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`: **k**ist **i**mplementing **s**ubregular **t**oolkit

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`: **k**ist **i**mplementing **s**ubregular **t**oolkit

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`: **k**ist **i**mplementing **s**ubregular **t**oolkit

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

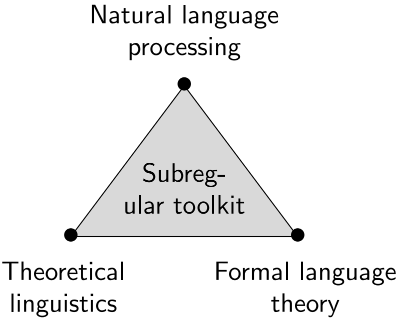
This subregular toolkit allows one to:

- 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.



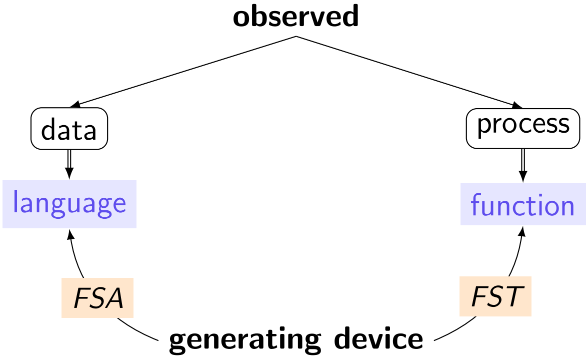
The importance of formalization

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

Languages vs. Functions



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.

What is done and what is left

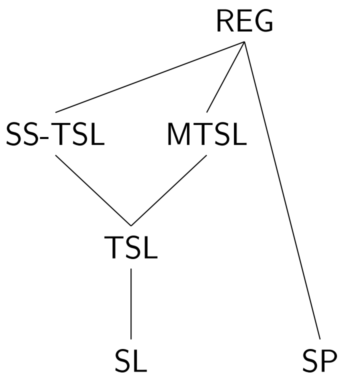
Last year:

- ☑ 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.

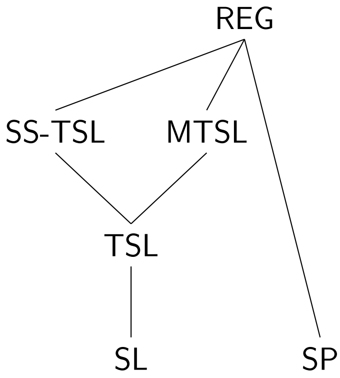
Languages and FSMs



Subregular hierarchy (simplified)

- 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!

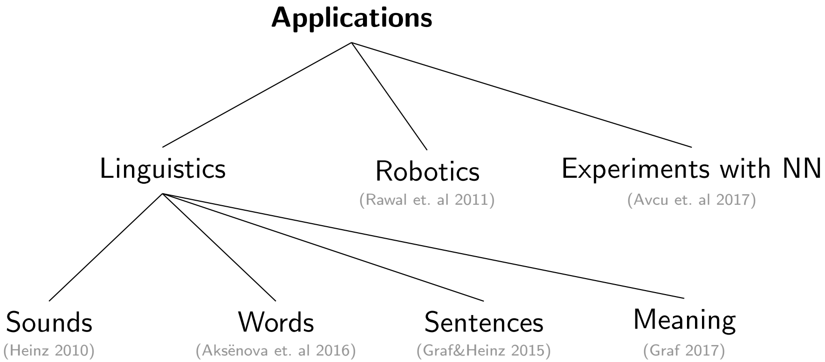
Languages and FSMs



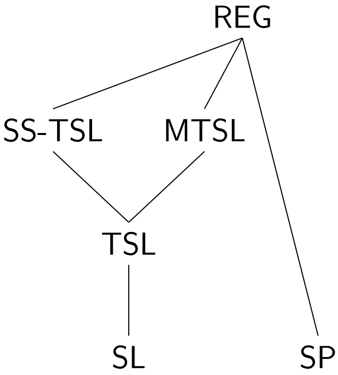
Subregular hierarchy (simplified)

- 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?



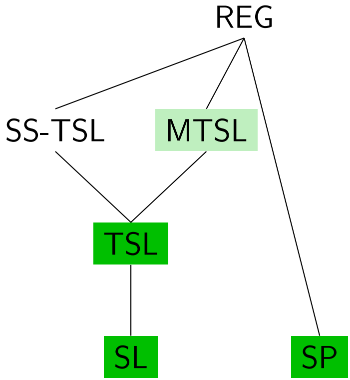
Subregular languages in KIST



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 $\leftrightarrow$ positive grammar translators;
- constructing corresponding FSA;
- trimming FSA.

Language example

- *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'

Language example

- *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'

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:
$(\mathtt{l},\mathtt{e})^+ \cup (\mathtt{l},\mathtt{i})^+ \cup (\mathtt{r},\mathtt{e})^+ \cup (\mathtt{r},\mathtt{i})^+$

- ^{ok}llliiiillliiii
- ^{ok}eeerreer
- ^{ok}lleeelle
- ^{ok}riiriirrr
- ¬liiirriii
- ¬leeelliii
- ...
- ...

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:
$(\mathtt{l},\mathtt{e})^+ \cup (\mathtt{l},\mathtt{i})^+ \cup (\mathtt{r},\mathtt{e})^+ \cup (\mathtt{r},\mathtt{i})^+$

- ^{ok}llliiillliiii
- ^{ok}eeerreer
- ^{ok}lleeelle
- ^{ok}riiriirrr
- $\neg$liiirriii
- $\neg$leeellliii
- ...
- ...

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:
$(\mathtt{l},\mathtt{e})^{+} \cup (\mathtt{l},\mathtt{i})^{+} \cup (\mathtt{r},\mathtt{e})^{+} \cup (\mathtt{r},\mathtt{i})^{+}$

- ^{ok}llliiilllliiii
- ^{ok}eeerreer
- ^{ok}lleeelle
- ^{ok}riiriirrr
- ¬liiirriii
- ¬leeelliii
- ...
- ...

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.]

$$\texttt{(l,e)}^+ \cup \texttt{(l,i)}^+ \cup \texttt{(r,e)}^+ \cup \texttt{(r,i)}^+$$

- *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.]

$$(\mathtt{l},\mathtt{e})^+ \cup (\mathtt{l},\mathtt{i})^+ \cup (\mathtt{r},\mathtt{e})^+ \cup (\mathtt{r},\mathtt{i})^+$$

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

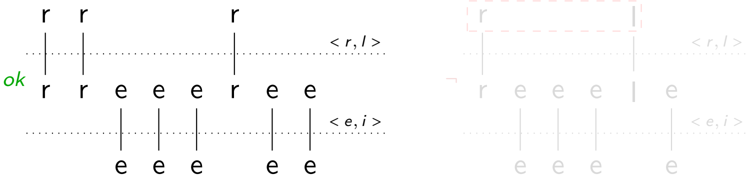
- $T_1 = \{l, r\}$, and $G_{1_{pos}} = \langle ll, rr \rangle$
- $T_2 = \{e, i\}$, and $G_{2_{pos}} = \langle ee, ii \rangle$

Language example [cont.]

$$(\mathtt{l},\mathtt{e})^{+} \cup (\mathtt{l},\mathtt{i})^{+} \cup (\mathtt{r},\mathtt{e})^{+} \cup (\mathtt{r},\mathtt{i})^{+}$$

- *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 $G_{2_{pos}} = \langle ee, ii \rangle$

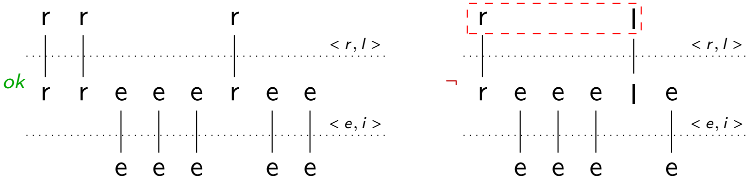


Language example [cont.]

$$\texttt{(l,e)}^{+} \cup \texttt{(l,i)}^{+} \cup \texttt{(r,e)}^{+} \cup \texttt{(r,i)}^{+}$$

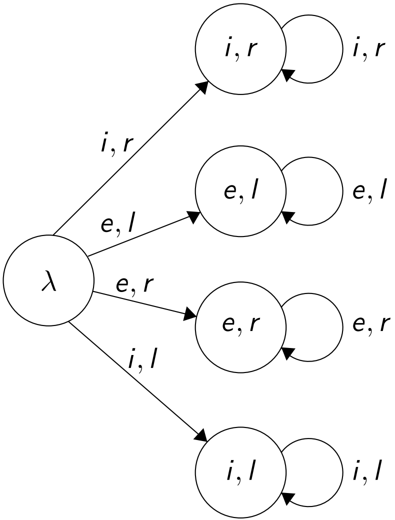
- *Complexity:* MTSL (multiple tier-based strictly local)
- *Meaning:* there are several sets of items involved in long-distance dependency.

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



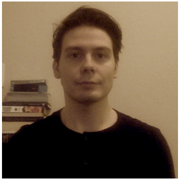
Language example [cont.]

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


Aniello De Santo


Sanket Deshmukh


Kevin McMullin


Sedigheh Moradi

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.

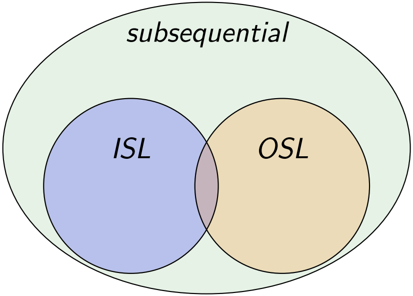
- A **Finite State Automaton** detects whether a given string belongs to a certain class.
- In order to re-write a string, one needs a **Finite State Transducer**.

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**.

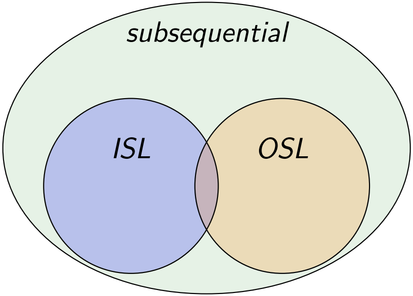
Functions and string FSTs



- 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.

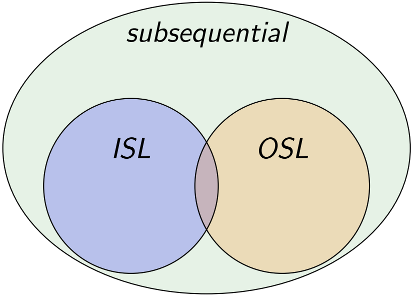
Functions and string FSTs



- 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.
  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.
  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.

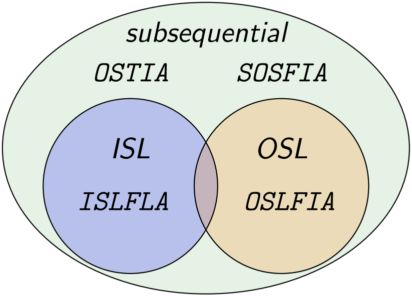
Learners for string transducers

- 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!

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!

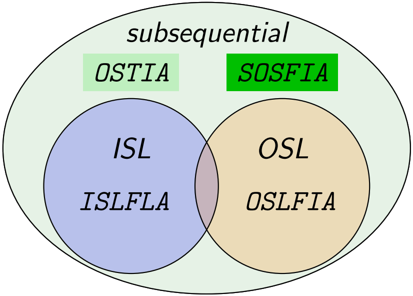
Subsequential transducers in KIST



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."

  $\mapsto$ " 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." $\mapsto$ " Dogs bark ."

    "Mr. Bean" $\mapsto$ " 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."

  $\mapsto$ " 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." $\mapsto$ " Dogs bark ."
    "Mr. Bean" $\mapsto$ " Mr. Bean "
- Existent tokenizers are language-specific and perform comparatively slow.

String FSTs: an example of application

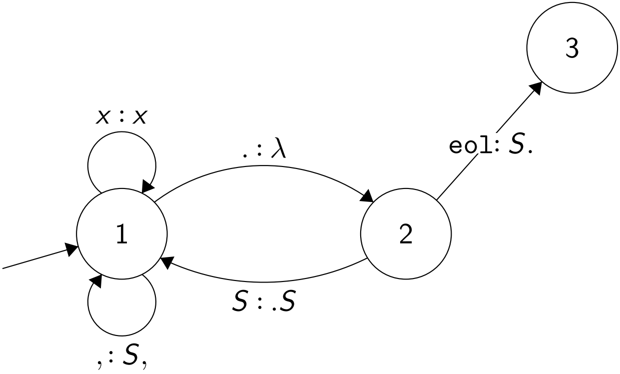
- **Tokenization** – separating words from sentence-level punctuations for further sentence processing.

  "Bob, Sue and Bill didn't buy sugar-free coffee."

  $\mapsto$ " 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." $\mapsto$ " Dogs bark ."
    "Mr. Bean" $\mapsto$ " 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

Kyle Gorman

- 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

Months	**Goals**
September	OSTIA (general subsequential learner)
October	OSLFIA (OSL transductions learner)
November	ISLFLA (ISL transductions learner)
December	MTSL learners, scanners, sample generators
January	SS-TSL learners, scanners, sample generators
February	Learner for regular languages (RPNI)
March	
April	Testing, documentation and publishing
May	

Conclusion

- `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.

Conclusion

- `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!

References I

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.
CLASP Papers in Computational Linguistics: Proceedings of LaML 2017.

Chandlee, Jane (2014)
Strictly Local Phonological Processes.
PhD thesis, University of Delaware.

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.
Slides of a talk given at GLOW 2015. Paris, France.

References II

Heinz, Jeffrey (2010)
Learning long-distance phonotactics.
Linguistic Inquiry 41(4): 623 – 661.

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.
In *JMLR: Workshop and Conference Proceedings*, 34:94-108.

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

Oncina, José, Pedro García and Enrique Vidal (1993)
Learning subsequential transducers for pattern recognition tasks.
In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:448-458.

Rawal, Chetan, Herbert Tanner and Jeffrey Heinz (2011)
(Sub)regular Robotic Languages.
In *Proceedings of IEEE Mediterranean Conference on Control and Automation*, 321–326.

Schützenberger, Marcel-Paul (1961)
A Remark on Finite Transducers.
In *Information and Control*, 4(2-3): 185–196.