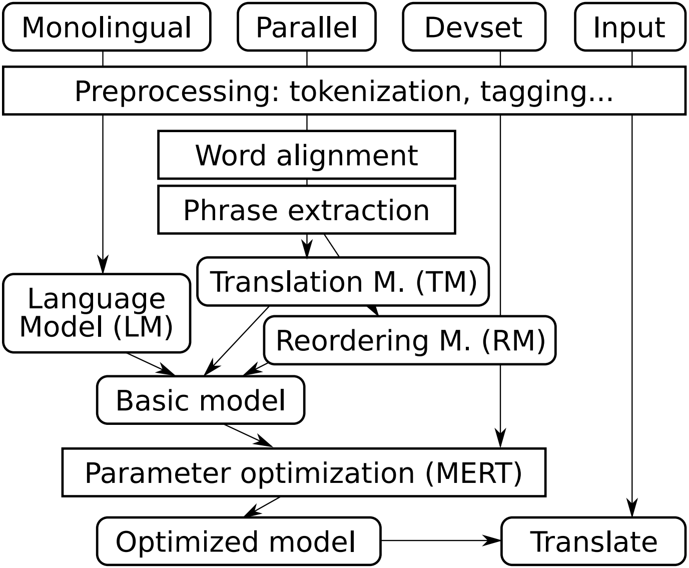
Bird's Eye View of Phrase-Based MT



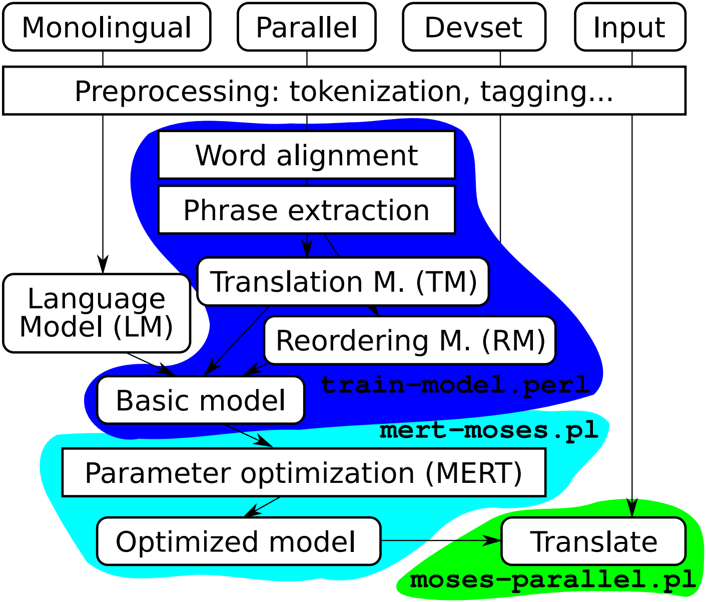
Bird’s Eye View of Phrase-Based MT



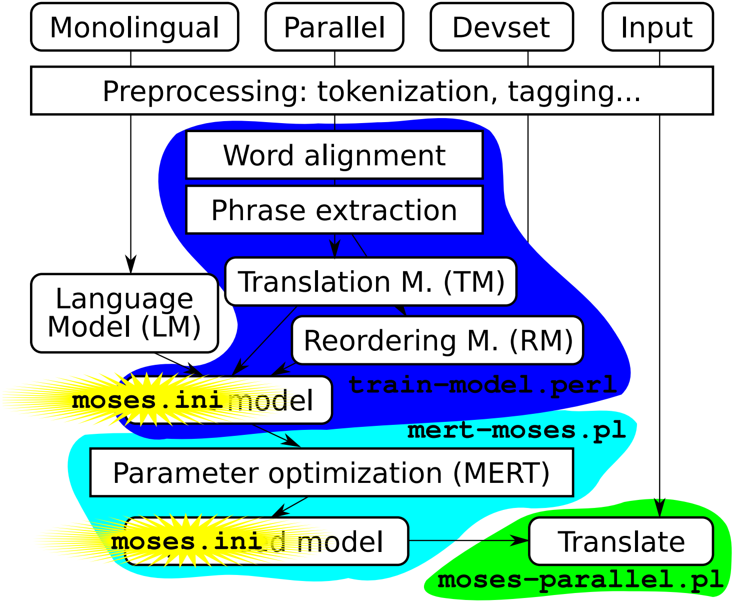
Bird's Eye View of Phrase-Based MT



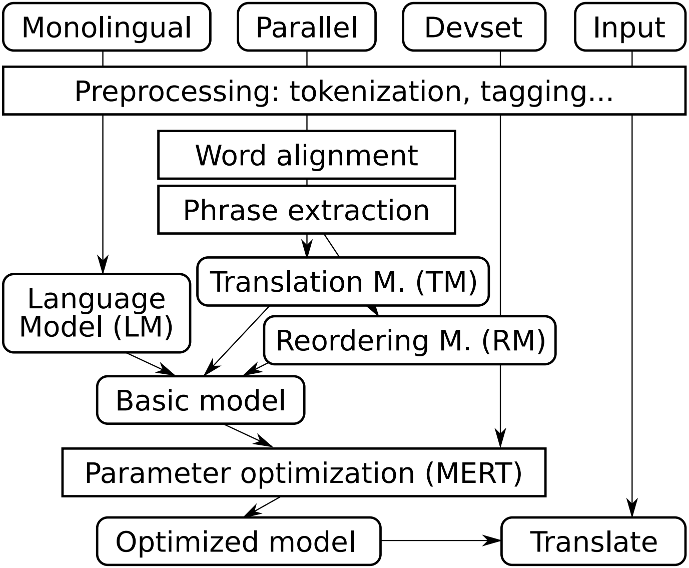
Bird's Eye View of Phrase-Based MT



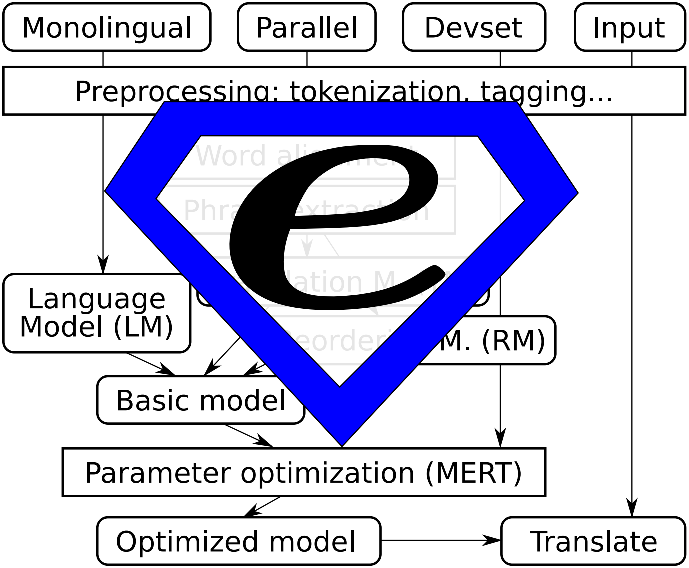
Bird's Eye View of Phrase-Based MT



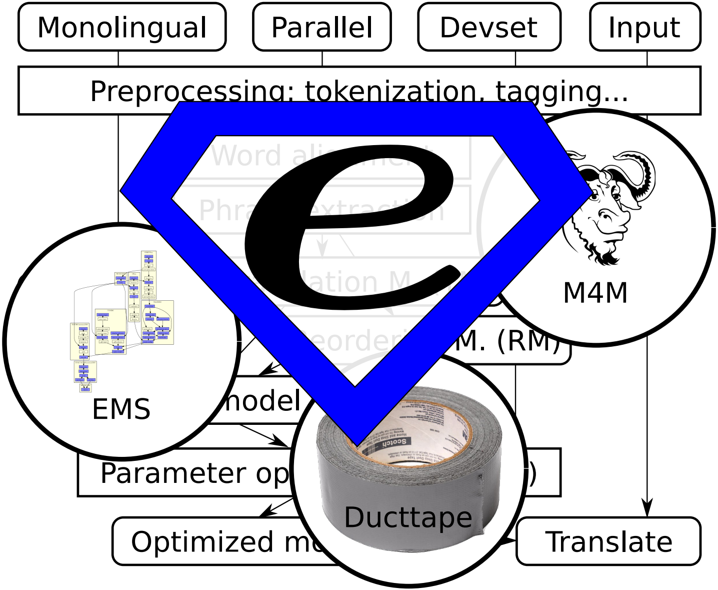
Now, This Complex World...



…Has to Be Ruled by Someone



...Has to Be Ruled by Someone



Monolingual

Parallel

Devset

Input

Preprocessing: tokenization, tagging...

Word ali men

Phra extraction

lation M

eorderi M. (RM)

nodel

Parameter op)

Optimized m

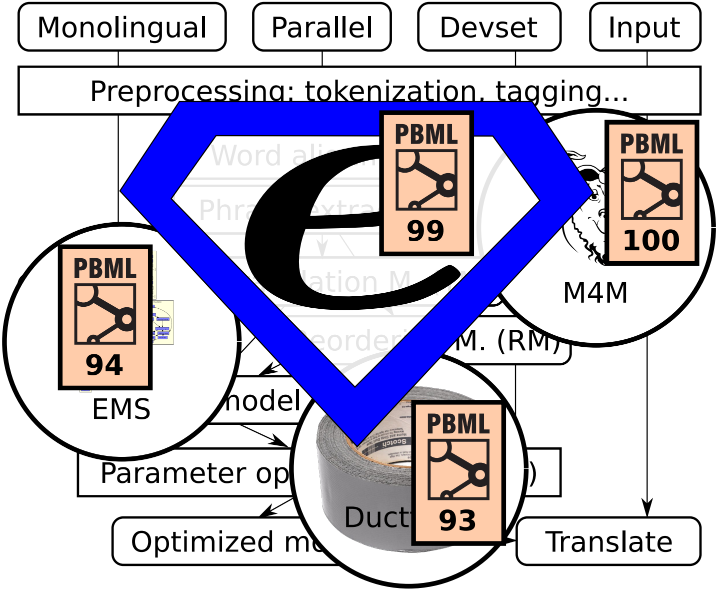
Translate

EMS

M4M

Ducttape

…Has to Be Ruled by Someone



Monolingual

Parallel

Devset

Input

Preprocessing: tokenization, tagging...

Word ali

Phra extra

lation M

eorderi M. (RM)

model

Parameter op)

Optimized m

Translate

Why Use an Experiment Manager?

- Automatic tracking of system configurations, versions of software,. . .
  $\Rightarrow$ reproducibility of results

Why Use an Experiment Manager?

- Automatic tracking of system configurations, versions of software,...
  $\Rightarrow$ reproducibility of results
- Efficiency/convenience
  - (MT) experiments are pipelines of complex components
    $\Rightarrow$ hide implementation details, provide a unified abstraction
  - easily run many experiments in parallel

Why Use an Experiment Manager?

- Automatic tracking of system configurations, versions of software,. . .
  $\Rightarrow$ reproducibility of results
- Efficiency/convenience
  - (MT) experiments are pipelines of complex components
    $\Rightarrow$ hide implementation details, provide a unified abstraction
  - easily run many experiments in parallel
- Re-use of intermediate files
  - different experiments may share e.g. the same language model

Features of Eman

- Console-based ⇒ easily scriptable (e.g. in bash).
- Versatile: "seeds" are up to the user, any language.

- Support for the manual search through the space of experiment configurations.
- Support for finding and marking ("tagging") steps or experiments of interest.
- Support for organizing the results in 2D tables.

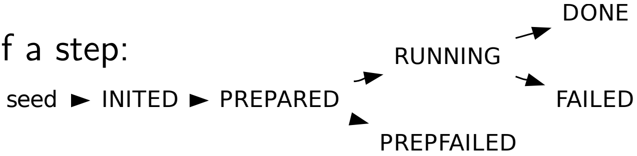
- Integrated with SGE
  ⇒ easy to run on common academic clusters.

eman --man will tell you some details.
http://ufal.mff.cuni.cz/eman/ has more.

Eman's View

- Experiments consist of processing STEPS.
- Steps are:
  - of a given type, e.g. **align**, **tm**, **lm**, **mert**,
  - defined by immutable variables, e.g. **ALISYM=gdfa**,
  - all located in one directory, the "**playground**",
  - timestamped unique directories, e.g. **s.mert.a123.20120215-1632**
  - self-contained in the dir as much as reasonable.
  - dependent on other steps, e.g. first **align**, then build **tm**, then **mert**.

Lifetime of a step:



Why INITED→PREPARED→RUNNING?

The call to **eman init** *seed*:

- Should be quick, it is used interactively.
- Should only check and set vars, "turn a blank directory into a valid eman step".

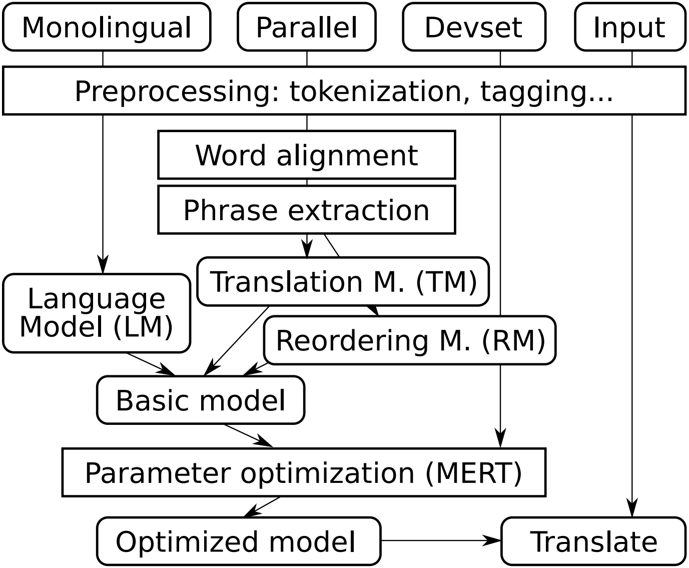
The call to **eman prepare** *s.step.123.20120215*:

- May check for various input files.
  - Less useful with heavy experiments where even corpus preparation needs cluster.
- Has to produce **eman.command**.
  ⇒ A chance to check it: are all file paths correct etc.?

The call to **eman start** *s.step.123.20120215*:

- Sends the job to the cluster.

Our Eman Seeds for MT



Monolingual

Parallel

Devset

Input

Preprocessing: tokenization, tagging...

Word alignment

Phrase extraction

Translation M. (TM)

Language Model (LM)

Reordering M. (RM)

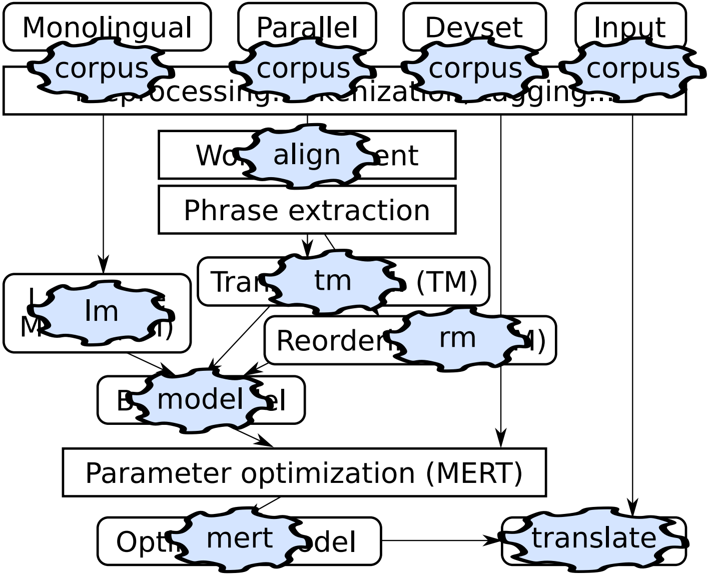
Basic model

Parameter optimization (MERT)

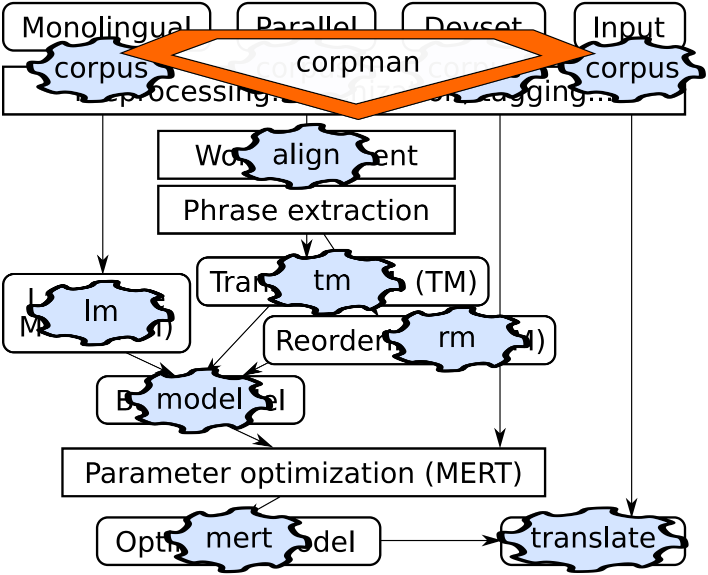
Optimized model

Translate

Our Eman Seeds for MT



Our Eman Seeds for MT



Eman's Bells and Whistles

Experiment management:

- **ls**, **vars**, **stat** for simple listing,
- **select** for finding steps,
- **traceback** for full info on experiments,
- **redo** failed experiments,
- **clone** individual steps as well as whole experiments.

Meta-information on steps:

- **status**,
- **tag**s, autotags,
- **collect**ing results,
- **tabulate** for putting results into 2D tables.

Whole Experiment = **eman traceback**

eman traceback s.evaluator.8102edfc.20120207-1611

```
+- s.evaluator.8102edfc.20120207-1611
|  +- s.mosesgiza.b6073a00.20120202-0037
|  +- s.translate.b17f203d.20120207-1604
|  |  +- s.mert.272f2f67.20120207-0013
|  |  |  +- s.model.3e28def7.20120207-0013
|  |  |  |  +- s.lm.608df574.20120207-0004
|  |  |  |  |  +- s.srilm.117f0cfe.20120202-0037
|  |  |  |  +- s.mosesgiza.b6073a00.20120202-0037
|  |  |  |  +- s.tm.527c9342.20120207-0012
|  |  |  |  |  +- s.align.dec45f74.20120206-0111
|  |  |  |  |  |  +- s.mosesgiza.b6073a00.20120202-0037
|  |  |  |  |  +- s.mosesgiza.b6073a00.20120202-0037
|  |  +- s.mosesgiza.b6073a00.20120202-0037
```

Options: **--vars --stat --log ... --ignore=***steptype*

Finding Steps: **eman select**

- Step dirs don't have nice names.
- You need to locate steps of given properties.

What language models do I have?

- **eman ls lm**
- **eman select t lm**

If we need just the finished ones:

- **eman stat lm | grep DONE**
- **eman select t lm d**

And just 5-gram ones for English:

- **eman select t lm d vre ORDER=5 vre CORPAUG=en**

Deriving Experiments using **clone**

The text form of traceback allows to tweak the experiment:

- **eman tb** *step* | **sed 's/cs/de/'** | **eman clone** replicates our experiment on German instead of Czech.

The regex substitution is available in eman itself:

- **eman tb** *step* **-s '/cs/de/' -s '/form/lc/'**
  shows the traceback with the substitutions highlighted.
  - A good chance to check if the derivation does the intended.
- **eman tb** *step* **-s '/cs/de/' -s '/form/lc/'** \\
  | **eman clone --dry-run**
  - Last chance to check if existing steps get reused and what vars will new steps be based on.
  - Drop --**dry-run** to actually init the new steps.
  - Add --**start** if you're feeling lucky.

Hacking Welcome

Eman is designed to be hacking-friendly:

- Self-contained steps are easy to inspect:
  - all logs are there,
  - all (or most of) input files are there,
  - the main code (**eman.command**) is there,
  - often, even the binaries are there, or at least clearly identifiable.
- Step halfway failed?

  ⇒ Hack its **eman.command** and use **eman continue**.
- Seed not quite fit for your current needs?

  ⇒ Just init the step and hack **eman.seed**.

  ⇒ Or also prepare and hack **eman.command**.

Always mark manually tweaked steps, e.g. using eman's tags.

Fit for Cell-Phone SSH ☺

- Experiments run long but fail often.
- You don’t want to be chained to a computer.

Most eman commands have a short nickname.

- How are my last 10 merts?

  ```
  eman sel t mert l 10 --stat
  ```

Specify steps using any part of their name/hash or result:

- `s.foobar.`**a0f3**`b123.20120215-1011` failed, retry it:

  ```
  eman redo a0f3 --start
  ```

- How did I achieve this great BLEU score of 25.10?

  ```
  eman tb 25.10 --vars | less
  ```

Fit for Team Work

Playgrounds can be effectively merged:

- **eman add-remote** */home/fred/playground freds-exps*
- You can re-interpret Fred's results.
- You can clone Fred's experiments.
- You can make your steps depend on Fred's steps.
  - Only a shared file system is needed.

Caveat: we don't bother checking for conflicts yet.

Getting Started

“Install” eman in your home directory:

```
git clone https://redmine.ms.mff.cuni.cz/eman.git
```

Make sure eman is in your PATH: Bad things happen if not.

```
export PATH=$HOME/eman/bin/:$PATH
echo "export PATH=$HOME/eman/bin/:\$PATH" >> ~/.bashrc
```

Get our SMT Playground (with all the seeds):

```
git clone \
https://redmine.ms.mff.cuni.cz/ufal-smt-playground.git
```

Fix Perl Dependencies

Set up a local Perl repository.

```
 wget -O- http://cpanmin.us \
| perl - -l  /perl5 App::cpanminus local::lib
eval ‘perl -I  /perl5/lib/perl5 -Mlocal::lib‘
echo ’eval ‘perl -I  /perl5/lib/perl5 -Mlocal::lib‘’ >>  /.bashrc
```

You can copy the answer from:

`http://stackoverflow.com/a/2980715`

(just replace `.profile` with `.bashrc`)

Install the required packages:

```
cpanm YAML::XS
```

Confirm that `eman` runs:

```
eman --man
```

Setup Corpora

- Czech→English translation
- Training data: roughly 0.1% of CzEng 1.0 (15k sentence pairs)
- Dev set: 10% of WMT 2012 (300 sentence pairs)
- Test set: 10% WMT 2013 (300 sentence pairs)

Download the data:

```
http://bit.ly/mtm13corpora
```

Extract it into a subdirectory your playground, e.g.:

```
mkdir ~/ufal-smt-playground/playground/corpora
```

Importing the Corpora

- Every corpus has to “enter the world of eman”.
- This can be done using the seed `corpus`.

“`eman init corpus`” requires the following variables:

- `TAKE_FROM_COMMAND` command which produces the corpus
- `OUTCORP` corpus name
- `OUTLANG` corpus language
- `OUTFACTS` description of factors
- `OUTLINECOUNT` number of lines that we are expecting to get, used as a sanity check

Importing the Corpora

E.g. for training data, the Czech side:

```
TAKE_FROM_COMMAND="cat ../corpora/train.cs" \
OUTLINECOUNT=15000 \
OUTCORP=train OUTLANG=cs \
OUTFACTS=lc+lemma+tag \
eman init --start corpus
```

✎ Inspect the step directory. Where is the corpus stored?
✎ Create a bash script/“one-liner” to import all corpora: train/dev/test, cs/en (loop over sections and languages).

Did it work? Find out:

```
eman ls --stat
```

Frequent mistake: wrong OUTLINECOUNT for dev and test.

Listing and Printing Corpora

Corpman links symbolic names with `corpus` steps:

```
./corpman ls # show all registered corpora
```

Corpman ensures uniform pre-processing:

```
./corpman train/cs+lemma --dump
 # (Construct and) print the corpus as lemmas.
```

✎Bonus: Calculate the OOV (out-of-vocabulary) rate of the test data given the training data for:

- English vs. Czech and lowercase forms vs. lemmas

Use `ufal-smt-playground/scripts/count-oov.pl` or `oov.pl` from Moses. (Or write your own.)

Compiling Moses

In eman's philosophy, software is just data.

- Binaries should be compiled in timestamped step dirs.
- . . . so we know the exact code that was used.

Compile Moses and GIZA++:

```
eman init --start mosesgiza
```

✎ Examine the step dir. Where is the compilation log?
✎ Bonus (hard): Make another `mosesgiza` step where Moses prints "OOV" every time it encounters an out-of-vocabulary word.

Getting Moses binaries

- In your playground, download the binary:

```
wget http://ufal.mff.cuni.cz/~tamchyna/mosesgiza.64bit.tar.gz
```

- Extract it:

```
tar xzf mosesgiza.64bit.tar.gz
```

- Some hacking:

```
./fix-symlinks.sh
```

- Let eman know what we did:

```
eman reindex
```

Baseline Experiment

In your playground:

```
wget http://ufal.mff.cuni.cz/~tamchyna/baseline.traceback
eman clone --start < baseline.traceback
```

✎ While the experiment runs:

- Make a copy of the traceback
- Modify it to train word alignment on **lemmas** instead of **lc**. (But preserve the translation lc→lc!)
    - Note that `ALILABEL` is somewhat arbitrary but has to match between `align` and `tm`.

✎ Bonus: do the required edits using substitution in `eman`.

Hint: `eman --man`, look for the “`traceback`” command.

Looking Inside the Models

- Go to one of your baseline `model` steps, look at files:
- Language model: `lm.1.corpus.lm.gz`

  ✎ What is more probable: “united kingdom” or “united states”?

Looking Inside the Models

- Go to one of your baseline `model` steps, look at files:
- Language model: `lm.1.corpus.lm.gz`
  - ✎ What is more probable: “united kingdom” or “united states”?
  - ✎ Why are longer *n*-grams more probable than short ones?

Looking Inside the Models

- Go to one of your baseline `model` steps, look at files:
- Language model: `lm.1.corpus.lm.gz`

  ✎ What is more probable: “united kingdom” or “united states”?

  ✎ Why are longer *n*-grams more probable than short ones?
- Phrase table: `tm.1/model/phrase-table.0-0.gz`

  ✎ How do you say “hi” in Czech?

Looking Inside the Models

- Go to one of your baseline `model` steps, look at files:
- Language model: `lm.1.corpus.lm.gz`

  ✎ What is more probable: “united kingdom” or “united states”?

  ✎ Why are longer n-grams more probable than short ones?
- Phrase table: `tm.1/model/phrase-table.0-0.gz`

  ✎ How do you say “hi” in Czech?

  ✎ Phrase scores are $P(f|e), lex(f|e), P(e|f), lex(e|f)$.
  Given that, what do the counts in the last column mean?

  (Let’s look e.g. at the phrase “`ahoj ||| hi`”.)

Tuning

✎ How many iterations did MERT take?

Tuning

✎ How many iterations did MERT take?

✎ How did the BLEU score on the devset change?

Tuning

✎ How many iterations did MERT take?
✎ How did the BLEU score on the devset change?
✎ How much disk space did your MERTs need?

Tuning

✎ How many iterations did MERT take?
✎ How did the BLEU score on the devset change?
✎ How much disk space did your MERTs need?

- Standard Unix tool:

  ```
  eman du -sh s.mert.*
  ```

- Eman status:

  ```
  eman eman ls mert --dus --stat
  ```

Results

Let's compare MT quality (BLEU) of 2 systems:

- alignment on lowercase forms
- alignment on lemmas

✎ Look at `evaluator` steps. Which one is the baseline?

Results

Let's compare MT quality (BLEU) of 2 systems:

- alignment on lowercase forms
- alignment on lemmas

✎ Look at `evaluator` steps. Which one is the baseline?

- Trace back + grep:

  ```
  eman tb --vars s.evaluator.xyz | grep ALIAUG
  ```

Results

Let's compare MT quality (BLEU) of 2 systems:

- alignment on lowercase forms
- alignment on lemmas

✎ Look at `evaluator` steps. Which one is the baseline?

- Trace back + grep:

  ```
  eman tb --vars s.evaluator.xyz | grep ALIAUG
  ```

- Trace forward from the alignment step:

  ```
  eman tf $(eman sel t align vre 'SRC.*lc')
  ```

Results

Let's compare MT quality (BLEU) of 2 systems:

- alignment on lowercase forms
- alignment on lemmas

✎ Look at `evaluator` steps. Which one is the baseline?

- Trace back + grep:

  ```
  eman tb --vars s.evaluator.xyz | grep ALIAUG
  ```

- Trace forward from the alignment step:

  ```
  eman tf $(eman sel t align vre ’SRC.*lc’)
  ```

- Or just one `select` query:

  ```
  eman sel t evaluator br t align vre ’SRC.*lc’
  ```

Results

Let's compare MT quality (BLEU) of 2 systems:

- alignment on lowercase forms
- alignment on lemmas

✎ Look at `evaluator` steps. Which one is the baseline?

- Trace back + grep:

  ```
  eman tb --vars s.evaluator.xyz | grep ALIAUG
  ```

- Trace forward from the alignment step:

  ```
  eman tf $(eman sel t align vre 'SRC.*lc')
  ```

- Or just one `select` query:

  ```
  eman sel t evaluator br t align vre 'SRC.*lc'
  ```

BLEU is in the "`s.evaluator.../scores`" file.

Wild Experimenting

✎ Run word alignment on `lcstem4`, `lcstem5`.

Wild Experimenting

- ✎ Run word alignment on `lcstem4`, `lcstem5`.
- ✎ Try different orders of the language model (3, 4, 6).

Wild Experimenting

- Run word alignment on `lcstem4`, `lcstem5`.
- Try different orders of the language model (3, 4, 6).
- Translate from Czech lemmas into English forms (`lc`).

Wild Experimenting

✎ Run word alignment on `lcstem4`, `lcstem5`.

✎ Try different orders of the language model (3, 4, 6).

✎ Translate from Czech lemmas into English forms (`lc`).

✎ Try the opposite translation direction: English→Czech.

Wild Experimenting

✎ Run word alignment on `lcstem4`, `lcstem5`.

✎ Try different orders of the language model (3, 4, 6).

✎ Translate from Czech lemmas into English forms (`lc`).

✎ Try the opposite translation direction: English→Czech.

✎ Set up a factored system:

- lc→lc (baseline path), and
- lemma→lc (alternative path).

Summary

Hopefully, you now understand:

- within (PB)MT:
  - the structure of a (PB)MT experiment,
  - what is the language model and the translation model,
- meta-level:
  - eman's organization of the experimentation playground,
  - the idea of cloning of experiments.

Extra Slides

Eman is Versatile

What types of steps should I have?

- Any, depending on your application.

What language do I write steps in?

- Any, e.g. bash.

What are the input and output files of the steps?

- Any, just make depending steps understand each other.
- Steps can have many output files and serve as prerequisites to different types of other steps.

What are measured values of my experiments?

- Anything from any of the files any step produces.

What the User Implements: Just Seeds

Technically, a seed is any program that:

- responds to arbitrary environment variables,
- runs **eman defvar** to register step variables with eman,
- produces another program, **./eman.command** that does the real job.

The seed is actually run twice:

- At "init": to check validity of input variables and register them with eman.
- At "prepare": to produce **eman.command**.

The user puts all seeds in **playground/eman.seeds**.

- Eman runs a local copy of the seed in a fresh step dir.

eman redo

On cluster, jobs can fail nondeterminically.

- Bad luck when scheduled to a swamped machine.
- Bad estimate of hard resource limits (RAM exceeds the limit ⇒ job killed).

Eman to the rescue:

- **eman redo** *step* creates a new instance of each failed step, preserving the experiment structure.
- **eman redo** *step* **--start** starts the steps right away.

To make sure eman will do what you expect, first try:

- **eman redo** *step* **--dry-run**

eman clone

CLONING is initing a new step using vars of an existing one.

Cloning of individual steps is useful:

- when a step failed (used in **eman redo**),
- when the seed has changed,
- when we want to redefine some vars:
  ORDER=4 eman clone s.lm.1d6f791c...

Cloning of whole tracebacks:

- The text of a traceback gets instantiated as steps.
- Existing steps are reused if OK and with identical vars.
- **eman traceback** *step* | **eman clone**
- **eman traceback** *step* | **mail bojar@ufal**
  followed by **eman clone** < **the-received-mail**.

eman tag or **eman ls --tag** shows tags

TAGS and AUTOTAGS are:

- arbitrary keywords assigned to individual steps,
- inherited from dependencies.

Tags are:

- added using **eman add-tag** *the-tag steps*,
- stored in `s.stepdir.123/`**eman.tag**.

⇒ Use them to manually mark exceptions.

Autotags are:

- specified in `playground/`**eman.autotags** as regexes over step vars, e.g.: **/ORDER=(.*)/$1gr/** for LM,
- (re-)observed at **eman retag**.

⇒ Use them to systematically mark experiment branches.

eman collect

Based on rules in **eman.results.conf**, e.g.:

```
BLEU  */BLEU.opt                  BLEU\s*=\s*([^\s,]+)
Snts  s.eval*/corpus.translation  CMD: wc -l
```

eman collects results from all steps into **eman.results**:

```
# Step Name                        Status  Score Value Tags and Autotags
s.evaluator.11ccf590.20120208-1554 DONE     TER  31.04 5gr DEVwmt10 LMc-news towards-
s.evaluator.11ccf590.20120208-1554 DONE     PER  44.61 5gr DEVwmt10 LMc-news towards-
s.evaluator.11ccf590.20120208-1554 DONE     CDER 33.97 5gr DEVwmt10 LMc-news towards-
s.evaluator.11ccf590.20120208-1554 DONE     BLEU 12.28 5gr DEVwmt10 LMc-news towards-
s.evaluator.11ccf590.20120208-1554 DONE     Snts  3003 5gr DEVwmt10 LMc-news towards-
s.evaluator.29fa5679.20120207-1357 OUTDATED TER  17.66 5gr DEVwmt10 LMc-news
...                                ...      ...    ...
s.evaluator.473687bb.20120214-1509 FAILED   Snts  3003
```

- Perhaps hard to read.
- Easy to grep, sort, whatever, or **tabulate**.

eman tabulate to Organize Results

The user specifies in the file **eman.tabulate**:

- which results to ignore, which to select,
- which tags contribute to col labels, e.g. **TER**, **BLEU**,
- which tags contribute to row labels, e.g. **[0-9]gr**, **towards-[A-Z]+**, **PRO**.

Eman tabulates the results, output in **eman.niceresults**:

```
                 PER   CDER  TER   BLEU
5gr towards-CDER 44.61 33.97 31.04 12.28
5gr              44.19 33.76 31.02 12.18
5gr PRO          43.91 33.87 31.49 12.09
5gr towards-PER  44.44 33.52 30.74 11.95
```

Related Experiment Mgmt Systems

Eman is just one of many, consider also:

- LoonyBin (Clark et al., 2010)
  - ⊖ Clickable Java tool.
  - ⊕ Support for multiple clusters and scheduler types.
- Moses EMS (Koehn, 2010)
  - Experiment Management System primarily for Moses.
  - Centered around a single experiment which consists of steps.
- Pure Makefiles
  Yes, you can easily live with fancy Makefiles.
  - You will use commands like **make init.mert**
    or **cp -r exp.mert.1 exp.mert.1b**
  - You need to learn to use **$***, **$@** etc.
  - You are likely to implement your own eman soon. ☺

There are also the following workflow management systems: DAGMan, Pegasus, Dryad.

References

Jonathan H. Clark, Jonathan Weese, Byung Gyu Ahn, Andreas Zollmann, Qin Gao, Kenneth Heafield, and Alon Lavie. 2010. The Machine Translation Toolpack for LoonyBin: Automated Management of Experimental Machine Translation HyperWorkflows. Prague Bulletin of Mathematical Linguistics, 93:117–126.

Philipp Koehn. 2010. An Experimental Management System. Prague Bulletin of Mathematical Linguistics, 94:87–96, September.