Математические модели в лингвистике

Вычислительная морфология.

Мати Пентус, Александр Пиперски, Алексей Сорокин

МГУ им. М. В. Ломоносова, межфакультетский курс,
осенний семестр 2015–2016 учебного года

Конечные преобразователи

Пусть Σ, Γ — конечные алфавиты.

Определение конечного преобразователя

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

Конечные преобразователи

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

Конечные преобразователи

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

Конечные преобразователи

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.

Конечные преобразователи

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.

Конечный преобразователь принимает пары слов $\langle u, v \rangle \in \Sigma^* \times \Gamma^*$, которые являются метками путей из начального состояния в завершающие. Также можно считать, что конечный преобразователь задаёт многозначное отображение из Σ^* в Γ^*.

Свойства конечных преобразователей

Конечные преобразования замкнуты относительно:

- Конкатенации

Свойства конечных преобразователей

Конечные преобразования замкнуты относительно:

- Конкатенации
- Объединения

Свойства конечных преобразователей

Конечные преобразования замкнуты относительно:

- Конкатенации
- Объединения
- Композиции

Свойства конечных преобразователей

Конечные преобразования замкнуты относительно:

- Конкатенации
- Объединения
- Композиции
- Приоритетного объединения ($\cup_p$).

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

Свойства конечных преобразователей

Конечные преобразования замкнуты относительно:

- Конкатенации
- Объединения
- Композиции
- Приоритетного объединения ($\cup_p$).

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

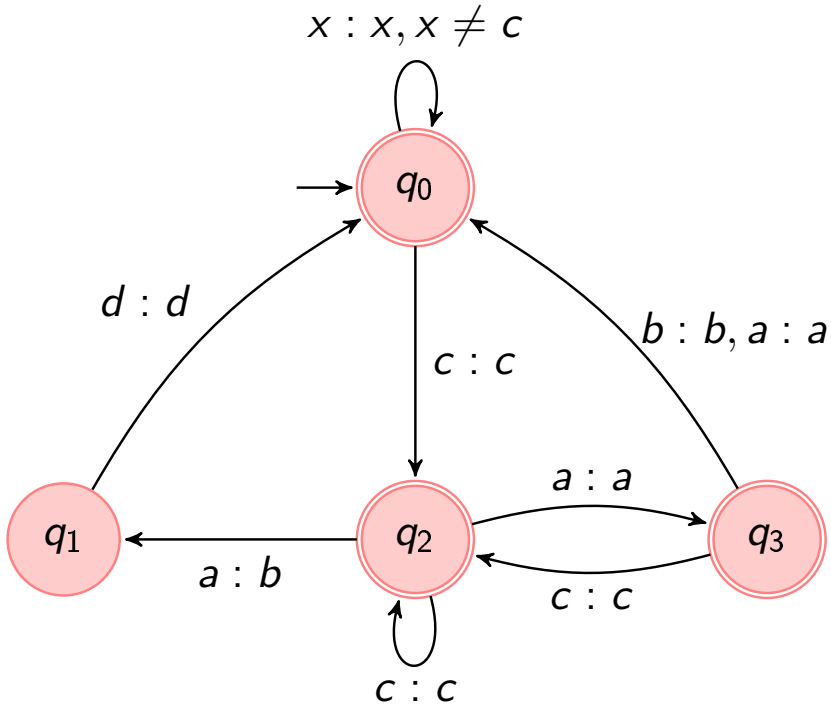
- Обращения: $\phi^{-1} = \{\langle y, x\rangle \mid \langle x, y\rangle \in \phi\}$.

Контекстные замены

С помощью конечных преобразователей можно задавать контекстные замены $X \rightarrow Y || U\_V$.

Контекстные замены

С помощью конечных преобразователей можно задавать контекстные замены $X \to Y||U\_V$. Конечный преобразователь для контекстной замены $a \to b||c\_d$ $(\Sigma = \Gamma = \{a, b, c, d\})$.



“Модельная задача”

Морфологический синтез: множественное число существительного в английском

“Модельная задача”

Морфологический синтез: множественное число существительного в английском

- *cat+N+Pl* $\leftrightarrows$ *cats*
- *torch+N+Pl* $\leftrightarrows$ *torches*
- *ally+N+Pl* $\leftrightarrows$ *allies*
- *play+N+Pl* $\leftrightarrows$ *plays*
- *goose+N+Pl* $\leftrightarrows$ *geese*
- *formula+N+Pl* $\leftrightarrows$ *formulas/formulae*

Решение модельной задачи

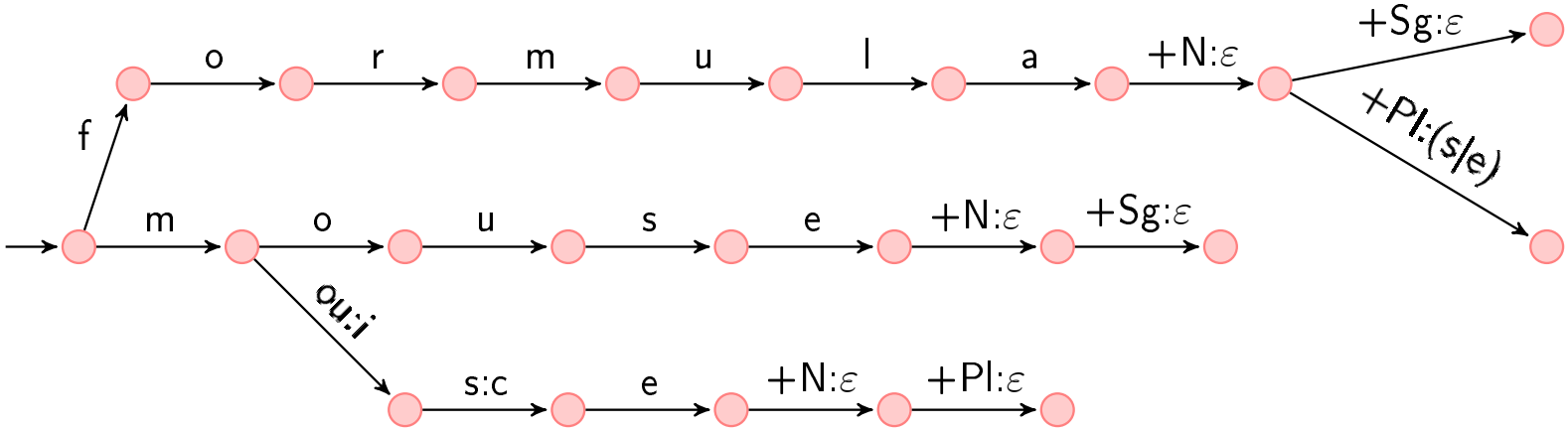
Для решения нужны следующие конечные преобразователи:

- A_1 — проверка условия $\Sigma^+(+\mathrm{N})(+\mathrm{Sg}|+\mathrm{Pl})$ (+N, +Sg,+Pl — вспомогательные символы)

Решение модельной задачи

Для решения нужны следующие конечные преобразователи:

- A_1 — проверка условия $\Sigma^+(+\mathrm{N})(+\mathrm{Sg}|+\mathrm{Pl})$ (+N, +Sg,+Pl — вспомогательные символы)
- T_{exc} — преобразователь для исключений (фрагмент):



Решение модельной задачи

Для решения нужны следующие конечные преобразователи:

- A_1 — проверка условия $\Sigma^+(+\mathrm{N})(+\mathrm{Sg}|+\mathrm{Pl})$ (+N, +Sg,+Pl — вспомогательные символы)
- T_{exc} — преобразователь для исключений (фрагмент):
- T_s: замены $+\mathrm{N}+\mathrm{Sg} \rightarrow \widehat{}||\_\#, +\mathrm{N}+\mathrm{Pl} \rightarrow \widehat{}s||\_\#$ (# — конец строки, $\widehat{}$ — вспомогательный символ)

Решение модельной задачи

Для решения нужны следующие конечные преобразователи:

- A_1 — проверка условия $\Sigma^+(+\mathrm{N})(+\mathrm{Sg}|+\mathrm{Pl})$ (+N, +Sg,+Pl — вспомогательные символы)
- T_{exc} — преобразователь для исключений (фрагмент):
- T_s: замены $+\mathrm{N}+\mathrm{Sg} \rightarrow \widehat{}\,||\_\#, +\mathrm{N}+\mathrm{Pl} \rightarrow \widehat{}\,s||\_\#$ (# — конец строки, $\widehat{}$ — вспомогательный символ)
- T_y: замена $\mathrm{y} \rightarrow \mathrm{ie} \,||\, \_\widehat{}\,s\#$

Решение модельной задачи

Для решения нужны следующие конечные преобразователи:

- A_1 — проверка условия $\Sigma^+(+\mathrm{N})(+\mathrm{Sg}|+\mathrm{Pl})$ (+N, +Sg,+Pl — вспомогательные символы)
- T_{exc} — преобразователь для исключений (фрагмент):
- T_s: замены $+\mathrm{N}+\mathrm{Sg} \rightarrow \widehat{}||\_\#, +\mathrm{N}+\mathrm{Pl} \rightarrow \widehat{}s||\_\#$ (# — конец строки, $\widehat{}$ — вспомогательный символ)
- T_y: замена $\mathrm{y} \rightarrow \mathrm{ie} \,||\, \_\widehat{}s\#$
- $A_{exc,sib}$: проверка условия $\Sigma^+ arch\widehat{}s\#$

Решение модельной задачи

Для решения нужны следующие конечные преобразователи:

- A_1 — проверка условия $\Sigma^+(+\mathrm{N})(+\mathrm{Sg}|+\mathrm{Pl})$ (+N, +Sg,+Pl — вспомогательные символы)
- T_{exc} — преобразователь для исключений (фрагмент):
- T_s: замены $+\mathrm{N}+\mathrm{Sg} \rightarrow \widehat{}\,||\_\#, +\mathrm{N}+\mathrm{Pl} \rightarrow \widehat{}\,s||\_\#$ (# — конец строки, $\widehat{}$ — вспомогательный символ)
- T_y: замена $\mathrm{y} \rightarrow \mathrm{ie} \,||\, \_\widehat{}\,s\#$
- $A_{exc,sib}$: проверка условия $\Sigma^+ arch\widehat{}\,s\#$
- T_{sib}: замена $\rightarrow \mathrm{e} \,||\, [s|z|x|sh|ch]\_\widehat{}\,s\#$

Решение модельной задачи

Для решения нужны следующие конечные преобразователи:

- A_1 — проверка условия $\Sigma^+(+\mathrm{N})(+\mathrm{Sg}|+\mathrm{Pl})$ (+N, +Sg,+Pl — вспомогательные символы)
- T_{exc} — преобразователь для исключений (фрагмент):
- T_s: замены $+\mathrm{N}+\mathrm{Sg} \rightarrow \widehat{}\,||\_\#, +\mathrm{N}+\mathrm{Pl} \rightarrow \widehat{}\,s||\_\#$ (# — конец строки, $\widehat{}$ — вспомогательный символ)
- T_y: замена $\mathrm{y} \rightarrow \mathrm{ie} \,||\, \_\widehat{}\,s\#$
- $A_{exc,sib}$: проверка условия $\Sigma^+ arch\widehat{}\,s\#$
- T_{sib}: замена $\rightarrow \mathrm{e} \,||\, [s|z|x|sh|ch]\_\widehat{}\,s\#$
- T_{cl}: очистка $\widehat{} \rightarrow \,||\, \_$

Решение модельной задачи

Для решения нужны следующие конечные преобразователи:

- A_1 — проверка условия $\Sigma^+(+\mathrm{N})(+\mathrm{Sg}|+\mathrm{Pl})$ (+N, +Sg,+Pl — вспомогательные символы)
- T_{exc} — преобразователь для исключений (фрагмент):
- T_s: замены $+\mathrm{N}+\mathrm{Sg} \rightarrow \widehat{}\,||\_\#, +\mathrm{N}+\mathrm{Pl} \rightarrow \widehat{}\,s||\_\#$ (# — конец строки, $\widehat{}$ — вспомогательный символ)
- T_y: замена $\mathrm{y} \rightarrow \mathrm{ie} \,||\, \_\widehat{}\,s\#$
- $A_{exc,sib}$: проверка условия $\Sigma^+ arch\widehat{}\,s\#$
- T_{sib}: замена $\rightarrow \mathrm{e} \,||\, [s|z|x|sh|ch]\_\widehat{}\,s\#$
- T_{cl}: очистка $\widehat{} \rightarrow \,||\, \_$
- Общий преобразователь: $A_1 \circ (T_{exc} \cup_p (T_s \circ T_y \circ (A_{exc,sib} \cup_P T_{sib}) \circ T_{cl}))$

Компиляторы для конечных преобразований

- XFST (Xerox finite-state toolkit)
- HFST (Helsinki finite-state toolkit)
- FOMA
- SFST (Stuttgart finite-state toolkit)
- OpenFST

Программа FOMA

- FOMA — программа для компиляции описаний конечных преобразователей в преобразователи.
- Разработана М. Халденом (Mans Hulden) в 2009–2015гг, последняя версия 0.9.18 вышла 12 июня 2015г.

Программа FOMA

- FOMA — программа для компиляции описаний конечных преобразователей в преобразователи.
- Разработана М. Халденом (Mans Hulden) в 2009–2015гг, последняя версия 0.9.18 вышла 12 июня 2015г.
- Программа с открытым кодом, написана на языке C++. Существует библиотека для использования в программах на C++. Написана под linux, допускает использование под Windows.

Программа FOMA

- FOMA — программа для компиляции описаний конечных преобразователей в преобразователи.
- Разработана М. Халденом (Mans Hulden) в 2009–2015гг, последняя версия 0.9.18 вышла 12 июня 2015г.
- Программа с открытым кодом, написана на языке C++. Существует библиотека для использования в программах на C++. Написана под linux, допускает использование под Windows.
- Получает на вход описания преобразований в xfst-формате (Karttunen, 2003) и преобразует их в конечные преобразователи.

Программа FOMA

- FOMA — программа для компиляции описаний конечных преобразователей в преобразователи.
- Разработана М. Халденом (Mans Hulden) в 2009–2015гг, последняя версия 0.9.18 вышла 12 июня 2015г.
- Программа с открытым кодом, написана на языке C++. Существует библиотека для использования в программах на C++. Написана под linux, допускает использование под Windows.
- Получает на вход описания преобразований в xfst-формате (Karttunen, 2003) и преобразует их в конечные преобразователи.
- Также позволяет преобразовывать регулярные выражения в конечные автоматы и выполнять с ними различные операции.
- Сайт: `https://code.google.com/p/foma/`.

Код для модельной задачи

```
### 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 yReplacement y -> i e || Consonant _ "^" s .#.;
define sibException [ Letter ]+ a r c h "^" s ;
define Sibilant [ x | s | z | c h | s h ];
define eInsertion [..] -> e || Sibilant _ "^" s .#.;
define checkSibilant [ sibException .P. eInsertion ];
define Cleanup "^" -> [] || _ ;
define RegularNoun [ NounAffixation .o. yReplacement .o. checkSibilant .o. Cleanup
    ] ;
define Grammar Noun .o. [ IrregularNounPlural .P. RegularNoun ];

push Grammar
```

Лексикон

- Лексикон (*.lexc) используется для задания конечных регулярных выражений/преобразований.

```
!!! lexicon.lexc !!!
Multichar_Symbols +N +V +Past +Pres +Part +1 +2 +3
    +Sg +Pl

LEXICON IrregularPlural

goose+N+Pl:geese #;
mouse+N+Pl:mice #;
foot+N+Pl:feet #;
formula+N+Pl:formulas #;
formula+N+Pl:formulae #;
cactus+N+Pl:cactuses #;
cactus+N+Pl:cacti #;
```

Лексикон

- Лексикон (*.lexc) используется для задания конечных регулярных выражений/преобразований.

```
!!! lexicon.lexc !!!
Multichar_Symbols +N +V +Past +Pres +Part +1 +2 +3
    +Sg +Pl

LEXICON IrregularPlural

goose+N+Pl:geese #;
mouse+N+Pl:mice #;
foot+N+Pl:feet #;
formula+N+Pl:formulas #;
formula+N+Pl:formulae #;
cactus+N+Pl:cactuses #;
cactus+N+Pl:cacti #;
```

- Применяется для запоминания словарей и создания списков исключений, а также задания согласовательных классов.

Лексикон

- Лексикон (*.lexc) используется для задания конечных регулярных выражений/преобразований.

```
!!! lexicon.lexc !!!
Multichar_Symbols +N +V +Past +Pres +Part +1 +2 +3
    +Sg +Pl

LEXICON IrregularPlural

goose+N+Pl:geese #;
mouse+N+Pl:mice #;
foot+N+Pl:feet #;
formula+N+Pl:formulas #;
formula+N+Pl:formulae #;
cactus+N+Pl:cactuses #;
cactus+N+Pl:cacti #;
```

- Применяется для запоминания словарей и создания списков исключений, а также задания согласовательных классов.
- Также в лексиконе принято задавать мультисимвольные последовательности.

Синтаксис FOMA: регулярные преобразования

- FOMA позволяет задавать регулярные выражения.

Синтаксис FOMA: регулярные преобразования

- FOMA позволяет задавать регулярные выражения.
- В качестве элементарных регулярных выражений используются ASCII-символы, а также сохранённые мультисимвольные последовательности. Например, “+N” считается одним символом.

Синтаксис FOMA: регулярные преобразования

- FOMA позволяет задавать регулярные выражения.
- В качестве элементарных регулярных выражений используются ASCII-символы, а также сохранённые мультисимвольные последовательности. Например, “+N” считается одним символом.
- Рекомендуется разделять перемножаемые регулярные выражения пробелами, чтобы конкатенация не воспринималась как один символ.

Синтаксис FOMA: регулярные преобразования

- FOMA позволяет задавать регулярные выражения.
- В качестве элементарных регулярных выражений используются ASCII-символы, а также сохранённые мультисимвольные последовательности. Например, “+N” считается одним символом.
- Рекомендуется разделять перемножаемые регулярные выражения пробелами, чтобы конкатенация не воспринималась как один символ.
- Регулярные преобразования определяются аналогично регулярным выражениям...

Синтаксис FOMA: регулярные преобразования

- FOMA позволяет задавать регулярные выражения.
- В качестве элементарных регулярных выражений используются ASCII-символы, а также сохранённые мультисимвольные последовательности. Например, “+N” считается одним символом.
- Рекомендуется разделять перемножаемые регулярные выражения пробелами, чтобы конкатенация не воспринималась как один символ.
- Регулярные преобразования определяются аналогично регулярным выражениям...
- Базовые преобразования: контекстные замены.

Операции над регулярными выражениями

Поддерживаются следующие операции над регулярными преобразованиями:

- объединение (|),

Операции над регулярными выражениями

Поддерживаются следующие операции над регулярными преобразованиями:

- объединение (|),
- конкатенация (задаётся простым приписыванием),

Операции над регулярными выражениями

Поддерживаются следующие операции над регулярными преобразованиями:

- объединение ($|$),
- конкатенация (задаётся простым приписыванием),
- итерация ($*$),
- положительная итерация ($+$),

Операции над регулярными выражениями

Поддерживаются следующие операции над регулярными преобразованиями:

- объединение ($|$),
- конкатенация (задаётся простым приписыванием),
- итерация ($*$),
- положительная итерация ($+$),
- взятие 0 или 1 раз (задаётся взятием в круглын скобки): (X) соответствует $X|1$ в стандартном синтаксисе,

Операции над регулярными выражениями

Поддерживаются следующие операции над регулярными преобразованиями:

- объединение ($|$),
- конкатенация (задаётся простым приписыванием),
- итерация ($*$),
- положительная итерация ($+$),
- взятие 0 или 1 раз (задаётся взятием в круглын скобки): (X) соответствует $X|1$ в стандартном синтаксисе,
- вместо круглых скобок — квадратные,

Операции над регулярными выражениями

Поддерживаются следующие операции над регулярными преобразованиями:

- объединение ($|$),
- конкатенация (задаётся простым приписыванием),
- итерация ($*$),
- положительная итерация ($+$),
- взятие 0 или 1 раз (задаётся взятием в круглын скобки): (X) соответствует $X|1$ в стандартном синтаксисе,
- вместо круглых скобок — квадратные,
- [] или [..] — обозначение для пустого слова.

Регулярные преобразования

- Базовые регулярные преобразования: контекстные замены $X \rightarrow Y || U \_ V$ записываются в точности в таком виде.
- В качестве X и Y можно брать произвольные регулярные выражения, а также символ начала/конца строки .#..

Регулярные преобразования

- Базовые регулярные преобразования: контекстные замены $X \rightarrow Y || U \_ V$ записываются в точности в таком виде.
- В качестве X и Y можно брать произвольные регулярные выражения, а также символ начала/конца строки .#.. Например, $a \rightarrow b || \_ c+.\#.$ означает заменить a на b, если после этого до конца слова встречаются только буквы c, притом в ненулевом количестве.

Регулярные преобразования

- Базовые регулярные преобразования: контекстные замены $X \rightarrow Y || U \_ V$ записываются в точности в таком виде.
- В качестве X и Y можно брать произвольные регулярные выражения, а также символ начала/конца строки .#.. Например, $a \rightarrow b || \_ c+.\#.$ означает заменить a на b, если после этого до конца слова встречаются только буквы c, притом в ненулевом количестве.
- В качестве преобразований также используются регулярные выражения X: такое преобразование действует только на слова из X, отображая их в себя.

Операции над регулярными преобразованиями

- Регулярные преобразования можно сохранять в переменные с помощью команды **define**:

define⟨имя_переменной⟩⟨выражение⟩;

Операции над регулярными преобразованиями

- Регулярные преобразования можно сохранять в переменные с помощью команды **define**:

$$\textbf{define}\langle\text{имя\_переменной}\rangle\langle\text{выражение}\rangle;$$

- Регулярные преобразования можно комбинировать с помощью операций над регулярными выражениями, а также:

Операции над регулярными преобразованиями

- Регулярные преобразования можно сохранять в переменные с помощью команды **define**:

$$\textbf{define}\langle\text{имя\_переменной}\rangle\langle\text{выражение}\rangle;$$

- Регулярные преобразования можно комбинировать с помощью операций над регулярными выражениями, а также:
  - Композиции (.*o*.)

Операции над регулярными преобразованиями

- Регулярные преобразования можно сохранять в переменные с помощью команды **define**:

$$\textbf{define}\langle\text{имя\_переменной}\rangle\langle\text{выражение}\rangle;$$

- Регулярные преобразования можно комбинировать с помощью операций над регулярными выражениями, а также:
  - Композиции (.*o*.)
  - Приоритетного объединения (.*p*.)

FOMA: команды

- Для сохранения промежуточных преобразователей используется стек.

FOMA: команды

- Для сохранения промежуточных преобразователей используется стек.
- Команды для работы со стеком:
  - **source** ⟨имя_файла⟩ — выполнить команды из файла
  - **push** ⟨имя_переменной⟩ — поместить преобразователь, хранящийся в переменной, на вершину стека.

FOMA: команды

- Для сохранения промежуточных преобразователей используется стек.
- Команды для работы со стеком:
  - **source** ⟨имя_файла⟩ — выполнить команды из файла
  - **push** ⟨имя_переменной⟩ — поместить преобразователь, хранящийся в переменной, на вершину стека.
  - **pop** — убрать преобразователь с вершины стека.

FOMA: команды

- Для сохранения промежуточных преобразователей используется стек.
- Команды для работы со стеком:
  - **source** ⟨имя_файла⟩ — выполнить команды из файла
  - **push** ⟨имя_переменной⟩ — поместить преобразователь, хранящийся в переменной, на вершину стека.
  - **pop** — убрать преобразователь с вершины стека.
  - **down** — запустить преобразователь на вершине стека в “прямом” направлении.

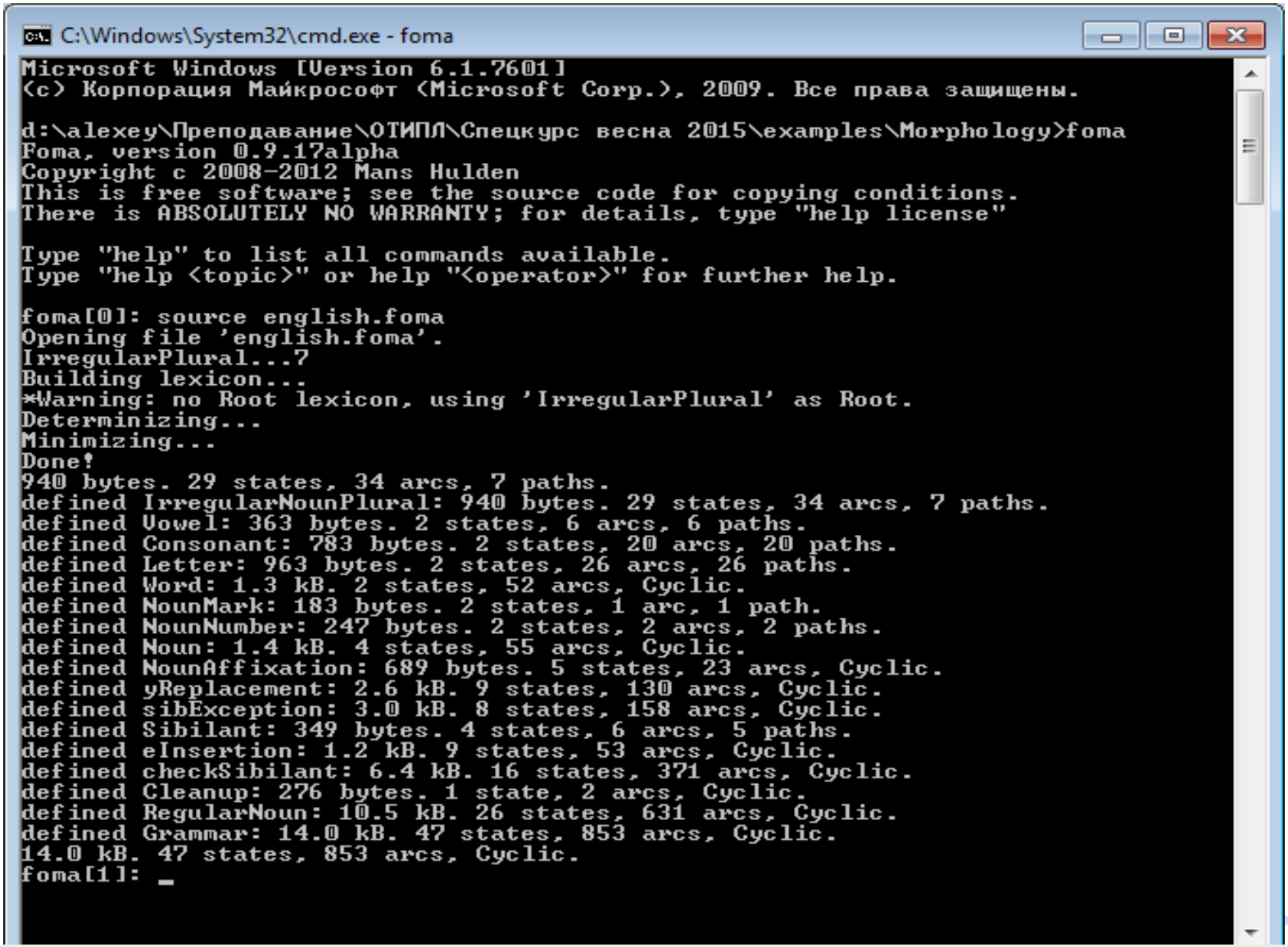
FOMA: команды

- Для сохранения промежуточных преобразователей используется стек.
- Команды для работы со стеком:
  - **source** ⟨имя_файла⟩ — выполнить команды из файла
  - **push** ⟨имя_переменной⟩ — поместить преобразователь, хранящийся в переменной, на вершину стека.
  - **pop** — убрать преобразователь с вершины стека.
  - **down** — запустить преобразователь на вершине стека в “прямом” направлении.
  - **up** — запустить преобразователь на вершине стека в “обратном” направлении.

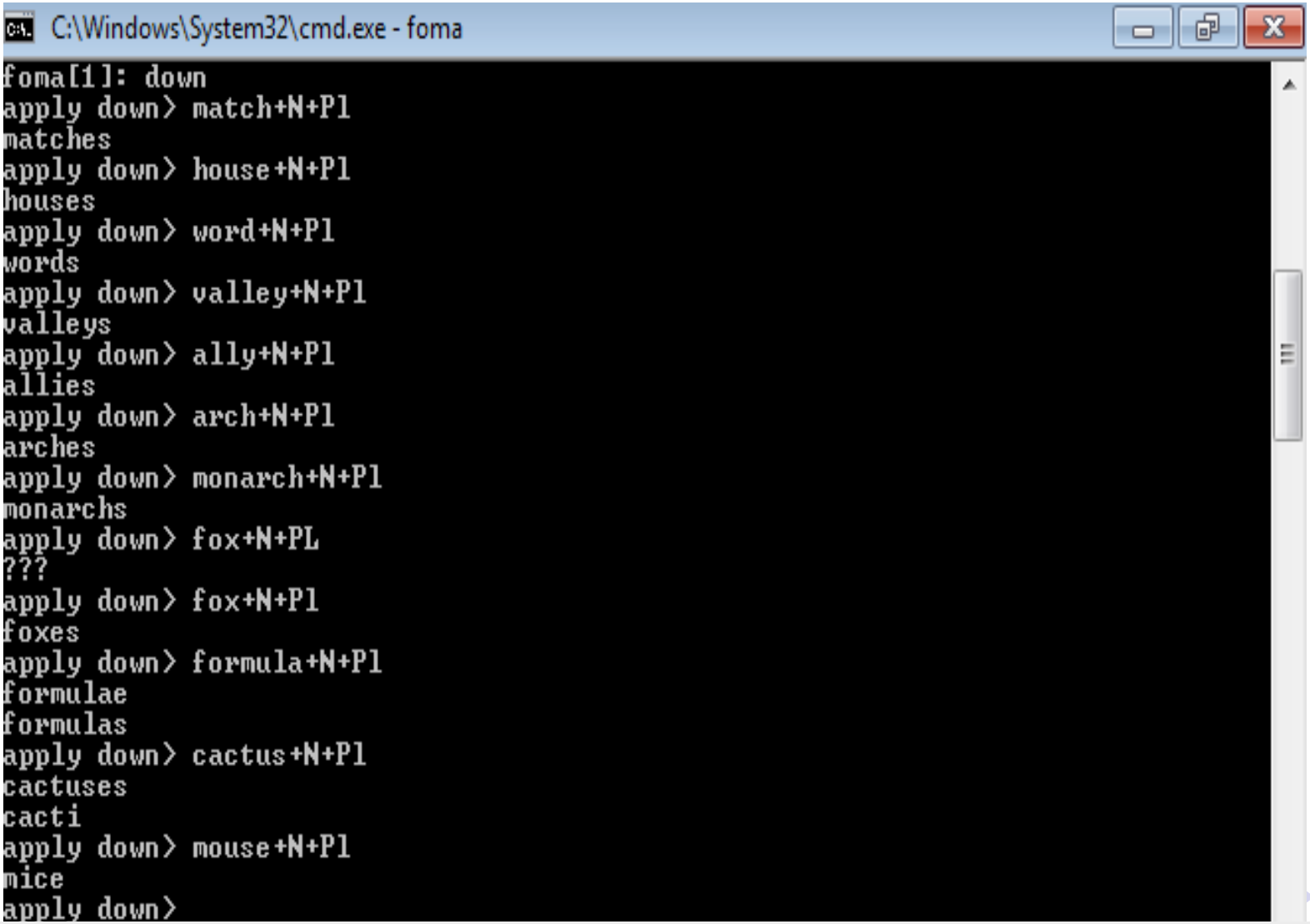
FOMA: команды

- Для сохранения промежуточных преобразователей используется стек.
- Команды для работы со стеком:
  - **source** ⟨имя_файла⟩ — выполнить команды из файла
  - **push** ⟨имя_переменной⟩ — поместить преобразователь, хранящийся в переменной, на вершину стека.
  - **pop** — убрать преобразователь с вершины стека.
  - **down** — запустить преобразователь на вершине стека в “прямом” направлении.
  - **up** — запустить преобразователь на вершине стека в “обратном” направлении.
  - **upper** — напечатать слова, к которым применим преобразователь.
  - **lower** — напечатать слова, которые могут быть результатами работы преобразователя.

Компиляция кода



Примеры использования



Свойства модели с конечным числом состояний

- Хорошо подходит для аддитивной модели словоизменения и агглютинативных языков.

Свойства модели с конечным числом состояний

- Хорошо подходит для аддитивной модели словоизменения и агглютинативных языков.
- Чем дальше морфология от аддитивной модели, тем менее естественна модель с конечным числом состояний.

Свойства модели с конечным числом состояний

- Хорошо подходит для аддитивной модели словоизменения и агглютинативных языков.
- Чем дальше морфология от аддитивной модели, тем менее естественна модель с конечным числом состояний.

Возможные трудности:

- нелокальные явления (гармония гласных и т.д),

Свойства модели с конечным числом состояний

- Хорошо подходит для аддитивной модели словоизменения и агглютинативных языков.
- Чем дальше морфология от аддитивной модели, тем менее естественна модель с конечным числом состояний.

Возможные трудности:

- нелокальные явления (гармония гласных и т.д),
- многочисленность словоизменительных классов,

Свойства модели с конечным числом состояний

- Хорошо подходит для аддитивной модели словоизменения и агглютинативных языков.
- Чем дальше морфология от аддитивной модели, тем менее естественна модель с конечным числом состояний.

Возможные трудности:

- нелокальные явления (гармония гласных и т.д),
- многочисленность словоизменительных классов,
- неавтоматические чередования, развитая фузия,

Свойства модели с конечным числом состояний

- Хорошо подходит для аддитивной модели словоизменения и агглютинативных языков.
- Чем дальше морфология от аддитивной модели, тем менее естественна модель с конечным числом состояний.

Возможные трудности:

- нелокальные явления (гармония гласных и т.д),
- многочисленность словоизменительных классов,
- неавтоматические чередования, развитая фузия,
- неконкатенационная морфология (внутренняя флексия, аблаут),

Свойства модели с конечным числом состояний

- Хорошо подходит для аддитивной модели словоизменения и агглютинативных языков.
- Чем дальше морфология от аддитивной модели, тем менее естественна модель с конечным числом состояний.

Возможные трудности:

- нелокальные явления (гармония гласных и т.д),
- многочисленность словоизменительных классов,
- неавтоматические чередования, развитая фузия,
- неконкатенационная морфология (внутренняя флексия, аблаут),
- редупликация.

Глагольные формы в языке йоулумни

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

Глагольные формы в языке йоулумни

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 $\alpha_1, \alpha_2, \in C, \alpha_3 \in C \cup \{\varepsilon\}$, то основа герундия имеет вид $\alpha_1 V \alpha_2 \alpha_3$,

.

Глагольные формы в языке йоулумни

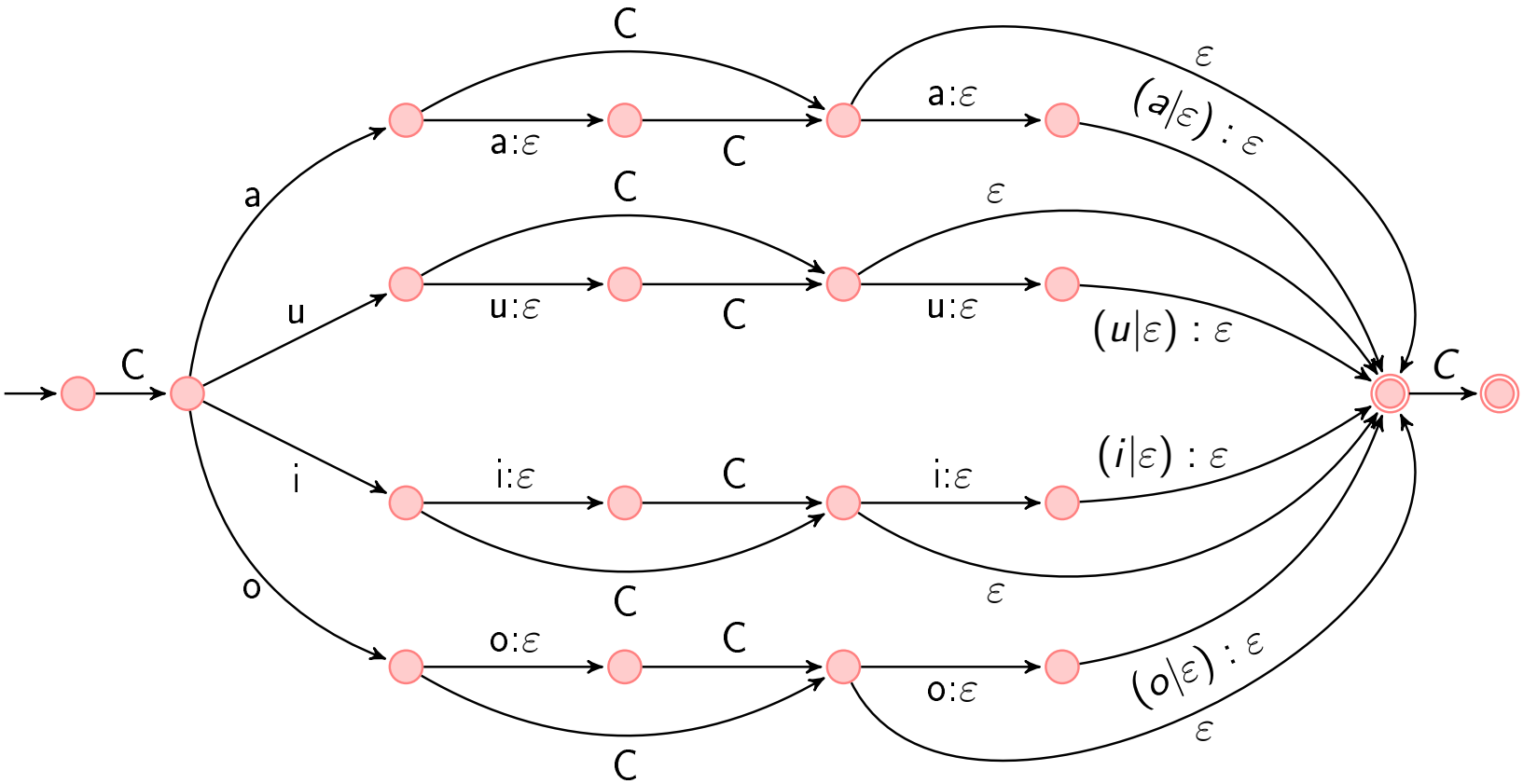
основа	герундий	дуратив
caw “кричать”	caw-inay	cawaa-ʔaa-n
cuum “разрушать”	cum-inay	cumuu-ʔaa-n
hoyoo “называть”	hoy-inay	hoyoo-ʔaa-n
diiyl “охранять”	diyl-inay	diyiil-ʔaa-n
ʔilk “петь”	ʔilk-inay	ʔiliik-ʔaa-n
hiwiit “гулять”	hiwt-inay	hiwiit-ʔaa-n

Глагольные формы в языке йоулумни

Если основа имела вид $\alpha_1 V(V)\alpha_2(V)(V)\alpha_3$, где $\alpha_1, \alpha_2, \in C, \alpha_3 \in C \cup \{\varepsilon\}$, то основа герундия имеет вид $\alpha_1 V \alpha_2 \alpha_3$, а основа дуратива — $\alpha_1 V \alpha_2 V V \alpha_3$. При этом все гласные должны быть одинаковы.

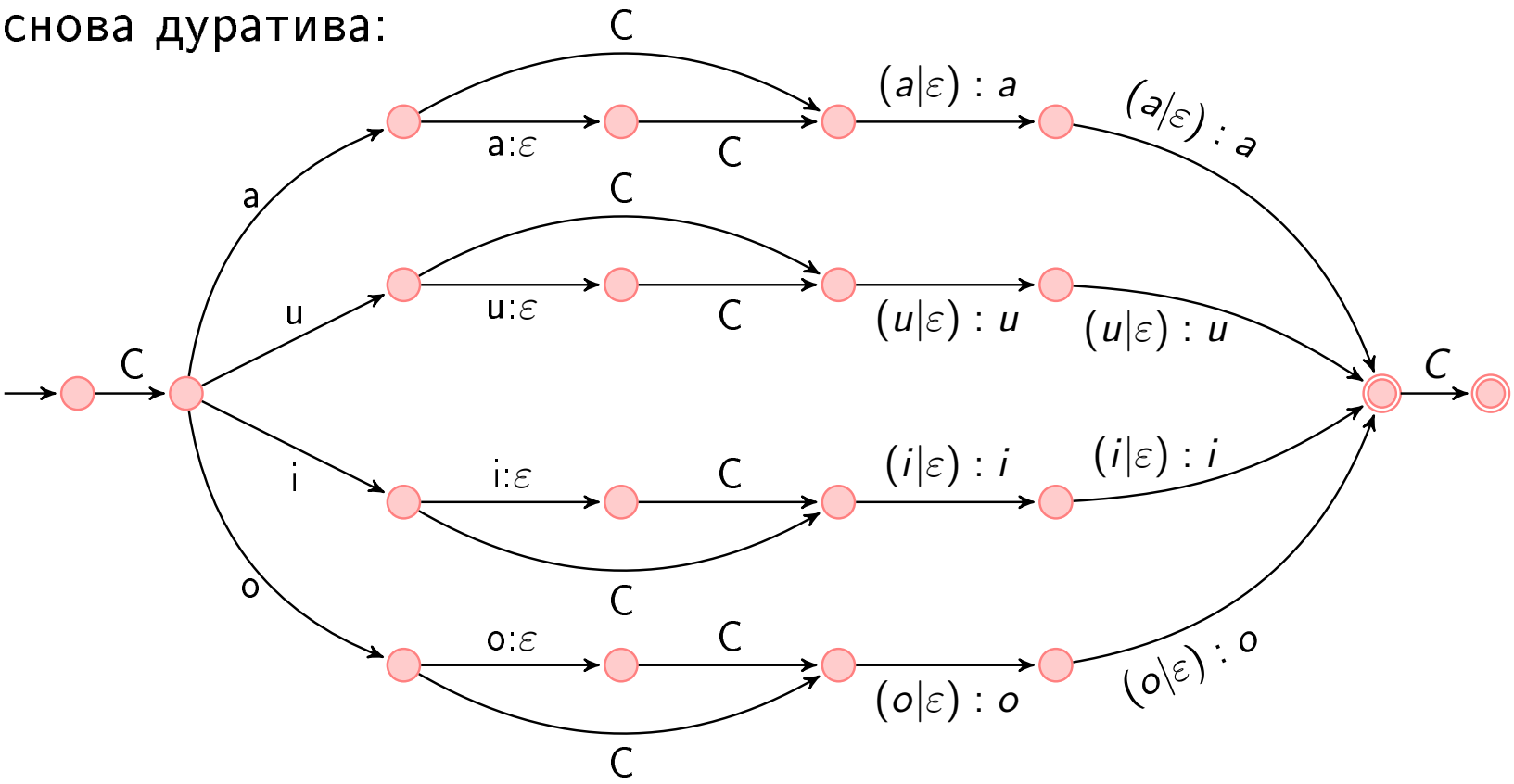
Преобразователи для глагольных форм в языке йоулумни

- Основа герундия:



Преобразователи для глагольных форм в языке йоулумни

- Основа герундия:
- Основа дуратива:



Код для морфологии в йоулумне

```
### youlumne.foma ###

define Vowel [a | i | o | u];
define Consonant [ c | w | m | h | y | d | l | g | k | t ];
define Letter [Consonant | Vowel];
define Stem Consonant Vowel (Vowel) Consonant (Vowel) (Vowel) (Consonant) ;
define VerbMark "+V";
define Mood "+Ger" "+Dur";
define Mark [ VerbMark Mood ];
define VowelPattern [ [Consonant | a]+ | [Consonant | i]+ | [Consonant | o]+ | [
    Consonant | u]+ ];
define Word Stem & VowelPattern;

# left context for not a leftmost vowel
define LeftContext1 Consonant Vowel [ Letter ]* ;
define VowelRemoval Vowel -> [] || LeftContext1 _ ;
define LeftContext2 Consonant Vowel Consonant ;
define DurativeVowelInsertion [..] -> [ a a | i i | o o | u u ] || LeftContext2 _ (
    Consonant) .#. ;
define GerundSuffixInsertion ["+V" "+Ger"] : [ i n a y ] ;
define DurativeSuffixInsertion ["+V" "+Dur"] : [ g a a n ] ;
define GerundStem Word .o. VowelRemoval ;
define DurativeStem GerundStem .o. DurativeVowelInsertion .o. VowelPattern ;
define Gerund [ GerundStem GerundSuffixInsertion ] ;
define Durative [ DurativeStem DurativeSuffixInsertion ] ;
define Grammar [ Gerund | Durative ] ;
push Grammar
```

Описание морфологии в йоулумне

- В суффиксе дуратива происходит вставка двух одинаковых гласных.

Описание морфологии в йоулумне

- В суффиксе дуратива происходит вставка двух одинаковых гласных.
- При этом эти гласные обязаны совпадать с гласной первого слога.

Описание морфологии в йоулумне

- В суффиксе дуратива происходит вставка двух одинаковых гласных.
- При этом эти гласные обязаны совпадать с гласной первого слога.
- Для этого нужно оставлять только слова, в которых все гласные одинаковы.

Описание морфологии в йоулумне

- В суффиксе дуратива происходит вставка двух одинаковых гласных.
- При этом эти гласные обязаны совпадать с гласной первого слога.
- Для этого нужно оставлять только слова, в которых все гласные одинаковы.
- Это можно задать регулярным выражением (шаблон Vowel-Pattern).

Описание морфологии в йоулумне

- В суффиксе дуратива происходит вставка двух одинаковых гласных.
- При этом эти гласные обязаны совпадать с гласной первого слога.
- Для этого нужно оставлять только слова, в которых все гласные одинаковы.
- Это можно задать регулярным выражением (шаблон Vowel-Pattern).
- Под тот же шаблон должны подходить и возможные основы.

Преобразователь для арабского словоизменения

- Для арабского характерна “разрывная” морфология.

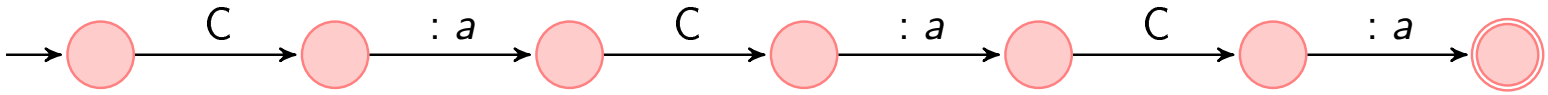
k t b *“писать” (корень)* + a-a + -a = kataba *“ (он) писал”*

Преобразователь для арабского словоизменения

- Для арабского характерна “разрывная” морфология.

k t b *“писать” (корень)* + a-a + -a = kataba *“ (он) писал”*

- Соответствующий преобразователь:

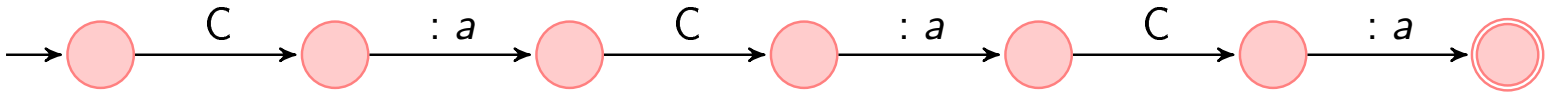


Преобразователь для арабского словоизменения

- Для арабского характерна “разрывная” морфология.

k t b *“писать” (корень)* + a-a + -a = kataba *“ (он) писал”*

- Соответствующий преобразователь:



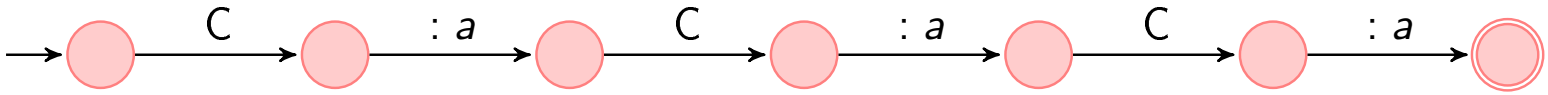
- Как описать то же самое в терминах контекстных замен?

Преобразователь для арабского словоизменения

- Для арабского характерна “разрывная” морфология.

k t b *“писать” (корень)* + a-a + -a = kataba *“ (он) писал”*

- Соответствующий преобразователь:



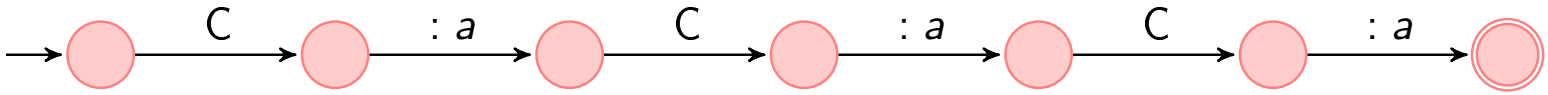
- Как описать то же самое в терминах контекстных замен?
- Можно ввести маркеры для позиций после первого согласного, после второго и т.д.

Преобразователь для арабского словоизменения

- Для арабского характерна “разрывная” морфология.

k t b *“писать” (корень)* + a-a + -a = kataba *“ (он) писал”*

- Соответствующий преобразователь:



- Как описать то же самое в терминах контекстных замен?
- Можно ввести маркеры для позиций после первого согласного, после второго и т.д.
- Дополнительная проблема — вставляемые гласные зависят от основы:

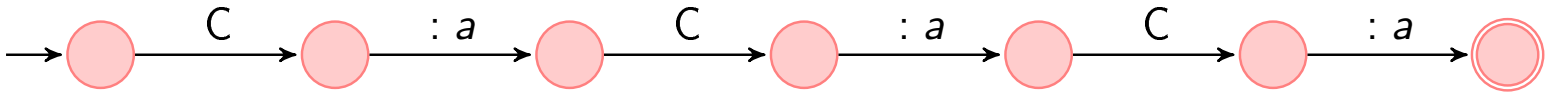
k t b *“писать” (корень)* + a-a + -a = kataba *“ (он) писал”*
m r d *“болеть” (корень)* + a-i + -a = marida *“ (он) болел”*

Преобразователь для арабского словоизменения

- Для арабского характерна “разрывная” морфология.

k t b *“писать” (корень)* + a-a + -a = kataba *“ (он) писал”*

- Соответствующий преобразователь:



- Как описать то же самое в терминах контекстных замен?
- Можно ввести маркеры для позиций после первого согласного, после второго и т.д.
- Дополнительная проблема — вставляемые гласные зависят от основы:

k t b *“писать” (корень)* + a-a + -a = kataba *“ (он) писал”*
m r d *“болеть” (корень)* + a-i + -a = marida *“ (он) болел”*

- Кроме того, есть “обычные” префиксы/суффиксы.

Арабская морфология: описание задачи

- Очень базовый пример морфологического синтезатора.

Арабская морфология: описание задачи

- Очень базовый пример морфологического синтезатора.
- вход: корень+⟨Type⟩+⟨Voice⟩+⟨Aspect⟩+⟨Person⟩+⟨Gender⟩

Арабская морфология: описание задачи

- Очень базовый пример морфологического синтезатора.
- вход: корень+⟨Type⟩+⟨Voice⟩+⟨Aspect⟩+⟨Person⟩+⟨Gender⟩
- ⟨Type⟩ $\in \{I, II\}$ — порода (базовая или интенсив)

Арабская морфология: описание задачи

- Очень базовый пример морфологического синтезатора.
- вход: корень+⟨Type⟩+⟨Voice⟩+⟨Aspect⟩+⟨Person⟩+⟨Gender⟩
- $\langle \text{Type} \rangle \in \{I, II\}$ — порода (базовая или интенсив)
- $\langle \text{Voice} \rangle \in \{\text{Act}, \text{Pass}\}$ — залог

Арабская морфология: описание задачи

- Очень базовый пример морфологического синтезатора.
- вход: корень+⟨Type⟩+⟨Voice⟩+⟨Aspect⟩+⟨Person⟩+⟨Gender⟩
- $\langle \text{Type} \rangle \in \{I, II\}$ — порода (базовая или интенсив)
- $\langle \text{Voice} \rangle \in \{\text{Act}, \text{Pass}\}$ — залог
- $\langle \text{Aspect} \rangle \in \{\text{Perf}, \text{Imperf}\}$ — вид

Арабская морфология: описание задачи

- Очень базовый пример морфологического синтезатора.
- вход: корень+⟨Type⟩+⟨Voice⟩+⟨Aspect⟩+⟨Person⟩+⟨Gender⟩
- $\langle \text{Type} \rangle \in \{I, II\}$ — порода (базовая или интенсив)
- $\langle \text{Voice} \rangle \in \{\text{Act}, \text{Pass}\}$ — залог
- $\langle \text{Aspect} \rangle \in \{\text{Perf}, \text{Imperf}\}$ — вид
- $\langle \text{Person} \rangle \in \{3\}$ — лицо
- $\langle \text{Gender} \rangle \in \{\text{M}, \text{F}\}$ — род
- Всего 16 вариантов.

Модель словообразования

Приводится по очерку арабской грамматики А. А. Зализняка.

Порода	Шаблон	Пример
I – базовая форма	K-T-B	kataba “писать”
II – интенсив	K-TT-B	kattaba “много писать”

Таблица основ

Форма	Окончание перфекта	Префикс и окончание имперфекта
3m	-a	ya- -u
3f	-at	ta- -u

Таблица окончаний

Модель словообразования

Вид	Залог	Преф.	Огласовка I	Огласовка II
Перфект	Актив		a-a, i-i, a-u	a-a
Перфект	Пассив		u-i	u-i
Имперфект	Актив	ya-	∅-u, ∅-i, ∅-a	a-i
Имперфект	Пассив	yu-	∅-a	a-a

Таблица огласовок

Модель словообразования

Вид	Залог	Преф.	Огласовка I	Огласовка II
Перфект	Актив		a-a, i-i, a-u	a-a
Перфект	Пассив		u-i	u-i
Имперфект	Актив	ya-	∅-u, ∅-i, ∅-a	a-i
Имперфект	Пассив	yu-	∅-a	a-a

Таблица огласовок

Вариант	Огласовка перфекта	Огласовка имперфекта	пример
1	a-a	∅-a	zahara “сиять”
2	a-a	∅-u	kataba “писать”
3	a-a	∅-i	safara “отправляться в путь”
4	a-i	∅-a	marida “быть больным”
5	a-u	∅-u	hasuna “быть красивым”

Варианты огласовок

Условие задачи

- $ktb{+}I{+}Act{+}Perf{+}3{+}M \leftrightarrows kataba$
- $ktb{+}II{+}Pass{+}Imperf{+}3{+}F \leftrightarrows tukattabu$
- $zhr{+}I{+}Act{+}Imperf{+}3{+}M \leftrightarrows yazharu$
- $sfr{+}I{+}Act{+}Imperf{+}3{+}F \leftrightarrows tasfiru$
- $mrd{+}II{+}Pass{+}Perf{+}3{+}M \leftrightarrows murrida$
- $hsn{+}I{+}Act{+}Perf{+}3{+}F \leftrightarrows hasunat$

Алгоритм работы синтезатора

- Input — проверка корректности и вставка метки класса

Алгоритм работы синтезатора

- Input — проверка корректности и вставка метки класса
- PosInsertion — вставка маркеров позиций

Алгоритм работы синтезатора

- Input — проверка корректности и вставка метки класса
- PosInsertion — вставка маркеров позиций
- Fill — заполнение огласовок (порода, вид, залог, класс),

Алгоритм работы синтезатора

- Input — проверка корректности и вставка метки класса
- PosInsertion — вставка маркеров позиций
- Fill — заполнение огласовок (порода, вид, залог, класс),
- Prefix — вставка префиксов (вид, залог, лицо, род)

Алгоритм работы синтезатора

- Input — проверка корректности и вставка метки класса
- PosInsertion — вставка маркеров позиций
- Fill — заполнение огласовок (порода, вид, залог, класс),
- Prefix — вставка префиксов (вид, залог, лицо, род)
- Suffix — вставка суффиксов (вид, лицо, род)

Алгоритм работы синтезатора

- Input — проверка корректности и вставка метки класса
- PosInsertion — вставка маркеров позиций
- Fill — заполнение огласовок (порода, вид, залог, класс),
- Prefix — вставка префиксов (вид, залог, лицо, род)
- Suffix — вставка суффиксов (вид, лицо, род)
- Cleanup — очистка меток
- Общий преобразователь: $\text{Input} \circ \text{PosInsertion} \circ \text{Fill} \circ \text{Prefix} \circ \text{Suffix} \circ \text{Cleanup}$.

Пример работы

- *sfr+I+Act+Imperf+3+F*

Пример работы

- *sfr+I+Act+Imperf+3+F*
- Input: *C3^sfr+I+Act+Imperf+3+F* (вставка метки класса)

Пример работы

- *sfr+I+Act+Imperf+3+F*
- Input: *C3^sfr+I+Act+Imperf+3+F* (вставка метки класса)
- PosInsertion: *C3^0s1f2r3+I+Act+Imperf+3+F* (вставка маркеров позиций)

Пример работы

- *sfr+I+Act+Imperf+3+F*
- Input: *C3^sfr+I+Act+Imperf+3+F* (вставка метки класса)
- PosInsertion: *C3^0s1f2r3+I+Act+Imperf+3+F* (вставка маркеров позиций)
- Fill: *C3^0sfir3+I+Act+Imperf+3+F*

Пример работы

- *sfr+I+Act+Imperf+3+F*
- Input: *C3^sfr+I+Act+Imperf+3+F* (вставка метки класса)
- PosInsertion: *C3^0s1f2r3+I+Act+Imperf+3+F* (вставка маркеров позиций)
- Fill: *C3^0sfir3+I+Act+Imperf+3+F*
- Prefix: *C3^tasfir3+I+Act+Imperf+3+F*

Пример работы

- *sfr+I+Act+Imperf+3+F*
- Input: *C3^sfr+I+Act+Imperf+3+F* (вставка метки класса)
- PosInsertion: *C3^0s1f2r3+I+Act+Imperf+3+F* (вставка маркеров позиций)
- Fill: *C3^0sfir3+I+Act+Imperf+3+F*
- Prefix: *C3^tasfir3+I+Act+Imperf+3+F*
- Suffix: *C3^tasfiru+I+Act+Imperf+3+F*

Пример работы

- *sfr+I+Act+Imperf+3+F*
- Input: *C3^sfr+I+Act+Imperf+3+F* (вставка метки класса)
- PosInsertion: *C3^0s1f2r3+I+Act+Imperf+3+F* (вставка маркеров позиций)
- Fill: *C3^0sfir3+I+Act+Imperf+3+F*
- Prefix: *C3^tasfir3+I+Act+Imperf+3+F*
- Suffix: *C3^tasfiru+I+Act+Imperf+3+F*
- Cleanup: *tasfiru*