

# FST Trimming: Ending Dictionary Redundancy in Apertium

Matthew Marting<sup>1</sup>   Kevin Unhammer<sup>2</sup>

<sup>1</sup>St. David's School  
Raleigh, NC.

0

<sup>2</sup>Kaldera språkteknologi  
Stavanger, Noreg  
unhammer+apertium@mm.st

27th May 2014

Introduction and  
background

FSTs in the Apertium pipeline

The Problem: Redundant data

A Solution: trim on compile

Implementation of  
lt-trim

Preprocessing the bilingual  
dictionary

Intersection

lt-trim in use

Ending Dictionary  
Redundancy

Conclusion

# Outline of talk

lt-trim

Matthew Marting,  
Kevin Unhammer

## Introduction and background

FST's in the Apertium pipeline

The Problem: Redundant data

A Solution: trim on compile

## Implementation of lt-trim

Preprocessing the bilingual  
dictionary

Intersection

lt-trim in use

## Ending Dictionary Redundancy

## Conclusion

Introduction and background

Implementation of lt-trim

Ending Dictionary Redundancy

Conclusion

# Introduction and background

lt-trim

Matthew Marting,  
Kevin Unhammer

## Introduction and background

FST's in the Apertium pipeline

The Problem: Redundant data

A Solution: trim on compile

## Implementation of lt-trim

Preprocessing the bilingual  
dictionary

Intersection

lt-trim in use

## Ending Dictionary Redundancy

## Conclusion

- ▶ Apertium: Free/Open Source, Rule-based Machine Translation platform
- ▶ Goals include:
  - ▶ supporting lesser-resourced languages
  - ▶ wide coverage
  - ▶ post-editable output
  - ▶ reusable resources
- ▶ Language data (dictionaries, etc.) typically organised in language *pairs* (Catalan-Spanish, Portuguese-Spanish, etc.)
  - ▶ historically: each with its own copy of monolingual data



Lea go šaddan nu ahte mii boares olbmot eat šat gula olmmošlohkui go eat leat dan ✖  
joavdelas interneahtas?

Mii dat lea sámegillii? Pláhtta čuojanas? Lea somá bidjat pláhta jorrat, dat jorrá ja  
jorrá ja musihkka čuodjá lanjas.

Mun lean guhká bargan dáinna dutkamiin, ja lean ilus go viimmat sáhtán čájehit  
bohtosiid!

Har det blitt slik at vi gamle folk ikke mer hører til mennesketallet når vi ikke er på  
dets nyttige internett?

Hva er det på samisk? En plate et musikkinstrument? Det er morsomt å sette  
platen å dreie, det dreier og dreier og musikk klinger i rommet.

Jeg har lenge arbeidd med denne undersøkelsen, og jeg er på gleden når endelig  
kan jeg vise resultat!

☒ Välitön käännös

# Apertium pipeline architecture

lt-trim

Matthew Marting,  
Kevin Unhammer

Introduction and  
background

FSTs in the Apertium pipeline

The Problem: Redundant data

A Solution: trim on compile

Implementation of  
lt-trim

Preprocessing the bilingual  
dictionary

Intersection

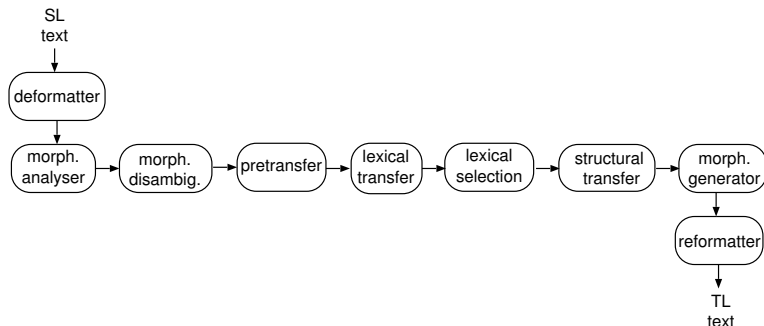
lt-trim in use

Ending Dictionary  
Redundancy

Conclusion

## ► Ittoolbox Finite State Transducers used for, among others:

- morph. analysis: 'fishes' to  
fish<n><pl>/fish<vblex><pres>
- lex. transfer: fish<n><pl> to fisk<n><m><pl>
- morph. generation: fisk<n><m><pl><def> to 'fiskane'



# Multiword support

lt-trim

Matthew Marting,  
Kevin Unhammer

Introduction and  
background

FSTs in the Apertium pipeline

The Problem: Redundant data

A Solution: trim on compile

Implementation of  
lt-trim

Preprocessing the bilingual  
dictionary

Intersection

lt-trim in use

Ending Dictionary  
Redundancy

Conclusion

Ittoolbox FST's support a variety of multiwords

An Ittoolbox “lexical unit” is one token, and can be:

- ▶ simple non-multi-words: ‘fish’
- ▶ simple space-separated words: ‘hairy frogfish’ as a single token
- ▶ multiwords with **inner inflection**: ‘takes out’,  
analysed as `take<vblex><pri><p3><sg># out`,  
converted to `take# out<vblex><pri><p3><sg>` before  
lexical transfer

# Multiword support

- ▶ **joined** multiwords: 'they'll';

analysed as single token

prpers<prn><subj><p3><mf><pl>+will<vaux><inf>,

then split into two tokens

prpers<prn><subj><p3><mf><pl> and

will<vaux><inf> before lexical transfer

- ▶ **compounds**: 'frogfish';

analysed as single token frog<n><sg>+fish<n><pl>,

then split into two tokens frog<n><sg> and fish<n><pl>

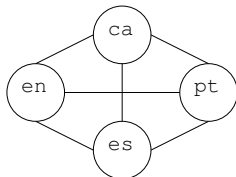
before lexical transfer

# Multiword support

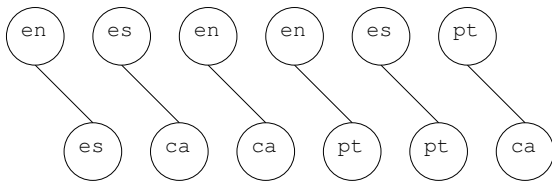
- ▶ combinations (space-separated + joined + inner inflection):  
‘creure-ho que’,  
analysed as single token  
creure<vblex><inf>+ho<prn><enc><p3><nt># que,  
then moved and split into two tokens  
creure# que<vblex><inf> and  
ho<prn><enc><p3><nt> before lexical transfer



# The Problem: Redundant data



Ideal number of monodixes with four languages



Current number of monodixes with pairs of four languages

Introduction and  
background

FSTs in the Apertium pipeline

The Problem: Redundant data

A Solution: trim on compile

Implementation of  
lt-trimPreprocessing the bilingual  
dictionary

Intersection

lt-trim in use

Ending Dictionary  
Redundancy

## Conclusion

Words in analyser but missing from lexical transfer can be problematic:

- ▶ ‘fishes’ to ‘@fish’: loses the inflection
- ▶ ‘gikk til hundene’ “went to the dogs” to ‘went to @hund’  
“went to dog”: losing the inflection hides the idiomatic meaning
- ▶ ‘öldürmedi’ “did not kill” to ‘@öl’ “kill”: loses the *negation*
- ▶ lexical transfer is also tag transfer – structural transfer thus needs exceptions for half-translated tags

Introduction and  
background

FST's in the Apertium pipeline

The Problem: Redundant data

A Solution: trim on compile

Implementation of  
lt-trimPreprocessing the bilingual  
dictionary

Intersection

lt-trim in use

Ending Dictionary  
Redundancy

## Conclusion

But, most importantly, multiword tokenisation means that

‘He takes out the trash’ translates to ‘Han @take out søpla’ *even though both ‘take’-‘ta’ and ‘out’-‘ut’ are in the bilingual dictionary.*

Adding more words makes the translator worse!

# A Solution: trim on compile

lt-trim

Matthew Marting,  
Kevin Unhammer

Introduction and  
background

FSTs in the Apertium pipeline

The Problem: Redundant data

A Solution: trim on compile

Implementation of  
lt-trim

Preprocessing the bilingual  
dictionary

Intersection

lt-trim in use

Ending Dictionary  
Redundancy

Conclusion

Compile a *trimmed* analyser-FST containing only those entries from original analyser FST that would pass through bilingual FST.

- ▶ FSA's closed under intersection:  $FSA1 \cap FSA2 = FSA3$
- ▶ Similarly, we can compose-intersect FST's:  
output-side of FST1  $\cap$  input-side of FST2 = FST3

Goal: One big monolingual source dictionary, trimmed during compile to language-pair specific analysers.

Dictionaries in HFST instead of Ittoolbox can trim already (but it breaks with compounds!).

Most Apertium dictionaries use Ittoolbox.

Our tool needs some exceptions to compose-intersect:

- ▶ Append .\* (any-symbol loop) to bilingual FST
  - ▶ Lexical transfer only needs a match on the start of the string
- ▶ Reorder #-multiwords in bilingual FST
  - ▶ so they look like analyser (else they won't match)
- ▶ Let + in analyser mean transition-to-start in bilingual FST
  - ▶ since single token  $a+b$  in analyser is split into two tokens  $a$   $b$  before lexical transfer

Introduction and  
background

FST's in the Apertium pipeline

The Problem: Redundant data

A Solution: trim on compile

Implementation of  
lt-trimPreprocessing the bilingual  
dictionary

Intersection

lt-trim in use

Ending Dictionary  
Redundancy

## Conclusion

Digression: Some Apertium dictionaries use HFST instead of lttoolbox.

Trimming with standard HFST commands works for most but not all such dictionaries.

Again multiword issues: compounds.

# Implementation of `lt-trim`

lt-trim

Matthew Marting,  
Kevin Unhammer

Introduction and  
background

FSTs in the Apertium pipeline

The Problem: Redundant data

A Solution: trim on compile

Implementation of  
`lt-trim`

Preprocessing the bilingual  
dictionary

Intersection

`lt-trim` in use

Ending Dictionary  
Redundancy

Conclusion

Tool takes two compiled FST's, produces a new, *trimmed* FST

1. Preprocess bilingual FST
  - 1.1 “Prefixing”: Append any-symbol loop
  - 1.2 Reorder #-multiwords
2. Depth-first intersection of output-side of analyser with input-side of bilingual FST
  - ▶ with an exception on seeing +

# Prefixing bilingual FST

## Introduction and background

FSTs in the Apertium pipeline

The Problem: Redundant data

A Solution: trim on compile

## Implementation of lt-trim

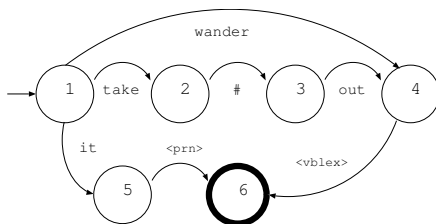
Preprocessing the bilingual dictionary

Intersection

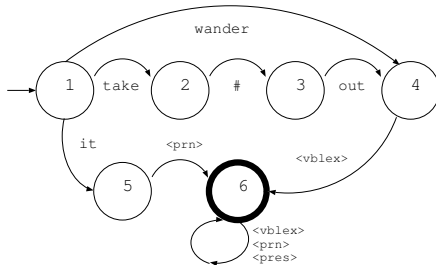
lt-trim in use

## Ending Dictionary Redundancy

## Conclusion



Input bilingual FST (letter transitions compressed to single arcs)



“Prefixed” bilingual FST (any-symbol loop appended)



# Moving uninflected lemma parts in bilingual FST

lt-trim

Matthew Marting,  
Kevin Unhammer

Introduction and  
background

FSTs in the Apertium pipeline

The Problem: Redundant data

A Solution: trim on compile

Implementation of  
lt-trim

Preprocessing the bilingual  
dictionary

Intersection

lt-trim in use

Ending Dictionary  
Redundancy

Conclusion

Want `take# out<vblex>` to become `take<vblex># out`, so

1. Depth-first traverse bilingual FST
2. On seeing a #, replace the transition  $t$  with results of `copyWithTagsFirst(t)`
3. Function `copyWithTagsFirst(t)` builds a new partial FST where any tag sequence and uninflected lemma parts have swapped places

# Moving uninflected lemma parts in bilingual FST

lt-trim

Matthew Marting,  
Kevin Unhammer

## Introduction and background

FSTs in the Apertium pipeline

The Problem: Redundant data

A Solution: trim on compile

## Implementation of lt-trim

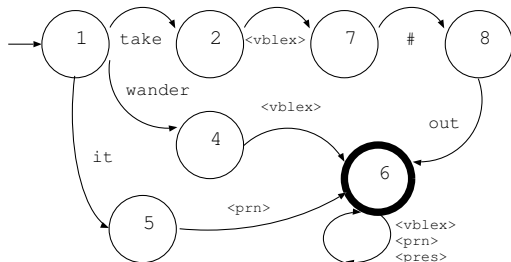
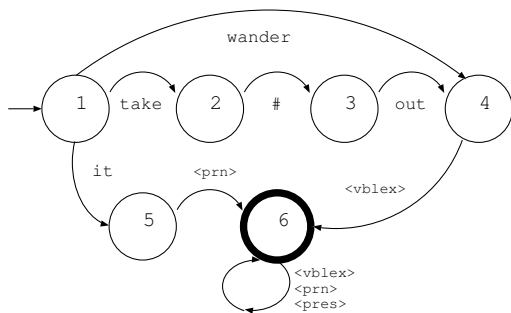
Preprocessing the bilingual dictionary

Intersection

lt-trim in use

## Ending Dictionary Redundancy

## Conclusion



# Intersection

lt-trim

Matthew Marting,  
Kevin Unhammer

## Introduction and background

FSTs in the Apertium pipeline

The Problem: Redundant data

A Solution: trim on compile

## Implementation of lt-trim

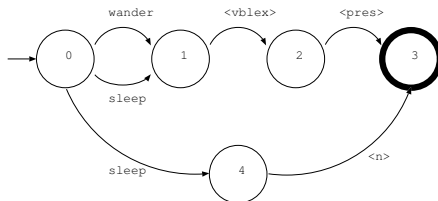
Preprocessing the bilingual dictionary

### Intersection

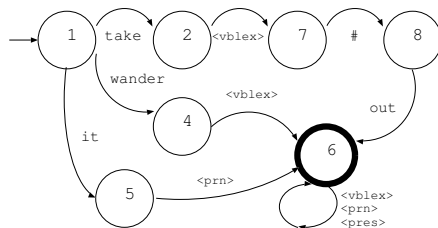
lt-trim in use

## Ending Dictionary Redundancy

## Conclusion



Input analyser



Preprocessed bilingual FST

Matthew Marting,  
Kevin Unhammer

## Introduction and background

FSTs in the Apertium pipeline

The Problem: Redundant data

A Solution: trim on compile

## Implementation of lt-trim

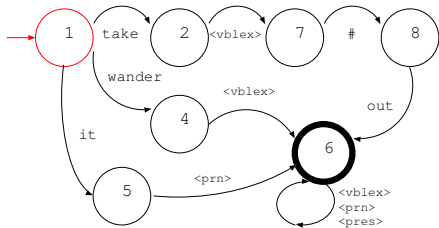
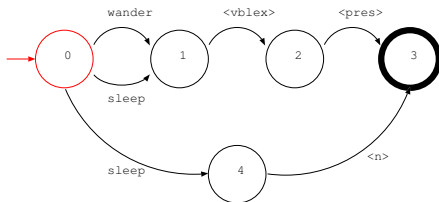
Preprocessing the bilingual dictionary

### Intersection

lt-trim in use

## Ending Dictionary Redundancy

## Conclusion



Introduction and  
background

FST's in the Apertium pipeline

The Problem: Redundant data

A Solution: trim on compile

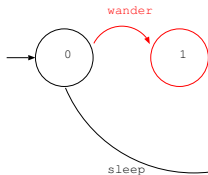
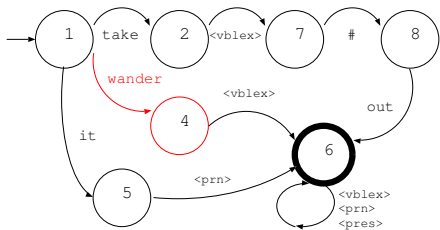
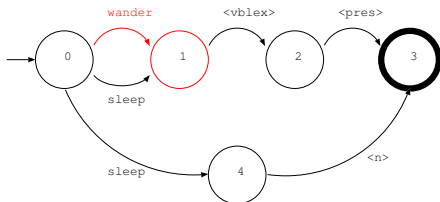
Implementation of  
lt-trimPreprocessing the bilingual  
dictionary

## Intersection

lt-trim in use

Ending Dictionary  
Redundancy

## Conclusion



Introduction and  
background

FSTs in the Apertium pipeline

The Problem: Redundant data

A Solution: trim on compile

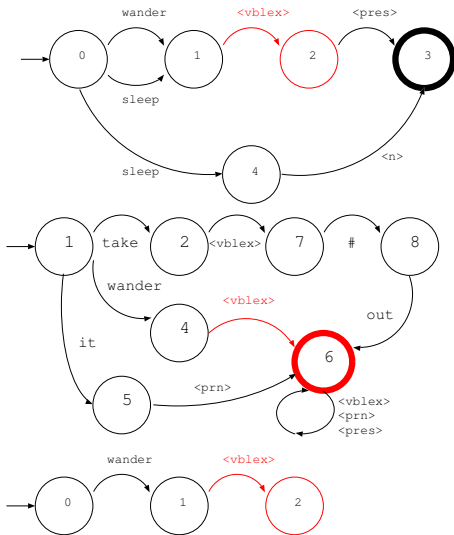
Implementation of  
lt-trimPreprocessing the bilingual  
dictionary

## Intersection

lt-trim in use

Ending Dictionary  
Redundancy

## Conclusion



Introduction and  
background

FSTs in the Apertium pipeline

The Problem: Redundant data

A Solution: trim on compile

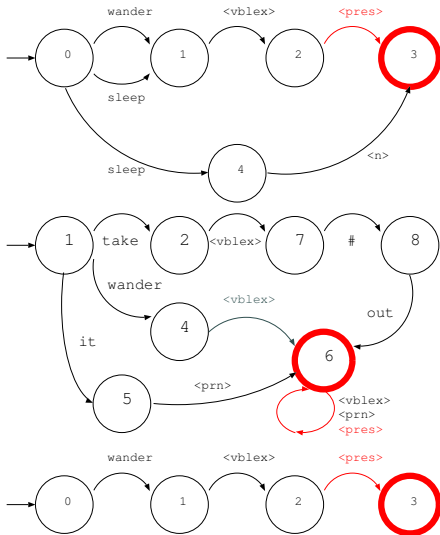
Implementation of  
lt-trimPreprocessing the bilingual  
dictionary

## Intersection

lt-trim in use

Ending Dictionary  
Redundancy

## Conclusion



Introduction and  
background

FSTs in the Apertium pipeline

The Problem: Redundant data

A Solution: trim on compile

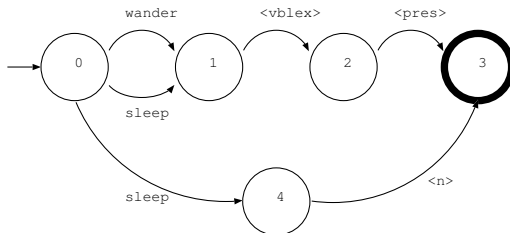
Implementation of  
lt-trimPreprocessing the bilingual  
dictionary

## Intersection

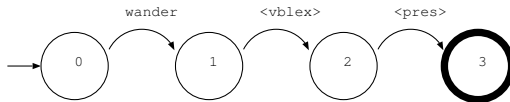
lt-trim in use

Ending Dictionary  
Redundancy

## Conclusion



Input analyser



Fully trimmed analyser



# Intersection with multiwords

## Introduction and background

FSTs in the Apertium pipeline

The Problem: Redundant data

A Solution: trim on compile

## Implementation of lt-trim

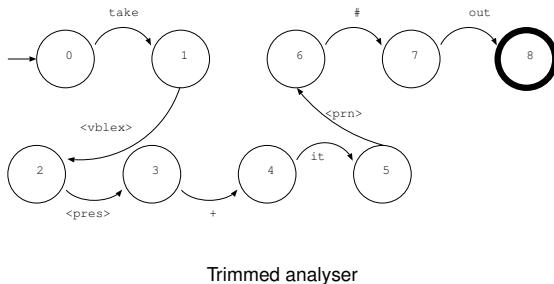
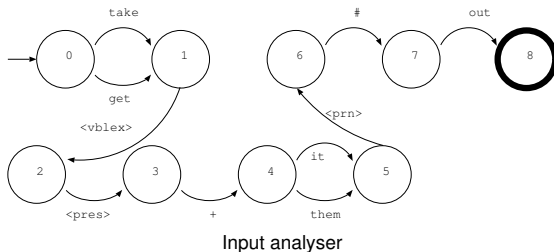
Preprocessing the bilingual dictionary

### Intersection

lt-trim in use

## Ending Dictionary Redundancy

## Conclusion



# lt-trim in use

```
$ lt-trim full-ana.bin bi.bin trimmed-ana.bin
```

(But language pair developers typically just type “make”.)

Speed and memory usage is comparable to regular Ittoolbox  
(lt-comp) compiling.

# Ending Dictionary Redundancy

lt-trim

Matthew Marting,  
Kevin Unhammer

Introduction and  
background

FSTs in the Apertium pipeline

The Problem: Redundant data

A Solution: trim on compile

Implementation of  
lt-trim

Preprocessing the bilingual  
dictionary

Intersection

lt-trim in use

Ending Dictionary  
Redundancy

Conclusion

- ▶ New Autotools rules let us formally depend on monolingual data packages
- ▶ All new languages added to Apertium use this system – no monolingual data redundancy
- ▶ But: Implementing trimming in old/well-developed language pairs means manual merging – divergent dictionaries problematic
  - ▶ Merging Norwegian Bokmål between sme-nob and nno-nob: about 3 hrs work
  - ▶ dan-nob added further 3 hrs

# Conclusion

lt-trim

Matthew Marting,  
Kevin Unhammer

Introduction and  
background

FSTs in the Apertium pipeline

The Problem: Redundant data

A Solution: trim on compile

Implementation of  
lt-trim

Preprocessing the bilingual  
dictionary

Intersection

lt-trim in use

Ending Dictionary  
Redundancy

Conclusion

- ▶ `lt-trim` works
- ▶ Monolingual data now in a single `/languages/` SVN module, easier for other projects to find and use our data
- ▶ Next up: special-purpose HFST trimming tool to get around compounding problem?
- ▶ Still much manual merge work to be done