Интерактивный конвертер *Chipollino* для наглядного изучения теории автоматов

ИУ9 МГТУ им. Н.Э. Баумана

Докладчик:
Дельман Александр

Разработчки:
А. Д. Дельман, А. М. Ильин, Д. П. Князихин, Э. А. Макаров, А. Н. Непейвода,
А. С. Терентьева, М. Р. Терюха, А. А. Чибизова, К. К.Шевченко

Мотивация

В свободном доступе представлено малое количество наглядных примеров, иллюстрирующих понятия теории автоматов

Студенту сложно разобраться в казалось бы несложной и красивой теории

Какие готовые решения удалось найти

Отдельные инструменты имеют такой функционал:

- Конвертер: регулярное выражение → автомат Томпсона, детерминизация, минимизация
- Конвертер: конечный автомат → регулярное выражение
- Проверка того, что регулярные выражения описывают один язык
- Парсинг слова по регулярному выражению или автомату

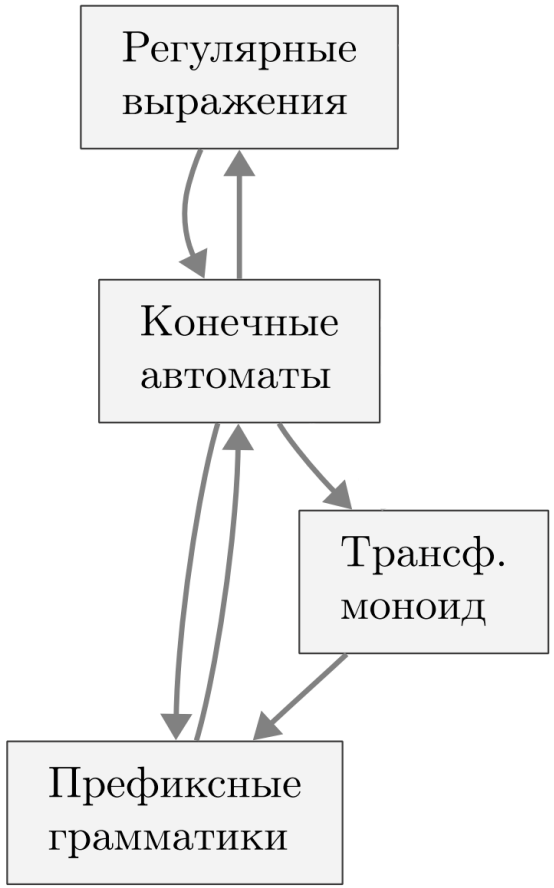
Цели

- Создание интерактивного справочника для изучения теории автоматов

- Охват малораспространенных алгоритмов функционалом конвертера

Возможности конвертера

Представления регулярных языков:



Функции преобразователя:

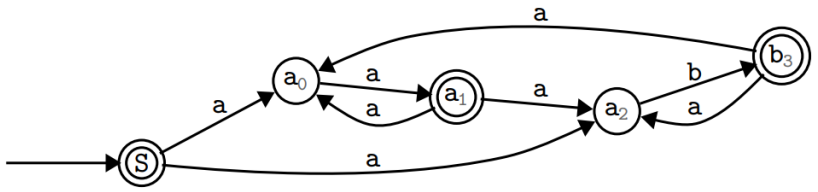
- Конвертация различных представлений регулярных языков
- Операции с НКА и ДКА: детерминизация, минимизация, дополнение, обращение и т.д.
- Определение меры неоднозначности КА
- Проверка языка на 1-однозначность
- Поиск нижней оценки числа состояний НКА
- Поиск длины накачки языка
- Прочие функции, определяющие свойства представлений

Тестирование

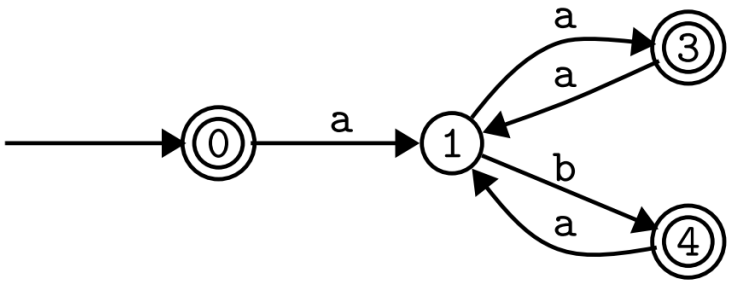
```
A = Glushkov {(aa|ab)*}
B = PGtoNFA.PrefixGrammar Glushkov {(aa|ab)*}
Equiv A B
```

Предикат `Equiv :: ⟨NFA, NFA⟩ → Bool`

Первый автомат:



Второй автомат:



Результат проверки на эквивалентность языков: 1

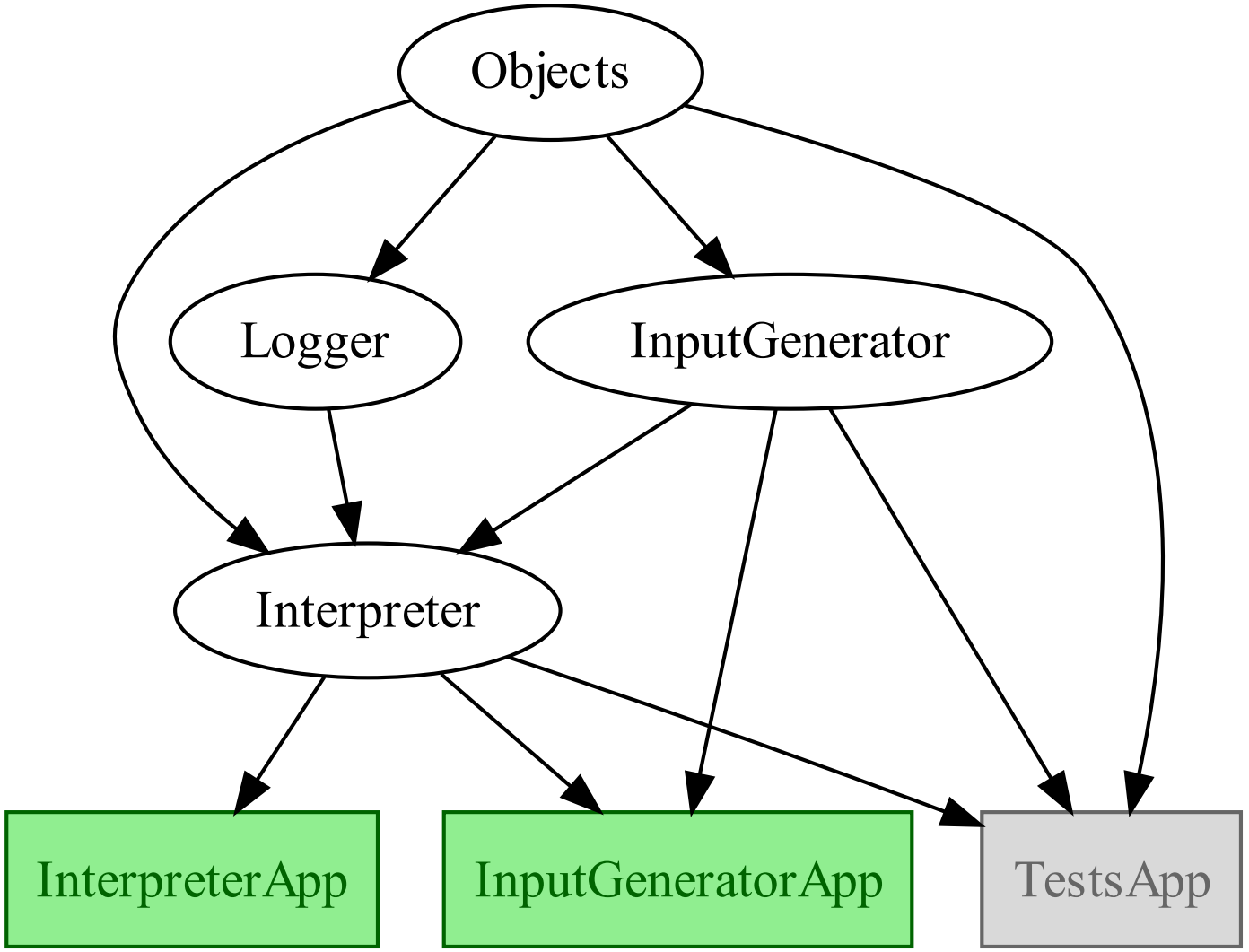
Минимальные автоматы сохранены в кэше

Тестирование и генератор входных данных

Пример случайно сгенерированных инструкций конвертеру:

Test {b(b(((a(b(a*)bb(|b))*)))))} {a*} 1*
N1 = Antimirov {(b|bb*(((b)a|a))b)(|||(|(|((a))))*)} !!*
SemDet N1
N2 = N1
N3 = Ambiguity.DeAnnote.RemEps.MergeBisim.Antimirov {a|} !!*
N4 = Reverse.Complement.Minimize.Antimirov {|bb|} !!
N5 = RemEps.Minimize.MergeBisim.RemEps N1 !!
N6 = GlaisterShallit.Determinize.PGtoNFA.PrefixGrammar N4 !!
Subset N4 N5
N7 = Ambiguity N4 !!
Test N1 {|||((b))*(a*|(((((b)))bab(a(a)))))} 5*
N8 = Ambiguity.Difference N4 N2 !!
N9 = MergeBisim.Difference N4 N5 !!
N10 = Minimize.Determinize N5 !!
N11 = ClassCard N10 !!

Структура проекта

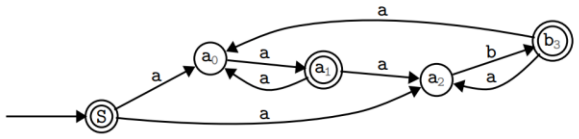


Генерация логов

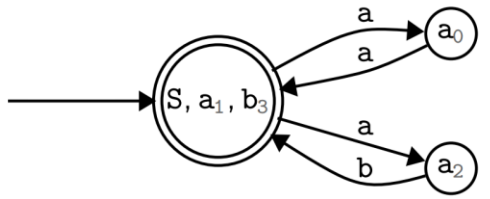
```
\section{MergeBisim}
\begin{frame}{Пример слияния по бисимуляции}
    Исходный автомат:

    %template_oldautomaton

    Итоговый автомат:

    %template_result

    Классы эквивалентности по бисимуляции:

    %template_equivclasses

\end{frame}
```

Преобразование `MergeBisim :: NFA → NFA`

Исходный автомат:



Итоговый автомат:



Классы эквивалентности по бисимуляции:
{S, a_1, b_3}; {a_0}; {a_2};

Как пользоваться конвертером

Команды интерпретатора:

- *Присвоение переменной значения*

```
A = Complement.Annote (Glushkov {(aa|ab)*}) !!
B = States.Reverse A
```

- *Команда test*

```
Test {(aa)*} {a*} 3
Test (Glushkov {(aa)*}) {a*} 1
```

- *Предикат*

```
Equal (Antimirov {ab|a}) (Glushkov {ab|a})
Subset {a*ba*} {aba*}
```

- *Верификатор гипотез*

```
Verify (Equal (Ambiguity.Glushkov.Arden.Glushkov *)
              (Ambiguity.Glushkov *))
```

Флаги интерпретатора:

- Слабой типизации
- Режима отображения теории

Примеры сгенерированных отчетов

Логи

Лучшая команда разработчиков по ТФЯ

2023 г.

Преобразование `Linearize` :: `Regex` → `Regex`

Определение

Если регулярное выражение $r \in \mathcal{RE}$ содержит n вхождений букв алфавита Σ, тогда линеаризованное регулярное выражение `Linearize`(r) получается из r приписыванием i-ой по счёту букве, входящей в r, индекса i.

Регулярное выражение до преобразования: $(aa \mid ab)^*$

Линеаризованное регулярное выражение: $(a_0a_1 \mid a_2b_3)^*$

Множества First, Last, Follow

Определение

Пусть $r \in \mathcal{RE}$, тогда:

- множество First — это множество букв, с которых может начинаться слово из $\mathscr{L}(r)$ (если $\varepsilon \in \mathscr{L}(r)$, то оно формально добавляется в First);
- множество Last — это множество букв, которыми может заканчиваться слово из $\mathscr{L}(r)$;
- множество Follow(c) — это множество букв, которым может предшествовать c. Т.е. $\{d \in \Sigma \mid \exists w_1, w_2 (w_1 c d w_2 \in \mathscr{L}(r))\}$.

Achtung!

Множество Follow в теории компиляции обычно определяется иначе — это множество символов, которые могут идти за выводом из определённого нетерминального символа. Два этих определения можно унифицировать, если рассматривать каждую букву в r как «обёрнутую» (в смысле, например, н.ф. Хомского).

First, Last, Follow — пример

Построим указанные множества для регулярного выражения $r = (ba \mid b)aa(a \mid ab)^*$.
Начнём с исходного регулярного выражения.

Исходное регулярное выражение

- $\mathrm{First}(r) = \{b\}$.
- $\mathrm{Last}(r) = \{a, b\}$.
- $\mathrm{Follow}_r(a) = \{a, b\}$; $\mathrm{Follow}_r(b) = \{a\}$.

Хотя данные множества описывают, как устроены слова из $\mathscr{L}(r)$ локально, однако они не исчерпывают всей информации о языке, поскольку разные вхождения букв в регулярное выражения никак не различаются.
Например, по множествам First и Last можно предположить, что $b \in \mathscr{L}(r)$, хотя это не так.

First, Last, Follow — пример

Построим указанные множества для регулярного выражения $r = (ba \mid b)aa(a \mid ab)^*$.
Вспомним, что $r_{\mathsf{Lin}} = (b_1a_2 \mid b_3)a_4a_5(a_6 \mid a_7b_8)^*$.

Линеаризованное выражение

- $\mathrm{First}(r_{\mathsf{Lin}}) = \{b_1, b_3\}$.
- $\mathrm{Last}(r_{\mathsf{Lin}}) = \{a_5, a_6, b_8\}$.
- $\mathrm{Follow}_{r_{\mathsf{Lin}}}(b_1) = \{a_2\}$; $\mathrm{Follow}_{r_{\mathsf{Lin}}}(a_2) = \{a_4\}$; $\mathrm{Follow}_{r_{\mathsf{Lin}}}(b_3) = \{a_4\}$; $\mathrm{Follow}_{r_{\mathsf{Lin}}}(a_4) = \{a_5\}$; $\mathrm{Follow}_{r_{\mathsf{Lin}}}(a_5) = \{a_6, a_7\}$; $\mathrm{Follow}_{r_{\mathsf{Lin}}}(a_6) = \{a_6, a_7\}$; $\mathrm{Follow}_{r_{\mathsf{Lin}}}(a_7) = \{b_8\}$; $\mathrm{Follow}_{r_{\mathsf{Lin}}}(b_8) = \{a_6, a_7\}$.

В описании данных множеств содержится исчерпывающая информация о языке $\mathscr{L}(r_{\mathsf{Lin}})$.

Конструкция автомата Глушкова

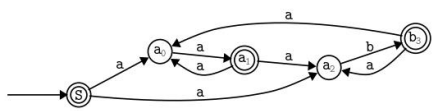
Алгоритм построения `Glushkov`(r)

- Строим линеаризованную версию r: $r_{\mathsf{Lin}} = \mathtt{Linearize}(r)$.
- Находим $\mathrm{First}(r_{\mathsf{Lin}})$, $\mathrm{Last}(r_{\mathsf{Lin}})$, а также $\mathrm{Follow}_{r_{\mathsf{Lin}}}(c)$ для всех $c \in \Sigma_{r_{\mathsf{Lin}}}$.
- Все состояния автомата, кроме начального (назовём его S), соответствуют буквам $c \in \Sigma_{r_{\mathsf{Lin}}}$.
- Из начального состояния строим переходы в те состояния, для которых $c \in \mathrm{First}(r_{\mathsf{Lin}})$. Переходы имеют вид $S \xrightarrow{c} c$.
- Переходы из состояния c соответствуют элементам d множества $\mathrm{Follow}_{r_{\mathsf{Lin}}}(c)$ и имеют вид $c \xrightarrow{d} d$.
- Конечные состояния — такие, что $c \in \mathrm{Last}(r_{\mathsf{Lin}})$, а также S, если $\varepsilon \in \mathscr{L}(R)$.
- Теперь стираем разметку, построенную линеаризацией, на переходах автомата. Конструкция завершена.

Построение `Glushkov` :: `Regex` → `NFA`

Регулярное выражение: $(aa \mid ab)^*$

Автомат:



Множество First:
a_0 a_2

Множество Last:
a_1 b_3

Множество Follow:
(a_0, a_1) (a_1, a_0) (a_1, a_2) (a_2, b_3) (b_3, a_0) (b_3, a_2)

Свойства автомата Глушкова

- Не содержит ε-переходов.
- Число состояний равно длине регулярного выражения (без учёта регулярных операций), плюс один (стартовое состояние).
- В общем случае недетерминированный.

Примечание

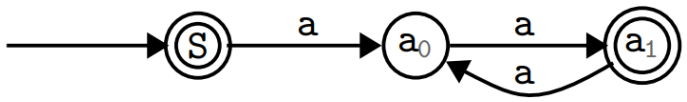
Для 1-однозначных регулярных выражений r автомат `Glushkov`(r) является детерминированным. Эту его особенность активно используют в современных библиотеках регулярных выражений, например, в RE2.
Выигрыш может получиться колоссальным: например, `Thompson`$((a^*)^*)$ является экспоненциально неоднозначным, а `Glushkov`$((a^*)^*)$ однозначен и детерминирован!

linearised regex:
(a.0a.1|a.2b.3)*

Примеры сгенерированных отчетов

Метод `Test` :: $\langle$`NFA, Regex`$\rangle \rightarrow$ `VOID`

Автомат:



Слова порождаются регуляркой a*

Шаг итерации: 5

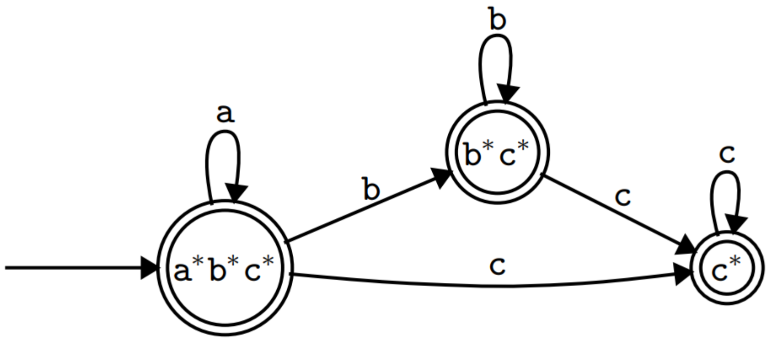
	Количество итераций	Длина строки	Время парсинга	Принадлежность языку
1	5	0	0.000000	true
2	5	5	0.001000	false
3	5	10	0.001000	true
4	5	15	0.002000	false
5	5	20	0.003000	true
6	5	25	0.004000	false
7	5	30	0.004000	true
8	5	35	0.005000	false
9	5	40	0.006000	true
10	5	45	0.006000	false
11	5	50	0.007000	true
12	5	55	0.007000	false
13	5	60	0.008000	true

Примеры сгенерированных отчетов

Построение `Antimirov :: Regex → NFA`

Регулярное выражение: $a^*b^*c^*$

Автомат Антимирова:



Частичные производные:

$a(a^*b^*c^*) = a^*b^*c^*$
$b(a^*b^*c^*) = b^*c^*$
$c(a^*b^*c^*) = c^*$
$b(b^*c^*) = b^*c^*$
$c(b^*c^*) = c^*$

$c(c^*) = c^*$
$a(a^*b^*c^*) = a^*b^*c^*$
$b(a^*b^*c^*) = b^*c^*$
$c(a^*b^*c^*) = c^*$
$b(b^*c^*) = b^*c^*$
$c(b^*c^*) = c^*$

$c(c^*) = c^*$
$a(a^*b^*c^*) = a^*b^*c^*$
$b(a^*b^*c^*) = b^*c^*$
$c(a^*b^*c^*) = c^*$
$b(b^*c^*) = b^*c^*$
$c(b^*c^*) = c^*$
$c(c^*) = c^*$

`MyhillNerode :: NFA → Int`

Автомат:

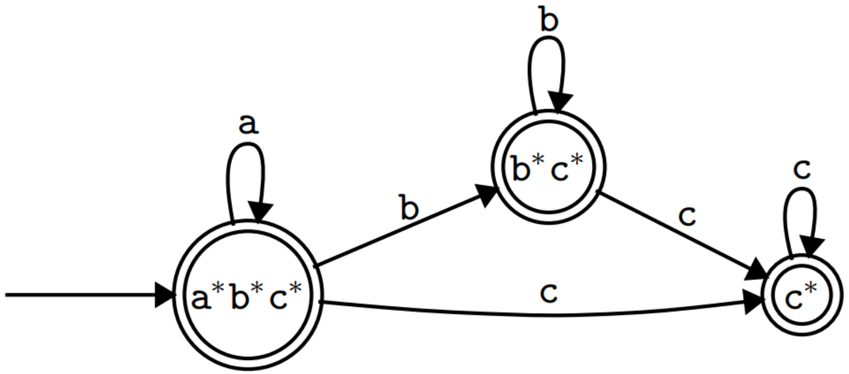


Таблица классов эквивалентности:

	ε	a	b	ba
ε	1	1	1	0
b	1	0	1	0
c	1	0	0	0

Итоговое число классов эквивалентности: 3

Применение в учебе

Верификатор гипотез

Гипотеза:
$(\texttt{Equal}(\texttt{RemEps.Thompson}^*)(\texttt{Glushkov}^*))$

Процент успешных проверок:
100%

Верификатор гипотез

Гипотеза:
$(\texttt{Equal}(\texttt{MergeBisim.Glushkov}^*)(\texttt{IlieYu}^*))$

Процент успешных проверок:
60%

Контрпримеры:
$(|\ \texttt{a}\ |\ \texttt{ab}^*\texttt{b}^*)(\texttt{a}\ |\ \texttt{b}^*\texttt{ab})$

$\texttt{aaaa}(\texttt{ba}^*\texttt{a}^*\texttt{a})^*$

Итоги

Результат нашей работы в цифрах:

- 19000+ строк кода
- 1200+ коммитов
- 100+ закрытых issue
- 100+ закрытых пулл-реквестов
- 40+ реализованных функций преобразователя

Материалы

- Документация:
github.com/StarikTenger/Chipollino/wiki
- Репозиторий проекта:
github.com/StarikTenger/Chipollino

Планы на будущее

- Подробные логи для всех функций
- REPL-среда
- Расширение языка регулярок перечислениями и интервалами, а также отрицанием
- Оптимизация некоторых алгоритмов
- Возможный переход к другим форматам сгенерированных отчётов
- Использование метаданных для масштабирования проекта

Спасибо за внимание!