FST Trimming: Ending Dictionary Redundancy in Apertium

Francis Tyers[0] Matthew Marting[1] Kevin Unhammer[2]

[0]UiT Norgga árktalaš universitehta
Romssa, Norga
ftyers@prompsit.com

[1]St. David's School
Raleigh, NC.
∅

[2]Kaldera språkteknologi
Stavanger, Noreg
unhammer+apertium@mm.st

27th May 2014

Outline of talk

Introduction and background

Implementation of `lt-trim`

Ending Dictionary Redundancy

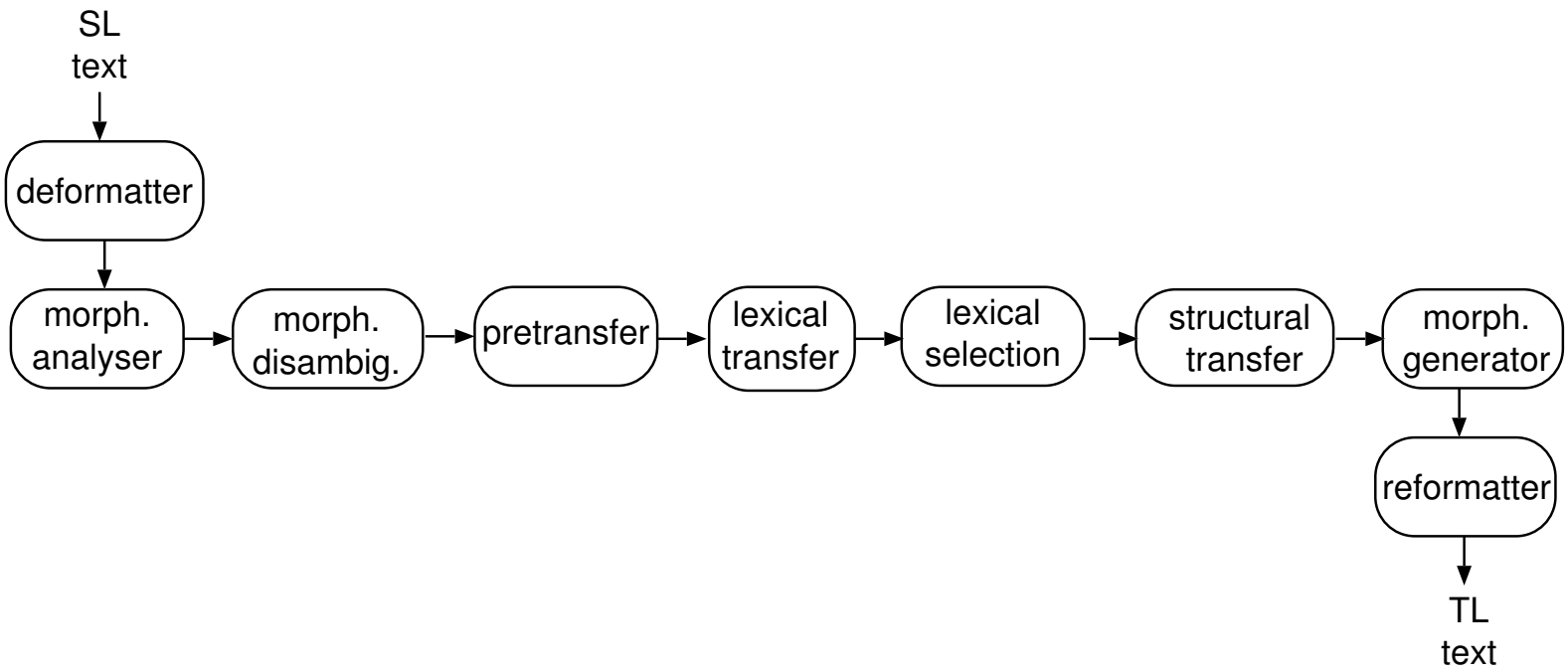
Conclusion

Introduction and background

- 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

Apertium pipeline architecture

- lttoolbox 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

lttoolbox FST's support a variety of multiwords

An lttoolbox "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

Francis Tyers,
Matthew Marting,
Kevin Unhammer

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

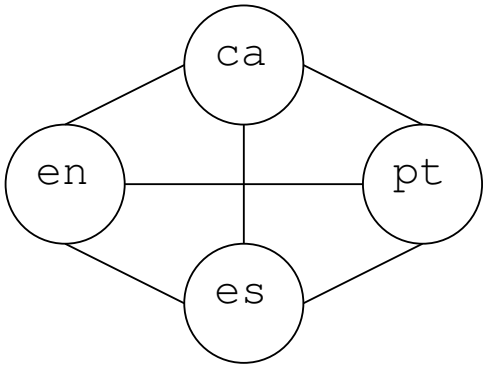


Figure: Ideal number of monodixes with four languages

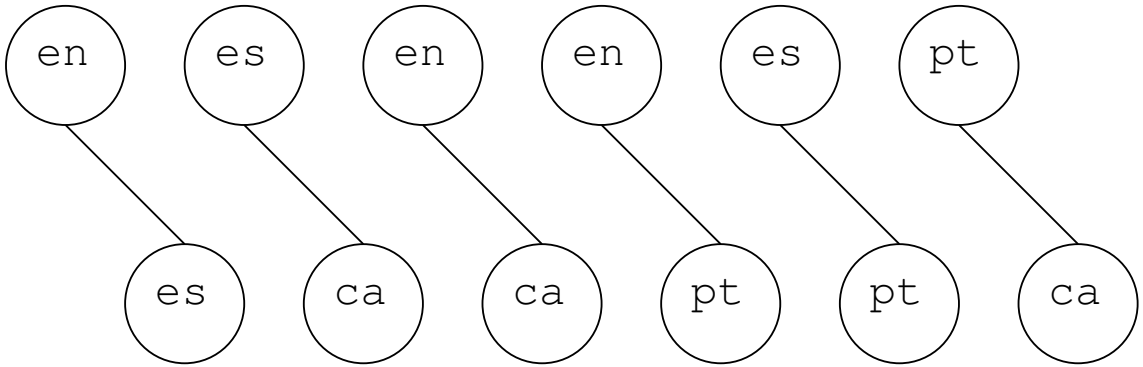


Figure: Current number of monodixes with pairs of four languages

Francis Tyers,
Matthew Marting,
Kevin Unhammer

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

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: Intersection

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

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

- We know we can do: FSA1 ∩ FSA2 = FSA3
- We want: output of FST1 ∩ input of FST2 = FST3

With some exceptions:

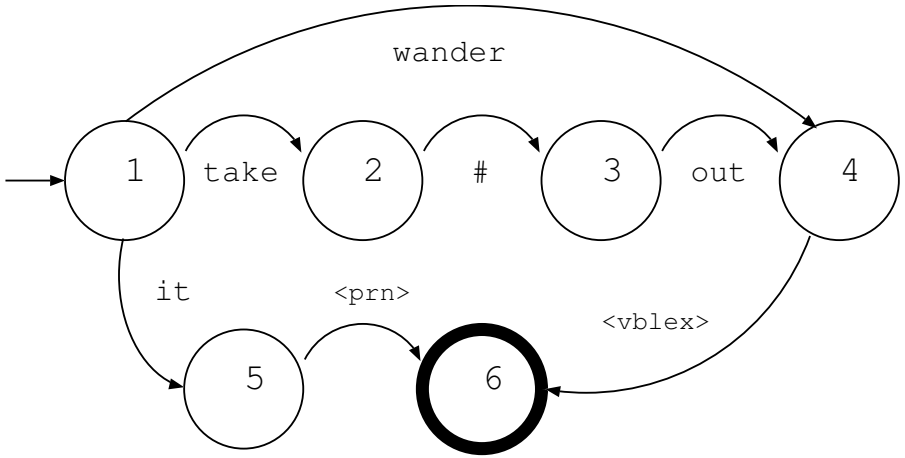
- Append `.*` (any-symbol loop) to bilingual FST
  - Lexical transfer only needs a match on the start of the string
- Reorder `#`-multiwords in bilingual FST
  - so they look like analyser (else they won't match)
- Let `+` in analyser mean transition-to-start in bilingual FST
  - since single token `a+b`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

Implementation of `lt-trim`

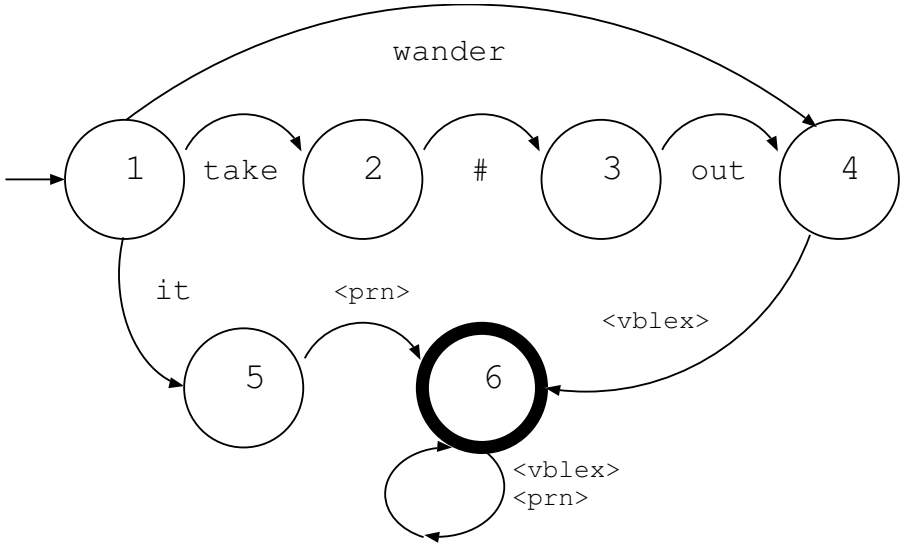
Tool takes two compiled FST's, produces a new, *trimmed* FST

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



Input bilingual FST (letter transitions compressed to single arcs)



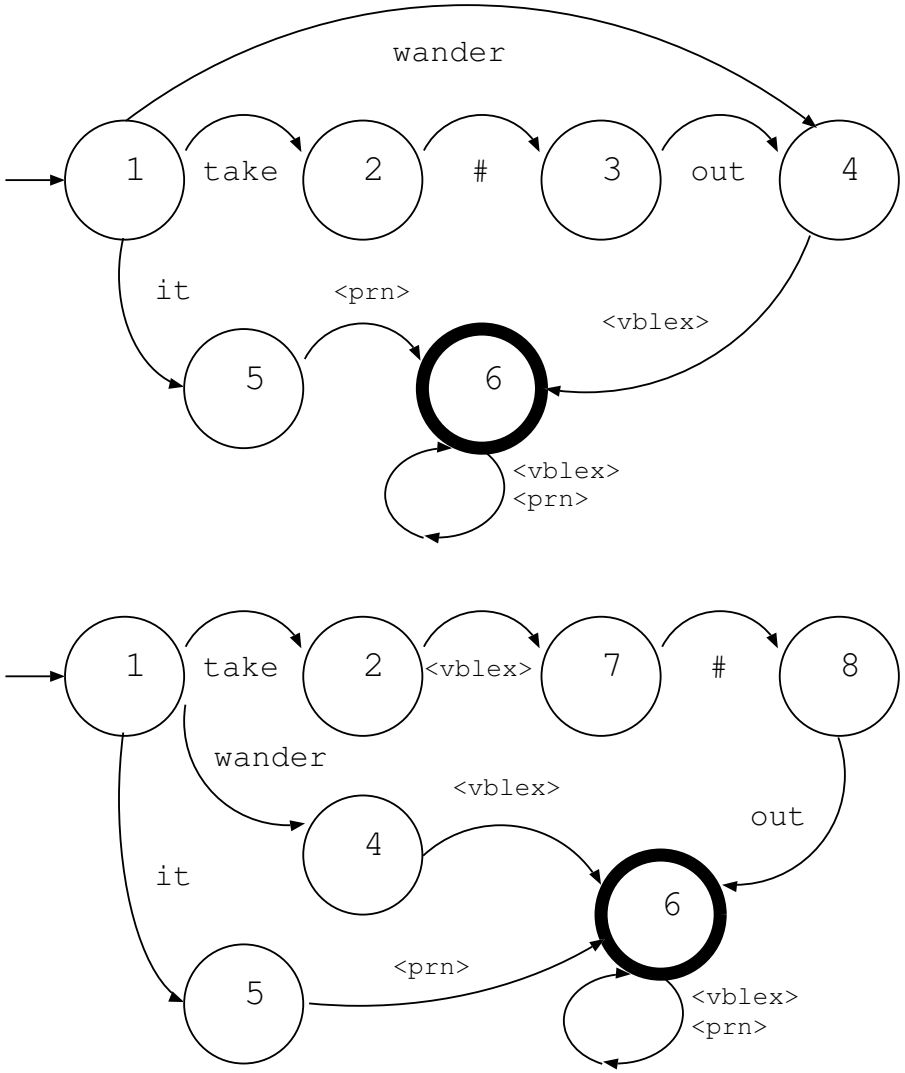
“Prefixed” bilingual FST (any-symbol loop appended)

Moving uninflected lemma parts in bilingual FST

Want: `take# out<vblex>` to become `take<vblex># out` and so

1. Do a depth-first traversal
2. On seeing a #, replace the transition *t* with results of copyWithTagsFirst(t)
3. This function builds a new partial FST where tag sequence and uninflected lemma part are reordered

Moving uninflected lemma parts in bilingual FST



Fully preprocessed; now matches both `take<vblex># out` and `take<vblex>+it<prn># out` (assuming special +-handling)

Francis Tyers,
Matthew Marting,
Kevin Unhammer

Intersection

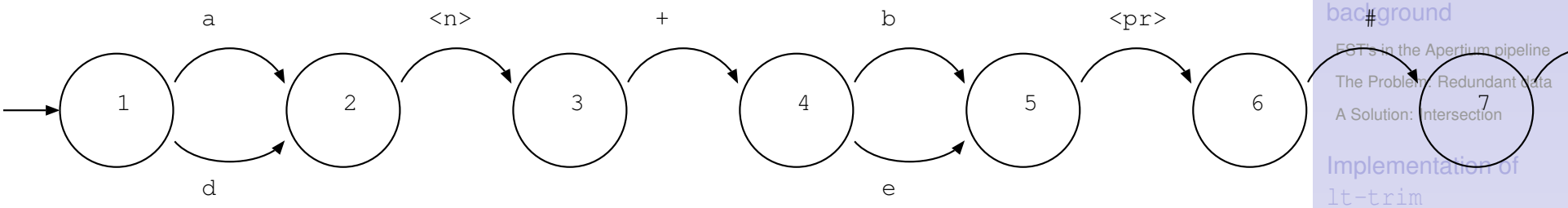


Figure: Input monolingual FST

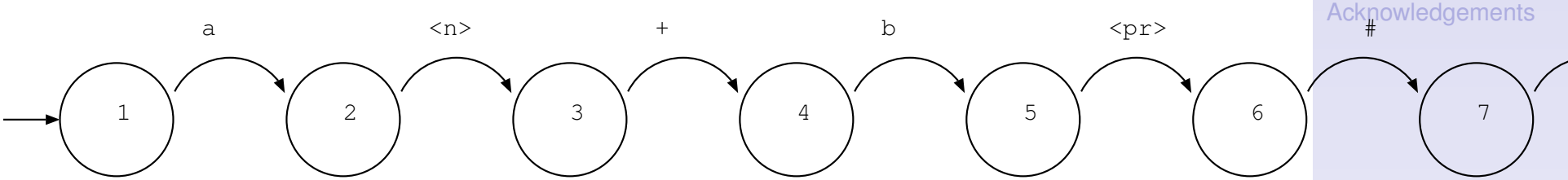


Figure: Trimmed monolingual FST

lt-trim in use

Ending Dictionary Redundancy

Conclusion

-

Acknowledgements