Математические модели в лингвистике

Вычислительная морфология.

Мати Пентус, Александр Пиперски, Алексей Сорокин

МГУ им. М. В. Ломоносова, межфакультетский курс,
осенний семестр 2017–2018 учебного года, 18 октября

Конечные преобразователи

Неформально, конечный преобразователь — это автомат с добавленными на рёбра выходными символами.

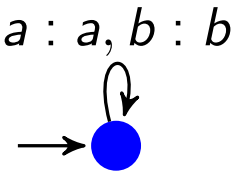
Пусть Σ, Γ — конечные алфавиты.

Определение конечного преобразователя

Конечный преобразователь: кортеж $M = \langle Q, \Sigma, \Gamma, \Delta, q_0, F \rangle$, где

- Q — конечное множество состояний
- $\Delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \times Q$ — конечное множество переходов
- $q_0 \in Q$ — стартовое состояние
- $F \subseteq Q$ — завершающие состояния.

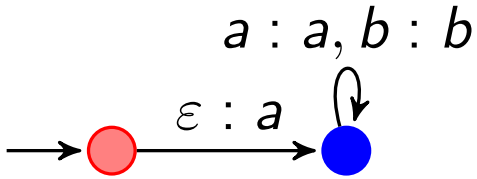
Простейший преобразователь — тождественный (алфавит a, b):



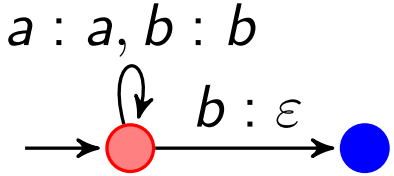
Конечные автоматы можно понимать как преобразователи, возвращающие свой вход (и проверяющие, что вход принадлежит нужному множеству).

Конечные преобразователи: примеры

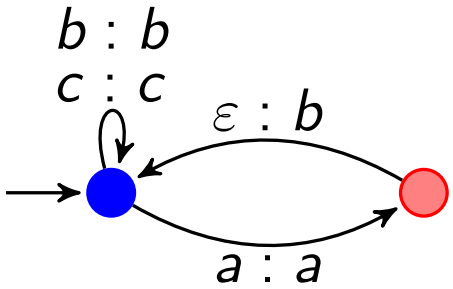
- Добавляет a в начале слова:



- Удаляет конечную b в тех словах, где она есть, и отвергает остальные:

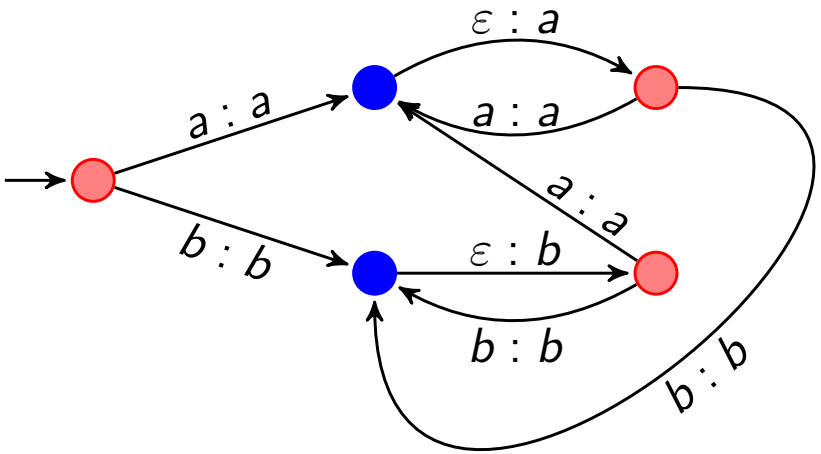


- После каждой a добавляет b:

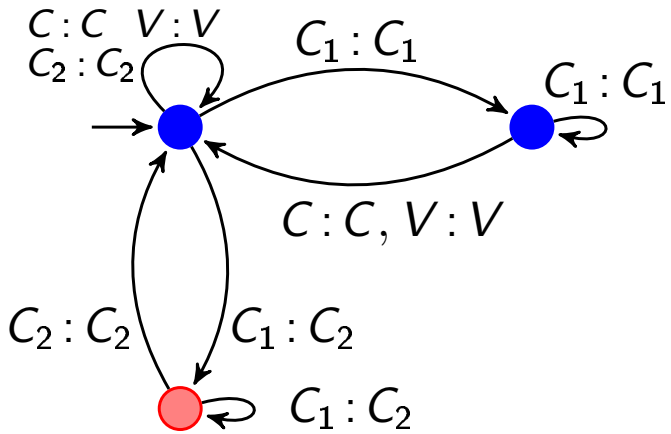


Конечные преобразователи: примеры

- Удваивает все буквы, кроме последней:



- Ретро-ассимилирует C_1 в C_2 (в последовательности C_1, идущей перед C_2, все буквы заменяются на C_2)



Свойства конечных преобразователей

- Замкнутость относительно операций:

Операция	Автоматы	Преобразователи
Конкатенация	Да	Да
Итерация	Да	Да
Объединение	Да	Да
Пересечение	Да	Нет
Дополнение	Да	Нет

Свойства конечных преобразователей

- Конечные преобразования также замкнуты относительно
  - Композиции ($\circ$).
  - Приоритетного объединения ($\cup_p$).

$$(T_1 \cup_p T_2)(u) = \begin{cases} T_1(u), & T_1(u) \neq \varnothing, \\ T_2(u), & T_1(u) = \varnothing. \end{cases}$$

  - Обращения ($(\cdot)^{-1}$): $\phi^{-1} = \{\langle y, x \rangle \mid \langle x, y \rangle \in \phi\}$.
- Применения операций:
  - Композиция: последовательное применение преобразований,
  - Обращение: переход от синтеза к анализу и наоборот,
  - Приоритетное объединение: отдельная обработка нерегулярных форм.

Множественное число существительного в английском

Пример.

Опишите преобразователь, преобразующий форму единственного числа существительного в форму множественного для английского языка.

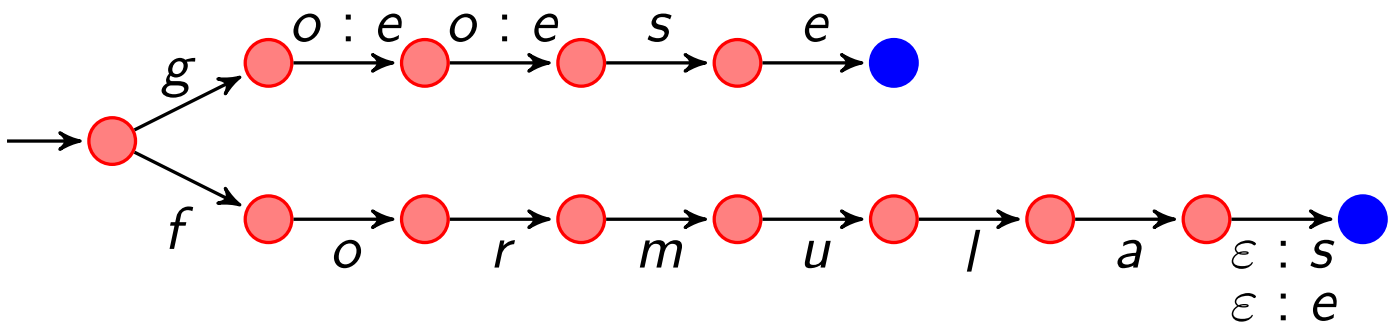
- *torch* $\leftrightarrows$ *torches*
- *monarch* $\leftrightarrows$ *monarchs*
- *ally* $\leftrightarrows$ *allies*
- *play* $\leftrightarrows$ *plays*
- *goose* $\leftrightarrows$ *geese*
- *formula* $\leftrightarrows$ *formulas/formulae*

Пример: множественное число в английском

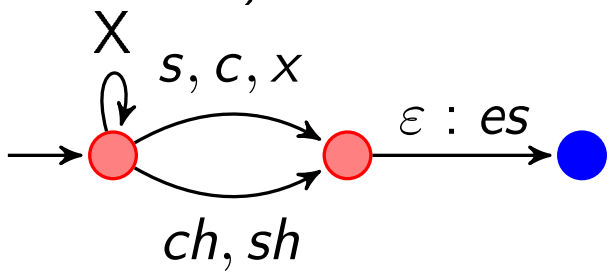
Пример.

Описать преобразователь, строящий форму множественного числа существительного в английском фзыке.

- Отдельный преобразователь T_{exc} для исключений:

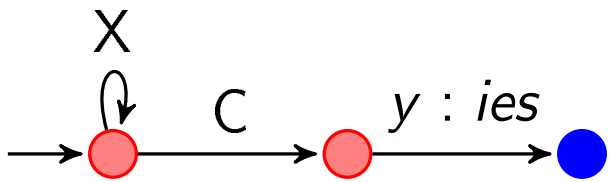


- T_{sib}: добавляет *-es* после финальной шипящей (X — произвольный символ):



Пример: множественное число в английском

- Преобазователь T_{exc} для исключений.
- Преобразователь T_{sib} для добавления *es*.
- Преобразователь T_{Cy} для замены *y* на *-ies* после согласной.



- T_s — добавляет s в конце.
- $T_{exc,sib}$ — добавляет s к исключениям на *-arch* и отвергает остальные слова (для *monarchs*, *tetrarchs*, . . .).
- Общее решение:

$$T_{exc} \cup_p T_{exc,sib} \cup_p T_{sib} \cup_p T_{Cy} \cup_p T_s$$

Пример: снова множественное число

- Наша модель для английского лингвистически неадекватна.
- Нет отдельных окончаний *-es*, *-ies*, *-s*, они алломорфы окончания *-s*.
- Нужно сначала присоединить прототипическое окончание, потом моделировать контекстные явления.
- Преобразователи для правил:
  - T_s: добавить $!s$ в конце слова (! — маркер границы) $\varepsilon \rightarrow !s \,||\, \_\$$ ($ — маркер конца слова).
  - T_{sib}: добавить e после ! и после шипящей $\varepsilon \rightarrow e \,||\, (s|z|x|sh|ch)\_!$.
  - T_y: заменить y на ie перед маркером границы и после согласной $y \rightarrow ie \,||\, C\_!$.
  - $T_{exc,sib}$: ничего не делать со словами с *arch* на конце (возможно, не все такие слова подойдут). $?^* V ?^* arch!s$
  - T_c: удалить маркер морфемной границы $! \rightarrow \varepsilon$.
- Финальный преобразователь строится с помощью композиции:

$$T_{exc} \cup_p (T_s \circ (T_{exc,sib} \cup_p T_{sib}) \circ T_y \circ T_c)$$

Примеры вычисления

- Исключения: *mouse*$\mapsto$*mice*, *cactus*$\mapsto$*cactuses, cacti*
  — обрабатывается в T_{exc} и не идёт в основную ветку.
- Регулярные формы обрабатываются в ветке
  $T_s \circ (T_{exc,sib} \cup_p T_{sib}) \circ T_y \circ T_c$:
  - *day* $\rightarrow$ *day*!*s* $\rightarrow$ *day*!*s* $\rightarrow$ *day*!*s* $\rightarrow$ *days*
  - *rally* $\rightarrow$ *rally*!*s* $\rightarrow$ *rally*!*s* $\rightarrow$ *rallie*!*s* $\rightarrow$ *rallies*
  - *witch* $\rightarrow$ *witch*!*s* $\rightarrow$ *witche*!*s* $\rightarrow$ *witche*!*s* $\rightarrow$ *witches*
- Частичное исключение *monarch* обрабатывается веткой
  $T_s \circ T_{exc,sib} \circ T_y \circ T_c$:
  monarch $\rightarrow$ *monarch*!*s* $\rightarrow$ *monarch*!*s* $\rightarrow$ *monarch*!*s* $\rightarrow$ *monarchs*.

Инфинитив пассивного залога в турецком

Инфинитив пассивного залога

Построить конечный преобразователь, преобразующий турецкий глагольный инфинитив в инфинитив пассивного залога.

- Пассив образуется вставкой суффикса перед *-mek/-mak*.
- Суффикс пассива: *-n* после гласной, *-In* после *l* и *-Il* иначе.
- I: *ı* после *a*, *ı*; *u* после *u*, *o*; *i* после *e*, *i*; *ü* после *ü*, *ö*.

Инфинитив	Инфинитив пассива
varmak “*прибывать*”	*varılmak* “*достичь*”
silmek “*удалять*”	*silinmek* “*удаляться кем-либо*”
büyümek “*возрастать*”	*büyünmek* “*увеличиваться*”
durmak “*останавливаться*”	*durulmak* “*задерживаться*”
bilmek “*знать*”	*bilinmek* “*быть известным*”

Инфинитив пассивного залога в турецком

Инфинитив пассивного залога

Построить конечный преобразователь, преобразующий турецкий глагольный инфинитив в инфинитив пассивного залога.

- Пассив образуется вставкой суффикса перед *-mek/-mak*.
- Суффикс пассива: *-n* после гласной, *-In* после *l* и *-Il* иначе.
- I: *ı* после *a*, *ı*; *u* после *u*, *o*; *i* после *e*, *i*; *ü* после *ü*, *ö*.

- T_{mark}: вставить ! перед *-mak/-mek*: $\varepsilon \rightarrow ! \,||\, \_\, m(a|e)k\$$.
- Заменить маркер подходящим суффиксом:
  - *-n* после гласной (T_V): $! \rightarrow n \,||\, V \_\, \$$,
  - *-In* после *l* (T_l): $! \rightarrow \mathrm{I}n \,||\, l \_\, \$$,
  - *-Il* по умолчанию (T_{def}): $! \rightarrow \mathrm{I}l \,||\, \_$,
- Соединить всё вместе $T_{suf} = T_V \circ T_l \circ T_{def}$.

Инфинитив пассивного залога в турецком

Инфинитив пассивного залога

Построить конечный преобразователь, преобразующий турецкий глагольный инфинитив в инфинитив пассивного залога.

- Пассив образуется вставкой суффикса перед *-mek/-mak*.
- Суффикс пассива: *-n* после гласной, *-In* после *l* и *-Il* иначе.
- I: *ı* после *a*, *ı*; *u* после *u*, *o*; *i* после *e*, *i*; *ü* после *ü*, *ö*.

- T_{mark} — вставляет ! перед *-mak/-mek*.
- T_{suf} заменяет маркер на подходящий суффикс.
- T_{fill} заполняет гласную суффикса: $T_{fill} = T_\imath \circ T_u \circ T_i \circ T_U$, where
  - $T_\imath$ проверяет условие для $\imath$: $\mathrm{I} \to \imath \,||\, (a|\imath)C^* \_$.
  - T_u для u: $\mathrm{I} \to u \,||\, (u|o)C^* \_$.
  - T_i для i: $\mathrm{I} \to i \,||\, (e|i)C^* \_$.
  - T_U для $ü$: $\mathrm{I} \to ü \,||\, (ü|ö)C^* \_$.
- Финальный ответ:

$$T_{mark} \circ T_{suf} \circ T_{fill}$$

Глагольные формы в языке йоулумни (америндская семья)

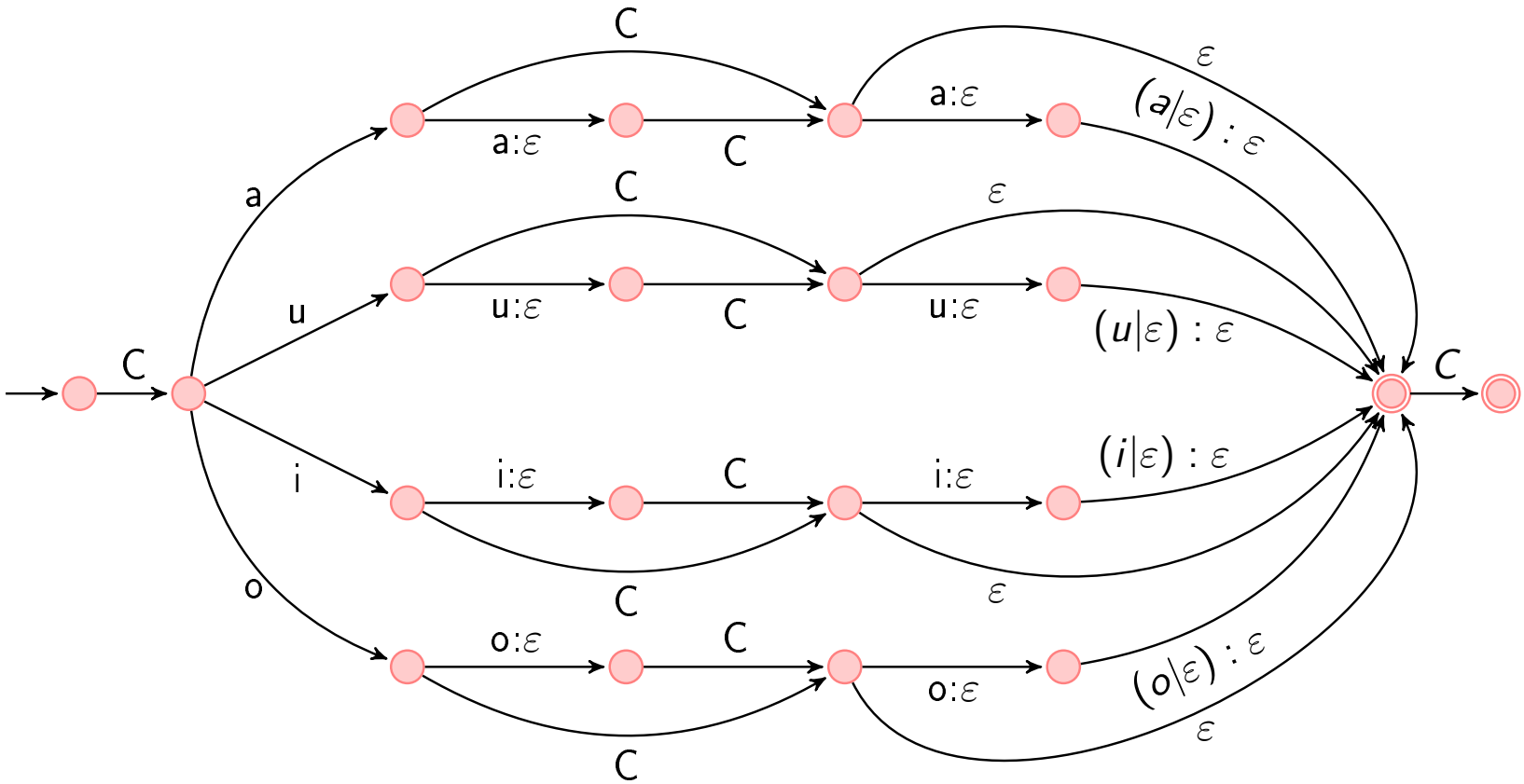
основа	герундий	дуратив
caw “кричать”	caw-inay	cawaa-ʔaa-n
cuum “разрушать”	cum-inay	cumuu-ʔaa-n
hoyoo “называть”	hoy-inay	hoyoo-ʔaa-n
diiyl “охранять”	diyl-inay	diyiil-ʔaa-n
ʔilk “петь”	ʔilk-inay	ʔiliik-ʔaa-n
hiwiit “гулять”	hiwt-inay	hiwiit-ʔaa-n

Глагольные формы в языке йоулумни

Если основа имела вид $\alpha_1 V(V)\alpha_2(V)(V)\alpha_3$, где $\alpha_1, \alpha_2 \in C$, $\alpha_3 \in \{C, \varepsilon\}$, то основа герундия имеет вид $\alpha_1 V \alpha_2 \alpha_3$, а основа дуратива — $\alpha_1 V \alpha_2 VV \alpha_3$.

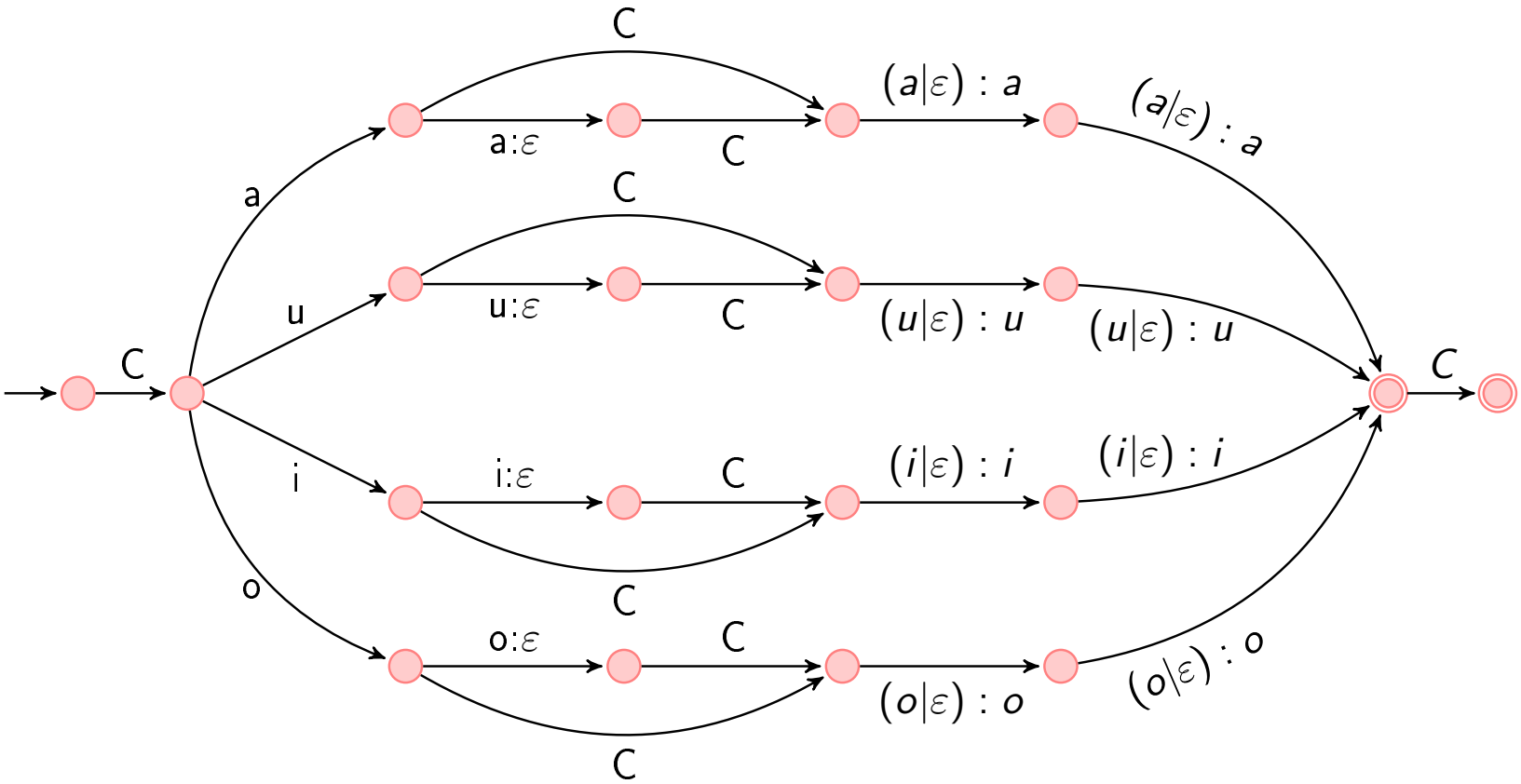
Преобразователи для глагольных форм в языке йоулумни

- Основа герундия:



Преобразователи для глагольных форм в языке йоулумни

- Основа дуратива:



FOMA: компилятор конечных преобразователей.

- FOMA — компилятор конечных преобразователей.
- Разработана Мансом Халденом в 2009–2015гг., последняя официальная версия 0.9.18 — июнь 2015.
- Стабильная версия: `https://code.google.com/archive/p/foma/`,
- Текущая версия `https://github.com/mhulden/foma/`.
- Программа с открытым кодом, написана C++, для отдельных операций есть интерфейс в Python.
- Основное применения: компиляция контекстных правил в конечные преобразователи.
- Можно использовать для операций с конечными автоматами.
- утилита flookup позволяет сохранять конечные преобразователи в двоичном формате.

FOMA: basic usage

Базовые операции: компиляция контекстных правил.

```
foma[0]: ##replace all a by b
foma[0]: regex a -> b || _ ;
374 bytes. 1 state, 3 arcs, Cyclic.
foma[1]: net
Sigma: ? @ a b
Size: 2.
Net: E20E6CF
Flags: deterministic pruned minimized epsilon_free
Arity: 2
Sfs0:   <a:b> -> fs0, b -> fs0, @ -> fs0.
foma[1]:
```

FOMA: basic usage

Базовые операции: задание преобразователей и их применение к строкам.

```
foma[0]: ##replace all a by b
foma[0]: regex a -> b || _ ;
374 bytes. 1 state, 3 arcs, Cyclic.
foma[1]: down
apply down> bcaba
bcbbb
apply down> bbb
bbb
apply down>
foma[1]: up
apply up> aba
???
apply up> cbdb
cada
cadb
cbda
cbdb
apply up> cdc
```

FOMA: basic usage

Множественное число для заканчивающихся на *y* существительных:

```
foma[0]: ## filter y-ending words
foma[0]: define yFinal ?* y ;
redefined yFinal: 321 bytes. 2 states, 4 arcs, Cyclic.
foma[0]: ## Vowel+y
foma[0]: define Vowel [ a | e | i | o | u ];
redefined Vowel: 413 bytes. 2 states, 5 arcs, 5 paths.
foma[0]: define yVowel [..] -> s || [ .#. | Vowel ] y _ .#. ; ## simply append s after Vowel+y
redefined yVowel: 872 bytes. 4 states, 25 arcs, Cyclic.
foma[0]: define yCons y -> i e s || \Vowel _ .#. ;
redefined yCons: 920 bytes. 6 states, 28 arcs, Cyclic.
foma[0]: ## combine the variants for vowels and consonants
foma[0]: define yChange yFinal .o. yVowel .o. yCons ;
redefined yChange: 936 bytes. 6 states, 29 arcs, Cyclic.
foma[0]: push yChange
936 bytes. 6 states, 29 arcs, Cyclic.
foma[1]: down
apply down> valley
valleys
apply down> ally
allies
apply down> y
ys
apply down> tray
trays
apply down> granny
grannies
```

FOMA: операции с автоматами

Операция	Обозначение
Конкатенация X, Y	XY
Пересечение X, Y	$X\&Y$
Объединение X, Y	$X\|Y$
Разность X, Y	$X - Y$
Итерация X	X^*
Положительная итерация X	X^+
Отрицание X	$\backslash X$
Контекстное ограничение (X возможен только между Y и Z)	$X \rightarrow Y \_ Z$

Операции с автоматами в FOMA

FOMA: операции с автоматами

Операция	Обозначение
Контекстная замена (Замена X на Y в контексте U_V)	$X \rightarrow Y \| \| U\_V$
Композиция X, Y	$X.\mathrm{o}.Y$
Приоритетное объединение X, Y	$X.\mathrm{P}.Y$
Декартово объединение X, Y	$X : Y$
Область определения X	$X.u$
Область значений X	$X.l$
Обратное преобразование к X	$X.i$
Множественные контексты	$X \rightarrow Y \| \| U1\_V1, U2\_V2$
Множественная замена	$X1 \rightarrow Y1, X2 \rightarrow Y2 \| \| U\_V$

Операции с преобразователями в FOMA

FOMA: применение преобразователей

Операция	Обозначение
Сохранить пр-ль в переменную	**define** ⟨var_name⟩ ⟨expression⟩
Поместить пр-ль в стек	**push** ⟨var_name⟩
Поместить выражение в стек	**regex** ⟨expression⟩
Применить верхний пр-ль в стеке в "прямом" направлении	**down** (apply down)
Применить верхний пр-ль в стеке в обратном направлении	**up** (apply up)
Очистить стек	**clear**
Прочитать файл с лексиконом и сохранить пр-ль в переменную	**read** ⟨filename⟩ **define** ⟨var_name⟩
сохранить пр-ль в двоичный файл	**save stack** ⟨filename⟩

Применение преобразователей в FOMA

FOMA: внешнее использование и документация

- Документация: `https://code.google.com/archive/p/foma/wikis`.
- Описание операций: `https://code.google.com/archive/p/foma/wikis/RegularExpressionReference.wiki`.
- Преобразователи можно сохранять в двоичном формате командой **save stack**. Потом их можно запускать из командной строки с помощью утилиты **flookup**.
- Основной способ использования:

  flookup -i -x -w ⟨binary_file⟩ < ⟨input_file⟩ (> ⟨output_file⟩)

- Применяет преобразователь из ⟨binary_file⟩ ко всем строкам в ⟨input_file⟩ и печатает результат (в выходной поток или ⟨output_file⟩).
- Без ключа -x key также печатается входное слово.
- Документация: `https://code.google.com/archive/p/foma/wikis/FlookupDocumentation.wiki`.

Множественное число существительного в английском

```
### english.foma ###
read lexc irregular.lexc
define IrregularNounPlural;

define Vowel [ a | i | e | o | u | y ];
define Consonant [ b | c | d | f | g | h | j | k | l | m | n | p | q | r | s | t | v | w | x | z ];
define Letter [Vowel | Consonant];
define Word [ Letter ]+;
define NounMark "+N";
define NounNumber "+Sg" | "+Pl";
define Noun Word NounMark NounNumber;

define NounAffixation "+N" "+Sg" -> "" || _ .#., "+N" "+Pl" -> "!" s || _ .#.;
define Sibilant [ x | s | z | c h | s h ];
define sibException ?* Vowel ?* a r c h "!" s ;
define eInsertion [..] -> e || Sibilant _ "!" s .#.;
define checkSibilant [ sibException .P. eInsertion ];
define yReplacement y -> i e || Consonant _ "!" s .#.;
define Cleanup "!" -> "" || _ ;
define RegularNoun [ NounAffixation .o. yReplacement .o. checkSibilant .o. Cleanup ] ;
define Grammar Noun .o. [ IrregularNounPlural .P. RegularNoun ];
push Grammar
```

Инфинитив пассивного залога в турецком

Инфинитив пассивного залога

Построить конечный преобразователь, преобразующий турецкий глагольный инфинитив в инфинитив пассивного залога.

- Пассив образуется вставкой суффикса перед *-mek/-mak*.
- Суффикс пассива: *-n* после гласной, *-In* после *l* и *-Il* иначе.
- I: *ı* после *a*, *ı*; *u* после *u*, *o*; *i* после *e*, *i*; *ü* после *ü*, *ö*.

Инфинитив	Инфинитив пассива
varmak “прибывать”	*varılmak*
silmek “удалять”	*silinmek*
büyümek “увеличивать”	*büyünmek*
durmak “останавливать”	*durulmak*
bilmek “знать”	*bilinmak*

Инфинитив пассивного залога в турецком

```
# symbol classes
define HardStraightVowel a | I ;
define HardRoundVowel o | u ;
define SoftStraightVowel e | i ;
define SoftRoundVowel O | U ;
define HardVowel HardStraightVowel | HardRoundVowel ;
define SoftVowel SoftStraightVowel | SoftRoundVowel ;
define Vowel HardVowel | SoftVowel ;
define Consonant b | c | C | d | f | g | G | h | j | k | l | m | n | p | r | s | S | t | v | y | z ;
define Letter Consonant | Vowel ;

# contexts for stem
define LastVowelHard HardVowel Consonant* ;
define LastVowelSoft SoftVowel Consonant* ;
define LastVowelHardRound HardRoundVowel Consonant* ;
define LastVowelHardStraight HardStraightVowel Consonant* ;
define LastVowelSoftRound SoftRoundVowel Consonant* ;
define LastVowelSoftStraight SoftStraightVowel Consonant* ;
```

Инфинитив пассивного залога в турецком

```
# infinitive vowel check
define Stem Letter* Vowel Letter* ;
define InfinitiveSuffixInsertion [..] -> E || _ .#. ;
define InfinitiveSuffix [ E -> m a k || HardVowel Consonant* _ .#. ] .o. [ E -> m e k ||
       SoftVowel Consonant* _ .#. ] ;
define SuffixTransform Stem .o. InfinitiveSuffixInsertion .o. InfinitiveSuffix ;
define Infinitive SuffixTransform.l ;
define Input Infinitive "+Pass";

# suffix insertion
define MarkerInsertion [..] -> "!" || _ m [ a | e ] k "+Pass" .#. ;
define MarkerAfterVowel "!" -> l || Vowel _ ;
define MarkerAfterL "!" -> A n || l _ ;
define MarkerAfterAll "!" -> A l || _ ;
define MarkerReplacement MarkerAfterVowel .o. MarkerAfterL .o. MarkerAfterAll ;

# combining all
define VowelFill [ A -> I || LastVowelHardStraight _ ] .o. [ A -> i ||
      LastVowelSoftStraight _ ] .o. [ A -> u || LastVowelHardRound _ ].o. [ A -> U ||
      LastVowelSoftRound _ ] ;
define Cleanup "+Pass" -> "" ;
define Grammar Input .o. MarkerInsertion .o. MarkerReplacement .o. VowelFill ;

push Grammar
```

Глагольные формы в йоулумне

stem	gerund	durative
caw “кричать”	caw-inay	cawaa-ʔaa-n
cuum “разрушать”	cum-inay	cumuu-ʔaa-n
hoyoo “называть”	hoy-inay	hoyoo-ʔaa-n
diiyl “охранять”	diyl-inay	diyiil-ʔaa-n
ʔilk “петь”	ʔilk-inay	ʔiliik-ʔaa-n
hiwiit “гулять”	hiwt-inay	hiwiit-ʔaa-n

Глагольные формы в йоулумне (американская семья)

Для основы $\alpha_1 V(V)\alpha_2(V)(V)\alpha_3$, где $\alpha_1, \alpha_2 \in C$, $\alpha_3 \in \{C, \varepsilon\}$:

- Основа герундия равна $\alpha_1 V \alpha_2 \alpha_3$,
- а основа дуратива $\alpha_1 V \alpha_2 VV \alpha_3$.

Глагольные формы в йоулумне

- Преобразователь для йоулумне легко строится вручную.
- Можно ли это сделать в FOMA?
- Первый шаг: задать изменения контекстными правилами.
- Герундий: удалить все гласные, кроме первой слева.
- Левый контекст: $C^*V(C|V)^*$.
- Дуратив:
  - Удалить вторую гласную в первом слоге (левый контекст $\hat{}C^+V$).
  - Вставить во второй слог удвоенную гласную из первого.
- Проверка равенства гласных для дуратива:
  - Вставить все возможные пары гласных (*aa*, *ii*, *oo*, *uu*).
  - Проверить гармонию гласных, перечислив все варианты вида $C^+xC^+x^+C^*$ где x — произвольный гласный.

Двухуровневая морфология

- Морфология с конечным числом состояний основана на конкатенации.
- Идеал: агглютинативные языки (турецкий, финский, etc.):

evlerinizdin	*ev-ler-iniz-din*
“*из ваших домов*”	“*дом*”+Pl+“*ваш*”+“*из*”

- Тяжело описывать:
  - Регулярные исключения, обилие словоизменительных моделей.
  - Нерегулярные исключения.
  - Фузия.
  - Неконкатенативная морфология (арабский, другие семитские языки).
- Не описывается моделью: неограниченная редупликация.

Регулярные исключения

- Глагольное спряжение в испанском:

correr “бежать”	$\mapsto$	corro “(я) бегу”
escribir “бежать”	$\mapsto$	escribo “(я) пишу”
hablar “говорить”	$\mapsto$	escribo “(я) говорю”
dormir “спать”	$\mapsto$	duermo “(я) сплю”
mentir “лгать”	$\mapsto$	miento “(я) лгу”
poder “мочь”	$\mapsto$	puedo “(я) могу”
vestir “одеваться”	$\mapsto$	visto “(я) одеваюсь”
soñar “мечтать”	$\mapsto$	sueno “(я) мечтаю”
pedir “просить”	$\mapsto$	pido “(я) прошу”

- Предсказать гласную в корне на 100% нельзя (нужно заучивать)

Русская глагольная морфология

Для русского всё ещё хуже:

играть	$\mapsto$	*играет*
писать	$\mapsto$	*пишет*
спать	$\mapsto$	*спит*
греть	$\mapsto$	*греет*
хрипеть	$\mapsto$	*хрипит*
петь	$\mapsto$	*поёт*
целовать	$\mapsto$	*целует*
хохотать	$\mapsto$	*хохочет*

Фузия: испанский глагол

- Фузия — явление на стыке морфем, при котором невозможно провести чёткую границу между ними. (*военный* + *начальник* $\mapsto$ *военачальник*, *чихать* $\mapsto$ *чихнуть*, но *пинать* $\mapsto$ *пнуть*)
- Немного испанских чередований:

Инфинитив	+1+Sg	герундий
partir	parto	partiendo
imbuir	imbuyo	imbuyendo
destruir	destruyo	destruyendo
delinquir	delinco	delinquiendo
distinguir	distingo	distinguiendo
coger	cojo	cogiendo
elegir	elijo	elegiendo
agradecer	agradezco	agradeciendo
mecer	mezo	meciendo

- Происходит фузия между основой и первым гласным окончания.
- Нужно много “фонетических” правил.

Трансфиксация: глагольные формы в арабском

- До этого момента морфемная структура была линейной.
- Недостаточно для семитских языков (e.g. Arabic):

kataba	“(он) писал+Perf”
kattabat	“(она много) писал+Perf”
yaktubu	“(это+Masc) было написано+Imp”
takattibu	“(это+Fem) было (интенсивно) написано+Imp”

- Корень *k-t-b* состоит из согласных (обычно 3).
- Гласные в основном отражают морфологию.
- Разным глаголам соответствуют разные трансфиксы:

marida	“(он стал) больным+Perf”
marradat	“(она стала сильно) больной+Perf”
yamradu	“(его) сделали больным+Imp”
tamarridu	“(её) сделали (сильно) больной+Imp”

Арабский: иллюстрационный пример

- Постановка задачи:

$$\langle \text{stem} \rangle \langle \text{Type} \rangle \langle \text{Voice} \rangle \langle \text{Aspect} \rangle \langle \text{Person} \rangle \langle \text{Gender} \rangle \mapsto \langle \text{wordForm} \rangle$$

- Возможные значения:
  - $\langle \text{Type (порода)} \rangle \in \{I, II (\text{интенсив})\}$,
  - $\langle \text{Voice} \rangle \in \{\text{Act}, \text{Pass}\}$,
  - $\langle \text{Aspect} \rangle \in \{\text{Perf}, \text{Imperf}\}$,
  - $\langle \text{Person} \rangle \in \{3\}$,
  - $\langle \text{Gender} \rangle \in \{\text{M}, \text{F}\}$.
- 16 вариантов.
- Моделируем лишь один словоизменительный класс (*KTB* “писать”).

Глагольное спряжение в арабском

- Словообразование в арабском (конспект А. А. Зализняка):
- Варианты основы:

Порода	Модель	Пример
I (базовая)	K-T-B	kataba “писать”
II (интенсивная)	K-TT-B	kattaba “много писать”

- Префиксы/суффиксы:

Лицо+Число	перфект (суфф.)	имперфект
+3+Masc	-a	ya- -u
+3+Fem	-at	ta- -u

- Варианты огласовок:

Вид	Залог	Префикс	Огласовка I	Огласовка II
Сов.	Активный		a-a	a-a
Сов.	Пассивный		u-i	u-i
Несов.	Активный	ya-	∅-u	a-i
Несов.	Пассивный	yu-	∅-a	a-a

Арабское спряжение в FOMA: формат входа

- Формат входа:

```
define Vowel [ a | i | u ];
define Consonant [ k | t | b | z | h | r | s | f | m | d | n | y];
define Letter [ Vowel | Consonant ];
define Stem Consonant Consonant Consonant;
define Type [ "+I" | "+II" ];
define Voice ["+Act" | "+Pass"];
define Aspect ["+Perf" | "+Imperf"];
define Person "+3";
define Gender ["+M" | "+F"];
define Input Stem Type Voice Aspect Person Gender;
```

- Позиции гласных маркируются цифрами:

```
define 0Insertion [..] -> "0" || .#. _ ;
define 1Insertion [..] -> "1" || "0" Consonant _ ;
define 2Insertion [..] -> "2" || "1" Consonant _ ;
define 3Insertion [..] -> "3" || "2" Consonant _ ;
define PosInsertion 0Insertion .o. 1Insertion .o. 2Insertion .o. 3Insertion;
```

Арабское спряжение в FOMA: огласовки

- Удвоение второй согласной в интенсиве:

```
define CheckTypeI ?* "+I" ?*;
define CheckTypeII ?* "+II" ?*;
define TypeIIDuplication k -> [k k], b -> [b b], t -> [t t], z -> [z z], h
      -> [h h], r -> [r r], s -> [s s], f -> [f f], m -> [m m], d -> [d d], n
       -> [n n] || _ "2";
define StemProcessing [ CheckTypeI ] | [ CheckTypeII .o. TypeIIDuplication ];
```

- Огласовки:

```
define aaFill "1" -> a, "2" -> a;
define aiFill "1" -> a, "2" -> i;
define uiFill "1" -> u, "2" -> i;
define 0aFill "1" -> [], "2"-> a;
define 0uFill "1" -> [], "2"-> u;
```

Арабское спряжение в FOMA: выбор правила

- Полный перебор в поисках правила:

```
define PerfectActiveFill aaFill;
define ImperfectActiveFill [ CheckTypeI .o. 0uFill ] | [ CheckTypeII .o. aiFill ];
define ActiveFill [CheckPerf .o. PerfectActiveFill] | [CheckImperf .o.
      ImperfectActiveFill];
define PerfectPassiveFill uiFill;
define ImperfectPassiveFill [ CheckTypeI .o. 0aFill ] | [ CheckTypeII .o. aaFill ];
define PassiveFill [CheckPerf .o. PerfectPassiveFill] | [CheckImperf .o.
      ImperfectPassiveFill];
define Fill [CheckPass .o. PassiveFill] | [CheckAct .o. ActiveFill] ;
```

- То же для префиксов (маркер 0):

```
define 0Prefix "0" -> [];
define yaPrefix "0" -> y a;
define yuPrefix "0" -> y u;
define taPrefix "0" -> t a;
define tuPrefix "0" -> t u;
define PerfectPrefix 0Prefix;
define ImperfectActivePrefix [CheckMasc .o. yaPrefix] | [CheckFem .o. taPrefix] ;
define ImperfectPassivePrefix [CheckMasc .o. yuPrefix] | [CheckFem .o. tuPrefix] ;
define ImperfectPrefix [CheckAct .o. ImperfectActivePrefix] | [CheckPass .o.
      ImperfectPassivePrefix] ;
define Prefix [CheckPerf .o. PerfectPrefix] | [CheckImperf .o. ImperfectPrefix] ;
```

Арабское спряжение в FOMA: выбор правила

- Выбор суффикса (маркер 3):

```
define ImperfectSuffix "3" -> u || _ Type;
define PerfectMascSuffix "3" -> a || _ Type;
define PerfectFemSuffix "3" -> a t || _ Type;
define PerfectSuffix [ CheckMasc .o. PerfectMascSuffix ] | [CheckFem .o.
      PerfectFemSuffix ] ;
define Suffix [ CheckPerf .o. PerfectSuffix ] | [ CheckImperf .o. ImperfectSuffix ];
```

- Объединение этапов:

```
define Cleanup Type | Voice | Aspect | Person | Gender -> [] ;
define Grammar Input .o. PosInsertion .o. StemProcessing .o. Fill .o. Prefix .o.
      Suffix .o. Cleanup;
```

- В реальности всё ещё сложнее.
- Тем не менее, один из первых языков с автоматной моделью морфологии (Beesley, 1990).

Редупликация: описание

- Редупликация — точное или приблизительное удвоение части слова (*фигли-мигли*, *штучки-дрючки*, . . .):
- В суахили — регулярная модель:

soma	“читать”	*somasoma*	“немного читать”
piga	“бить”	*somasoma*	“слабо бить”
kula	“есть”	*kulakula*	“немного есть”
enda	“идти”	*endaenda*	“медленно идти”

- Можно ли выразить конечными преобразователями?

Редупликация: математические свойства

- Область определения конечного преобразования — автоматный язык.
- Область значений конечного преобразования — автоматный язык.

Теорема

Язык квадратов $\{ww|w \in \Sigma^*\}$ неавтоматен при $|\Sigma| \geqslant 2$ (то есть уже над $\{a, b\}$).

Доказательство

- От противного: пусть данный язык автоматен, тогда для него существует ДКА с конечным числом состояний.
- Слов вида a^k бесконечное число $\Leftrightarrow$ какие-то два из них приводят в одно состояние.
- Пусть это a^r и a^s, $r \neq s$.
- Продолжим их словом b^r, тогда $a^r b^r$ и $a^s b^r$ тоже приводят в одно состояние.
- Но это невозможно: одно из них в языке, другое — нет.

Редупликация: невозможность реализации

- Если б редупликация задавалась конечным преобразованием, то её область значений была бы автоматной.
- Это не так: шаблон ww не автоматен.
- Невозможно задать:

  Редупликация в языке бамбара (семья манде):

wulu “собака”	*wulu-o-wulu* “любая собака”
wulu-nyinina “видящий собаку”	*wulu-nyinina-o-wulu-nyinina* “кто угодно, видящий собаку”

- Ограниченная редупликация (только первый слог) тоже не задаётся.