Subregular toolkit implemented in Python

Alëna Aksënova

Stony Brook University

IACS Jr. Researcher Award Presentation

IACS @ SBU
August 16, 2018
Subregular toolkit: general information

**kist**: kist implementing subregular toolkit

**Motivation**: to collect in one place the functionality for subregular languages and subsequential transducers.

- **For researchers**: to avoid manual burden of extracting grammars and designing transducers, creating data samples, or scanning strings;
- **For practitioners**: to start using tools in practice that are currently available only in the literature.

- Python 3 (will be available via pip)
- Open source
- Available on GitHub
  # https://github.com/loisetoil/slp
Subregular toolkit: general information

**kist**: kist implementing subregular toolkit

**Motivation**: to collect in one place the functionality for subregular languages and subsequential transducers.

- **For researchers**: to avoid manual burden of extracting grammars and designing transducers, creating data samples, or scanning strings;
- **For practitioners**: to start using tools in practice that are currently available only in the literature.

- Python 3 (will be available via pip)
- Open source
- Available on GitHub
  
  # https://github.com/loisetoil/slp
kist: kist implementing subregular toolkit

Motivation: to collect in one place the functionality for subregular languages and subsequential transducers.

- For researchers: to avoid manual burden of extracting grammars and designing transducers, creating data samples, or scanning strings;
- For practitioners: to start using tools in practice that are currently available only in the literature.

Python 3 (will be available via pip)
Open source
Available on GitHub

# https://github.com/loisetoil/slp
More motivations

This subregular toolkit allows one to:

- use recent theoretical results in practice;
- test ideas currently available in the literature;
- explore new methods to model natural language;
- automatically extract dependencies therefore avoiding manual burden of automata/transducer construction.
More motivations

This subregular toolkit allows one to:

- use recent theoretical results in practice;
- test ideas currently available in the literature;
- explore new methods to model natural language;
- automatically extract dependencies therefore avoiding manual burden of automata/transducer construction.
The importance of formalization

- In order to abstract away from details and look at the big picture, we need to formalize:
  - Languages $\rightarrow$ sets of strings of a particular type;
  - Functions $\rightarrow$ descriptions of processes.

- **KIST** toolkit provides functionality that allows one to work with (sub)regular languages and functions.

- Such a toolkit is useful for NLP, and not only.
Here, I only work with (sub)regular – requiring a finite amount of memory – languages and functions.
What is done and what is left

**Last year:**

- ✔ FSA implementation:
  - ✔ architecture;
  - ✔ optimization.
- ✔ Languages (SL, TSL, SP):
  - ✔ learners;
  - ✔ scanners;
  - ✔ sample generators;
  - ✔ neg↔pos switch;
  - ✔ corresponding FSA.

**This year:**

- Languages (MTSL, SS-TSL):
  - learners;
  - scanners;
  - sample generators;
  - neg↔pos switch;
  - corresponding FSA.
- Transduction learners:
  - OSTIA;
  - ISLFLA;
  - OSLFIA.
What is done and what is left

Last year:

✓ FSA implementation:
  ✓ architecture;
  ✓ optimization.

✓ Languages (SL, TSL, SP):
  ✓ learners;
  ✓ scanners;
  ✓ sample generators;
  ✓ neg↔pos switch;
  ✓ corresponding FSA.

This year:

☐ Languages (MTSL, SS-TSL):
  ☐ learners;
  ☐ scanners;
  ☐ sample generators;
  ☐ neg↔pos switch;
  ☐ corresponding FSA.

☐ Transduction learners:
  ☐ OSTIA;
  ☐ ISLFLA;
  ☐ OSLFIA.
The class of regular languages consists of smaller sub-classes. (McNaughton & Papert 1971)

For every (sub)regular language, it is possible to construct a corresponding finite state automaton.

Most subregular classes are learnable in polynomial time with positive data only.

There is a variety of applications for subregular languages!
The class of regular languages consists of smaller sub-classes. (McNaughton & Papert 1971)

For every (sub)regular language, it is possible to construct a corresponding finite state automaton.

Most subregular classes are learnable in polynomial time with positive data only.

There is a variety of applications for subregular languages!
What are the applications?

Applications

- **Linguistics**
  - Sounds (Heinz 2010)
  - Words (Aksënova et. al 2016)

- **Robotics**
  - Sentences (Graf&Heinz 2015)
  - Meaning (Graf 2017)

- **Experiments with NN**
  - Rawal et. al 2011
  - Avcu et. al 2017

**What are the applications?**

- **Linguistics**
  - Sounds (Heinz 2010)
  - Words (Aksënova et. al 2016)

- **Robotics**
  - Sentences (Graf&Heinz 2015)
  - Meaning (Graf 2017)

- **Experiments with NN**
  - Rawal et. al 2011
  - Avcu et. al 2017
Subregular languages in KIST

Implemented functionality:
- learners;
- scanners;
- sample generators;
- negative ↔ positive grammar translators;
- constructing corresponding FSA;
- trimming FSA.
Subregular languages in KIST

Implemented functionality:

- learners;
- scanners;
- sample generators;
- negative ↔ positive grammar translators;
- constructing corresponding FSA;
- trimming FSA.
**Language example**

- **Language**: **Bukusu** (Kenya)

- **Construction**: \( V + \text{el/er/il/ir} \)
  ‘use something to \( V \)’

- **Rule**: “match the sounds of the suffix with the sounds of the verb”

- **Example**:
  - \( \text{tleex-el} \) ‘use smth to cook’
  - \( \text{reeb-er} \) ‘use smth to ask’
  - \( \text{lim-il} \) ‘use smth to cultivate’
  - \( \text{ir-ir} \) ‘use smth to die’
**Language example**

- **Language:** **Bukusu** (Kenya)

- **Construction:** $V + \text{el/er/il/ir}$
  - ‘use something to $V$’

- **Rule:** “match the sounds of the suffix with the sounds of the verb”

- **Example:**
  - **tleex-el** ‘use smth to cook’
  - **reeb-er** ‘use smth to ask’
  - **lim-il** ‘use smth to cultivate’
  - **ir-ir** ‘use smth to die’

- **Map of Kenya**
Language example [cont.]

- **Language:** Bukusu (Kenya)

- **Construction:** V + el/er/il/ir
  ‘use something to V’

- **Rule:** “match the sounds of the suffix with the sounds of the verb”

- tleex-el ‘use smth to cook’
- reeb-er ‘use smth to ask’
- lim-il ‘use smth to cultivate’
- ir-ir ‘use smth to die’

Simple formal version of the pattern: 
\[(l,e)^+ \cup (l,i)^+ \cup (r,e)^+ \cup (r,i)^+\]

Intuition is that [e] and [i] need to agree with each other, as well as [l] and [r]. Among themselves, these two agreements do not interact.
Language example [cont.]

- **Language:** Bukusu (Kenya)
- **Construction:** $V + \text{el/er/il/ir}$
  - ‘use something to $V$’
- **Rule:** “match the sounds of the suffix with the sounds of the verb”
  - **Example:**
    - *tleex-el* ‘use smth to cook’
    - *reeb-er* ‘use smth to ask’
    - *lim-il* ‘use smth to cultivate’
    - *ir-ir* ‘use smth to die’

Simple formal version of the pattern: $\ (l,e)^+ \cup (l,i)^+ \cup (r,e)^+ \cup (r,i)^+$

- **Example:**
  - $\text{ok }\text{llliilliiii}$
  - $\text{ok }\text{eeerreer}$
  - $\text{ok }\text{lleeelle}$
  - $\text{ok }\text{riiriirrr}$
  - $\neg \text{liiirriii}$
  - $\neg \text{leeelliiii}$
  - $\ldots$
  - $\ldots$

Intuition is that $[e]$ and $[i]$ need to agree with each other, as well as $[l]$ and $[r]$. Among themselves, these two agreements do not interact.
Language example [cont.]

- **Language**: **Bukusu** (Kenya)
- **Construction**: V + el/er/il/ir
  ‘use something to V’
- **Rule**: “match the sounds of the suffix with the sounds of the verb”
  - tleex-el ‘use smth to cook’
  - reeb-er ‘use smth to ask’
  - lim-il ‘use smth to cultivate’
  - ir-ir ‘use smth to die’

Simple formal version of the pattern:
\[(l,e)^+ \cup (l,i)^+ \cup (r,e)^+ \cup (r,i)^+\]

- \(ok\) illiiiiiii
- \(ok\) eererreer
- \(ok\) lleellle
- \(ok\) riiriirrr

Intuition is that [e] and [i] need to agree with each other, as well as [l] and [r]. Among themselves, these two agreements do not interact.
(l, e)⁺ ∪ (l, i)⁺ ∪ (r, e)⁺ ∪ (r, i)⁺

- **Complexity**: MTSL (multiple tier-based strictly local)
- **Meaning**: there are several sets of items involved in long-distance dependency.

- $T_1 = \{l, r\}$, and $G_{1_{pos}} = \langle ll, rr \rangle$
- $T_2 = \{e, i\}$, and $G_{2_{pos}} = \langle ee, ii \rangle$
Language example [cont.]

\((l, e)^+ \cup (l, i)^+ \cup (r, e)^+ \cup (r, i)^+\)

- **Complexity**: MTSL (multiple tier-based strictly local)
- **Meaning**: there are several sets of items involved in long-distance dependency.

- \(T_1 = \{l, r\}\), and \(G_{1pos} = \langle ll, rr \rangle\)
- \(T_2 = \{e, i\}\), and \(G_{2pos} = \langle ee, ii \rangle\)
Language example [cont.]

\[(l,e)^+ \cup (l,i)^+ \cup (r,e)^+ \cup (r,i)^+\]

- **Complexity**: MTSL (multiple tier-based strictly local)
- **Meaning**: there are several sets of items involved in long-distance dependency.

- \(T_1 = \{l, r\}\), and \(G_{1\text{pos}} = \langle ll, rr \rangle\)
- \(T_2 = \{e, i\}\), and \(G_{2\text{pos}} = \langle ee, ii \rangle\)
Language example [cont.]

\[(l,e)^+ \cup (l,i)^+ \cup (r,e)^+ \cup (r,i)^+\]

- **Complexity**: MTSL (multiple tier-based strictly local)
- **Meaning**: there are several sets of items involved in long-distance dependency.

- \(T_1 = \{ l, r \}, \text{ and } G_{1_{pos}} = \langle ll, rr \rangle\)
- \(T_2 = \{ e, i \}, \text{ and } G_{2_{pos}} = \langle ee, ii \rangle\)
Language example [cont.]

Corresponding FSA:
Languages in kist: outcomes

- Aksënova, Alëna and Sanket Deshmukh (2018)
  Formal Restrictions on Multiple Tiers
  Proceedings of SCiL-2018, ACL anthology, Salt Lake City.

- Aksënova, Alëna (2018)
  The Hitchhiker’s Guide to Harmony Interactions
  Poster at GLOW41, Budapest.

- McMullin, Kevin, Alëna Aksënova and Aniello De Santo (submitted)
  Learning Phonotactic Restrictions on Multiple Tiers

- Moradi, Sedigheh, Alëna Aksënova and Thomas Graf (submitted)
  The Computational Cost of Explicit Generalizations
Languages: local summary

- Subregular classes accommodate most linguistic patterns.
- They are learnable from positive data only.
- For every subregular pattern, it is possible to construct a FSA.

A Finite State Automaton detects whether a given string belongs to a certain class.

In order to re-write a string, one needs a Finite State Transducer.
Languages: local summary

- Subregular classes accommodate most linguistic patterns.
- They are learnable from positive data only.
- For every subregular pattern, it is possible to construct a FSA.

- A **Finite State Automaton** detects whether a given string belongs to a certain class.
- In order to re-write a string, one needs a **Finite State Transducer**.
Functions and string FSTs

- String transducers have been used for different tasks since 1960-s. (Schützenberger 1961)

- In linguistics, multiple string extending and rewriting operations are represented via transductions.
  - cat + s → cats
  - witch + s → witches

- Currently, one of the directions of research is to carve sub-classes of the whole class of subsequential transducers.
  (Chandlee 2014, i.a.)

Here, I only focus on subsequential – reading input symbol-by-symbol – transducers.
Functions and string FSTs

- String transducers have been used for different tasks since 1960-s. (Schützenberger 1961)
- In linguistics, multiple string extending and rewriting operations are represented via transductions.
  - $\text{cat} + s \mapsto \text{cats}$
  - $\text{witch} + s \mapsto \text{witches}$
- Currently, one of the directions of research is to carve sub-classes of the whole class of subsequential transducers. (Chandlee 2014, i.a.)

Here, I only focus on subsequential – reading input symbol-by-symbol – transducers.
String transducers have been used for different tasks since 1960-s. (Schützenberger 1961)

In linguistics, multiple string extending and rewriting operations are represented via transductions. cat + s → cats
witch + s → witches

Currently, one of the directions of research is to carve sub-classes of the whole class of subsequential transducers.
(Chandlee 2014, i.a.)

Here, I only focus on subsequential – reading input symbol-by-symbol – transducers.
There are numerous learners for transductions. Among them:

- **OSTIA**: sub subsequential transductions, cubic time, less data  
  (Oncina, García, and Vidal 1993; de la Higuera 2010)
- **SOSFIA**: sub subsequential transductions, linear time, more data  
  (Jardine et. al 2014)
- **ISLFLA**: ISL transductions, quadratic time  
  (Chandlee, Eyraud, and Heinz 2014)
- **OSLFIA**: OSL transductions, quadratic time  
  (Chandlee, Eyraud, and Heinz 2015)

- They work in polynomial time and need positive data only.
- Not all of them are implemented and used in practice!
Learners for string transducers

- There are numerous learners for transductions. Among them:
  - **OSTIA**: subsequential transductions, cubic time, less data
    (Oncina, García, and Vidal 1993; de la Higuera 2010)
  - **SOSFIA**: subsequential transductions, linear time, more data
    (Jardine et. al 2014)
  - **ISLFLA**: ISL transductions, quadratic time
    (Chandlee, Eyraud, and Heinz 2014)
  - **OSLFIA**: OSL transductions, quadratic time
    (Chandlee, Eyraud, and Heinz 2015)

- They work in polynomial time and need positive data only.
- Not all of them are implemented and used in practice!
Subsequential transducers in KIST

Implemented functionality:

- transducer’s template construction;
- learners;
- string rewriting;
- transducer trimming;
- onwarding the outputs.
Subsequential transducers in KIST

Implemented functionality:

- transducer’s template construction;
- learners;
- string rewriting;
- transducer trimming;
- onwarding the outputs.
String FSTs: an example of application

- **Tokenization** – separating words from sentence-level punctuations for further sentence processing.
  
  “Bob, Sue and Bill didn’t buy sugar-free coffee.”
  
  ⇨ “Bob, Sue and Bill didn’t buy sugar-free coffee.”

- **Challenges:**
  - Trying to avoid hard-coding the linguistic variety of contexts and punctuations (for example, Spanish ‘¿’)
  - Not all same symbols are treated in the same way:
    
    “Dogs bark.” ⇨ “Dogs bark.”
    “Mr. Bean” ⇨ “Mr. Bean”

- Existent tokenizers are language-specific and perform comparatively slow.
String FSTs: an example of application

- **Tokenization** – separating words from sentence-level punctuations for further sentence processing.
  
  “Bob, Sue and Bill didn’t buy sugar-free coffee.”
  
  ⟷ “Bob, Sue and Bill didn’t buy sugar-free coffee.”

- **Challenges:**
  
  - Trying to avoid hard-coding the linguistic variety of contexts and punctuations (for example, Spanish ‘¿’)
  
  - Not all same symbols are treated in the same way:
    
    “Dogs bark.” ⟷ “Dogs bark.”
    
    “Mr. Bean” ⟷ “Mr. Bean”

- Existent tokenizers are language-specific and perform comparatively slow.
String FSTs: an example of application

- **Tokenization** – separating words from sentence-level punctuations for further sentence processing.
  “Bob, Sue and Bill didn’t buy sugar-free coffee.”
  → “Bob, Sue and Bill didn’t buy sugar-free coffee.”

- **Challenges:**
  - Trying to avoid hard-coding the linguistic variety of contexts and punctuations (for example, Spanish ‘¿’)
  - Not all same symbols are treated in the same way:
    “Dogs bark.” → “Dogs bark .”
    “Mr. Bean” → “Mr. Bean ”

- Existent tokenizers are language-specific and perform comparatively slow.
String FSTs: an example of application [cont.]

Simplified FST for tokenization:
Functions in kist: current projects

- with Jeffrey Heinz and Kyle Gorman
  OSTIA-based tokenizer
  developing a low memory resource and high-accuracy tokenizer that avoids hard-coding language-specific information

- with Thomas Graf and Jeffrey Heinz
  Transduction learner for insufficient data
  creating a learning algorithm that allows to learn a class of non-equivalent transducers that can be inferred based on the insufficient input data
String transducers: local summary

- **Finite State Transducers** read a string as input, and return another string as output.

- Variety of different tasks can be performed via FSTs:
  - tokenization;
  - XML parsing;
  - multiple linguistic processes;
  - even machine translation!

- Current lines of research:
  - tree transducers;
  - one-to-many transductions;
  - learning of equivalent transducers for insufficient data;
  - ... and many others.
String transducers: local summary

- **Finite State Transducers** read a string as input, and return another string as output.

- Variety of different tasks can be performed via FSTs:
  - tokenization;
  - XML parsing;
  - multiple linguistic processes;
  - even machine translation!

- Current lines of research:
  - tree transducers;
  - one-to-many transductions;
  - learning of equivalent transducers for insufficient data;
  - ... and many others.
String transducers: local summary

- **Finite State Transducers** read a string as input, and return another string as output.

- Variety of different tasks can be performed via FSTs:
  - tokenization;
  - XML parsing;
  - multiple linguistic processes;
  - even machine translation!

- Current lines of research:
  - tree transducers;
  - one-to-many transductions;
  - learning of equivalent transducers for insufficient data;
  - . . . and many others.
**Timeline**

<table>
<thead>
<tr>
<th>Months</th>
<th>Goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>September</td>
<td>OSTIA (general subsequential learner)</td>
</tr>
<tr>
<td>October</td>
<td>OSLFIA (OSL transductions learner)</td>
</tr>
<tr>
<td>November</td>
<td>ISLFLA (ISL transductions learner)</td>
</tr>
<tr>
<td>December</td>
<td>MTSL learners, scanners, sample generators</td>
</tr>
<tr>
<td>January</td>
<td>SS-TSL learners, scanners, sample generators</td>
</tr>
<tr>
<td>February</td>
<td>Learner for regular languages (RPNI)</td>
</tr>
<tr>
<td>March</td>
<td></td>
</tr>
<tr>
<td>April</td>
<td>Testing, documentation and publishing</td>
</tr>
<tr>
<td>May</td>
<td></td>
</tr>
</tbody>
</table>
kist package is a subregular toolkit for linguistics and NLP.

Last year, I implemented subregular language tools. **Why?** They learn and generate formal languages of a required complexity.

This year, I am implementing transduction learners. **Why?** They extract different types of maps from input to output forms.

- **For researchers**, this toolkit facilitates the process of data analysis and generation, as well as assists in measuring complexities of already existent datasets.
- **For practitioners**, it opens new ways and perspectives of modeling natural language dependencies, and not only.
kist package is a subregular toolkit for linguistics and NLP.

Last year, I implemented subregular language tools. **Why?** They learn and generate formal languages of a required complexity.

This year, I am implementing transduction learners. **Why?** They extract different types of maps from input to output forms.

**For researchers**, this toolkit facilitates the process of data analysis and generation, as well as assists in measuring complexities of already existent datasets.

**For practitioners**, it opens new ways and perspectives of modeling natural language dependencies, and not only.
What I cannot create, I do not understand.

*Richard Feynman*

Thank you!
Aksënova, Alëna, Thomas Graf and Sedigheh Moradi (2016)
Morphotactics as Tier-Based Strictly Local Dependencies.
In Proceedings of SIGMorPhon 2016.

Avcu, Enes, Chihiro Shibata, and Jeffrey Heinz.
Subregular Complexity and Deep Learning.

Chandlee, Jane (2014)
Strictly Local Phonological Processes.

Chandlee, Jane, Rémi Eyraud and Jeffrey Heinz (2014)
Learning Strictly Local Subsequential Functions.
In TACL, 2:491-503.

Chandlee, Jane, Rémi Eyraud and Jeffrey Heinz (2015)
Output Strictly Local Functions.
In Proceedings of MoL 14, 112-125.

Graf, Thomas (2017)
The subregular complexity of monomorphemic quantifiers.
Ms., Stony Brook University.

Graf, Thomas and Jeffrey Heinz (2015)
Commonality in Disparity: The Computational View of Syntax and Phonology.
Heinz, Jeffrey (2010)  
Learning long-distance phonotactics.  

de la Higuera, Colin (2010)  
*Grammatical Inference: Learning Automata and Grammars.*  
Cambridge University Press.

Jardine, Adam, Jane Chandlee, Rémi Eyraud and Jeffrey Heinz (2014)  
Very efficient learning of structured classes of subsequential functions from positive data.  

McNaughton, Robert and Seymour Papert (1971)  
Counter-Free Automata.  

Oncina, José, Pedro García and Enrique Vidal (1993)  
Learning subsequential transducers for pattern recognition tasks.  

Rawal, Chetan, Herbert Tanner and Jeffrey Heinz (2011)  
(Sub)regular Robotic Languages.  

Schützenberger, Marcel-Paul (1961)  
A Remark on Finite Transducers.  