Использование автоматов в интерпретаторе MACASM

Соколова Полина

Научный руководитель: ст. преп. кафедры СП, к. ф.-м. н. Луцив Д. В.

Консультант: ст. преп. кафедры СП Баклановский М. В.

Рецензент: главный научный сотрудник Университета ИТМО,
докт. техн. наук, профессор Шалыто А. А.

2020

Многоуровневая архитектура кода

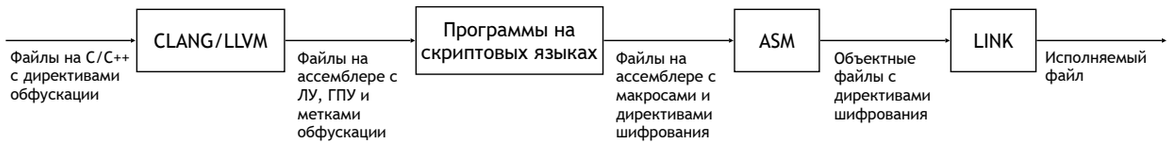


Схема трансляции программы в МАК

```
{BLOCK bool_expr}
        mov rax, 1
{END}
```

Пример программы на Macer

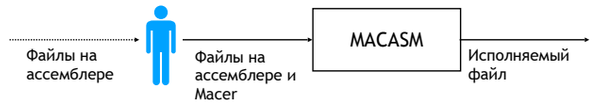


Схема трансляции программы в МАК с использованием MACASM

Многоуровневая архитектура кода (МАК) - программное обеспечение для обфускации, нанесения цифровых водяных знаков на код и многого другого. Сейчас задачи МАК делаются при помощи программ на скриптовых языках и могут выполняться десятками минут, потому что мы работаем с текстом. Для того, чтобы задачи МАК выполнялись быстрее, нужно иметь возможность работать и с текстом, и с кодом одновременно. Например, цифровые водяные знаки наносятся на код, а поиск следующей инструкции в тексте небыстрый. С другой стороны, в ассемблере есть макросы, но это слабый инструмент для текстового процессинга, поэтому возникла потребность в написании своего ассемблера MACASM с внутренним языком Macer.

Введение

- Автоматный массив (автомассив)
- Регулярные выражения
- Синтаксический анализатор
- Распознавание служебных слов и идентификаторов
- Минимизация ДКА

Проблема быстрого поиска переменной в списке является одной из главных внутренних задач в системах с именной адресацией переменных. Для её решения используются ассоциативные массивы. Ассоциативный массив — это словарь (для хранения ключей) + массив (для хранения значений). Автоматный массив — это придуманный нами термин для обозначения автоматной реализации ассоциативного массива с хранением словаря в автомате.

Разработка системы начинается с реализации ассоциативных массивов в виде автомассивов. Автоматная техника является достаточно скоростной, и автомассивы — это только часть возможного её применения. Таким образом, можно сделать шаг в сторону других модулей интерпретатора, использующих конечные автоматы: работа с регулярными выражениями, синтаксический разбор языка Macer и распознавание служебных слов и идентификаторов при лексическом анализе. Также для уменьшения занимаемого места автомат бывает удобно минимизировать, то есть преобразовать в эквивалентный с минимальным количеством состояний.

Постановка задачи

Разработка модулей интерпретатора MACASM, использующих конечные автоматы

- Анализ популярных реализаций ассоциативных массивов и алгоритмов минимизации ДКА
- Реализация ассоциативного массива в виде автоматного массива
- Разработка утилиты построения минимального ДКА по словарю
- Реализация модуля для построения ДКА по регулярному выражению
- Реализация модуля для синтаксического разбора языка Macer
- Апробация созданных средств

Целью данной работы является разработка модулей интерпретатора MACASM, использующих конечные автоматы. Для достижения этой цели были поставлены следующие задачи:

- анализ популярных реализаций ассоциативных массивов и алгоритмов минимизации ДКА;
- реализация ассоциативного массива в виде автоматного массива;
- разработка утилиты построения минимального ДКА по словарю;
- реализация модуля для построения ДКА по регулярному выражению;
- реализация модуля для синтаксического разбора языка Macer;
- апробация созданных средств.

Минимизация ДКА

- Алгоритм Глушкова (1962)
  - автоматы Мура и Мили
- Алгоритм Хопкрофта (1971)
- Алгоритм Дацюка-Михова (2000)
  - инкрементальные построение и псевдоминимизация

До сих пор лучшим результатом среди алгоритмов минимизации ДКА остается алгоритм Хопрофта, предложенный в 1971 году. В последнее время исследования в основном сосредоточены на постепенном построении и сжатии автомата на лету, например, алгоритм Дацюка-Михова. Кроме того, существуют алгоритмы минимизации автоматов Мили и Мура, например, алгоритм Глушкова, и множество других алгоритмов.

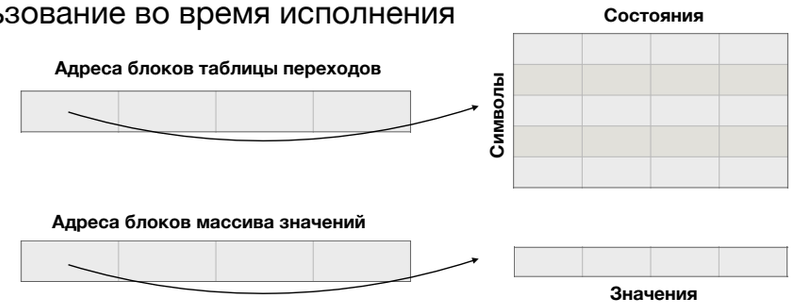
Используемые методы

- Автоматная техника реализации словаря
- Алгоритм Хопкрофта минимизации ДКА
- Блочный разбор и Visibly Pushdown Automaton (VPA)
- Алгоритм Мак-Нотона—Ямады—Томпсона для преобразования регулярного выражения в НКА
- Построение подмножества (subset construction) ДКА из НКА

В реализации словаря используется автоматная техника, для минимизации автомата - алгоритм Хопкрофта. Для разбора языка регулярных выражений и языка Macer используется техника блочного разбора и Visibly Pushdown Automaton (VPA). VPA — это конечный автомат, который использует стек только для определения начала и конца блока. Алгоритм Мак-Нотона—Ямады—Томпсона и построение подмножества используются для преобразования регулярных выражений в автоматы.

Автоматный массив

- Использование ДКА для хранения словаря
- Нет перехеширования, как у хеш-таблиц
- Использование во время исполнения



Блоки в автомассиве

Словарь в автомассиве хранится в детерминированном конечном автомате, который строится инкрементальной при добавлении нового слова. В автомассиве есть 2 типа блоков: таблица переходов (для хранения ключей) и массив значений (для хранения значений). В данной реализации нет такой проблемы, как перехеширование, поскольку все данные лежат не в одной области памяти (блоки), а сразу в нескольких. Изначально выделяется 1 блок, как только он заканчивается — выделяется новый, и мы храним адреса всех таких выделенных блоков. Автомассивы реализованы для использования во время исполнения программы.

Построение минимального ДКА по словарю

- Использование во время разработки интерпретатора
- Вход: список слов с атрибутами
- Выход: ассемблерный файл с минимальным ДКА
- Вставка в автомассив и минимизация ДКА по алгоритму Хопкрофта
- Вторая версия без минимизации

Поскольку автоматная техника является скоростной, было решено применять её и в других частях интерпретатора. Утилита построения минимального автомата по словарю предназначена для разработчиков интерпретатора MACASM и использования во время разработки. На вход она получает список служебных слов с атрибутами (например, токенами или номерами слов). Работа над интерпретатором ведётся на языке ассемблера, поэтому утилита возвращает ассемблерный файл (с минимальным автоматом), который можно подключить и использовать в своей работе.

В основе данной утилиты лежит реализация автомассива. На первом этапе происходит добавление слов в автомассив, то есть строится таблица переходов автомата. Далее символы объединяются в группы, чтобы не хранить пустые или похожие строки. После этого автомат минимизируется алгоритмом Хопкрофта. Также сделана вторая версия утилиты без минимизации автомата.

Построение ДКА по регулярному выражению

- Использование во время исполнения
- Разбор VPA и построение НКА
- Преобразование в ДКА
- Минимизация ДКА

Модуль преобразования регулярного выражения в автомат создан для использования во время исполнения программы. Данный модуль разбирает язык регулярных выражений VPA и одновременно строит по нему НКА (недетерминированный конечный автомат). Далее он преобразует его в ДКА и минимизирует по алгоритму Хопкрофта.

Синтаксический разбор языка Macer

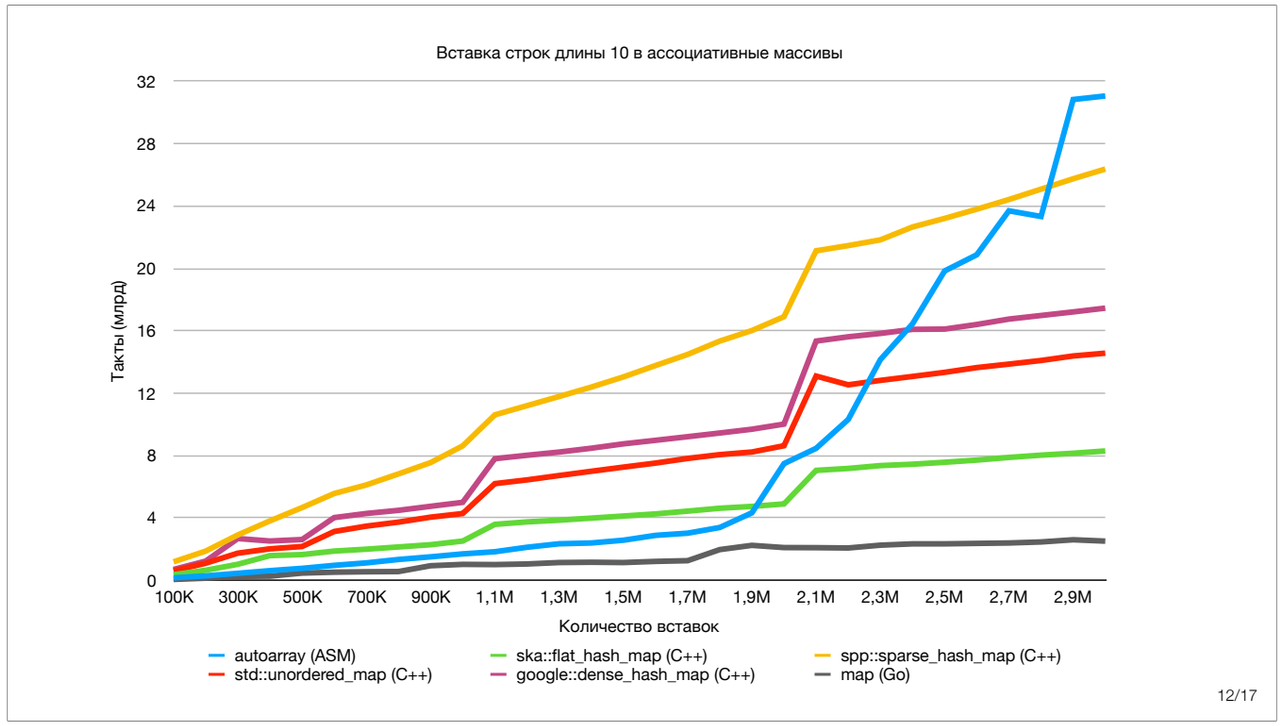
- Вход: программа на Macer (в фигурных скобках) и на языке ассемблера (снаружи)
- Фигурные скобки
- Лексер
  - автомат для разбора служебных слов
- Парсер
  - VPA

На вход синтаксическому анализатору поступает программа на языке Macer в фигурных скобках, на языке ассемблера — снаружи. Основной анализатор состоит из трех автоматов: первый контролирует фигурные скобки, второй — это лексер, его частью является автомат для разбора служебных слов, сгенерированный утилитой (о ней говорилось ранее), третий — это парсер в виде VPA. По окончании разбора выдается «OK» или тип и место ошибки.

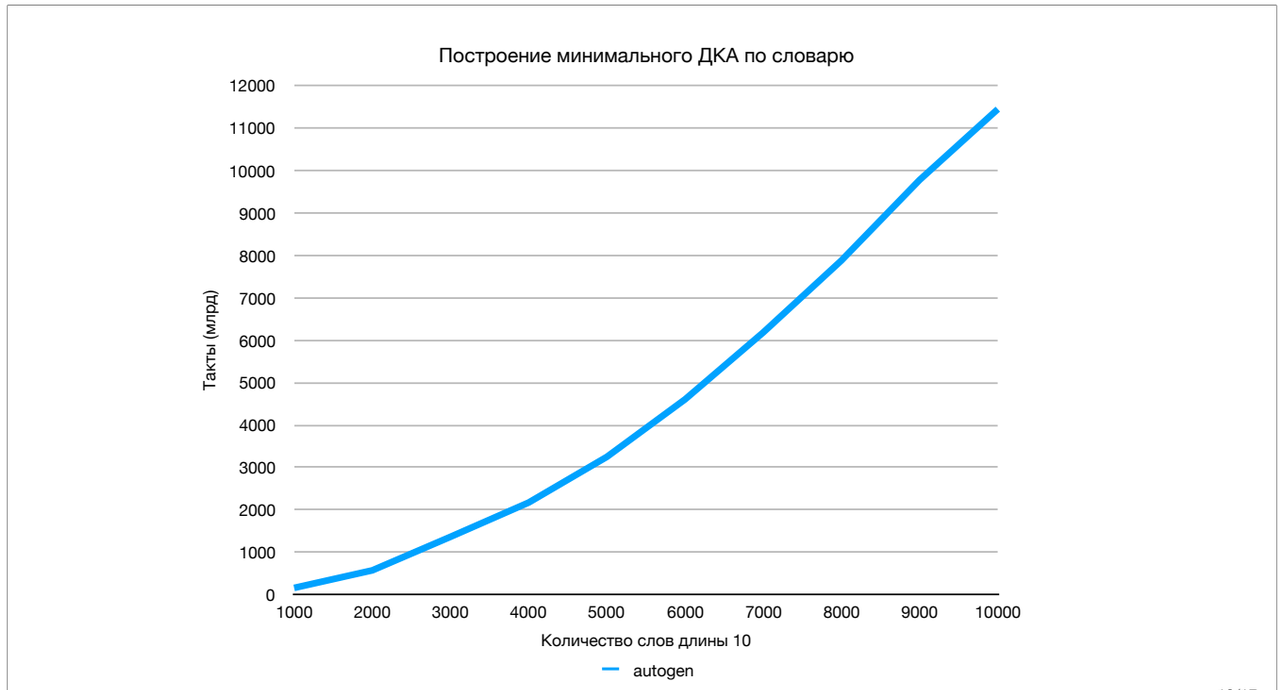
Хеш-таблицы в C++ и Go

	Разрешение коллизий	Пробирование (probing)	Особенности
ska::flat_hash_map (C++)	открытая адресация	линейное	хеширование Робина Гуда, ограничение на количество проб, самая быстрая хеш-таблица в C++
spp::sparse_hash_map (C++)	открытая адресация	квадратичное	—
google::dense_hash_map (C++)	открытая адресация	квадратичное	—
std::unordered_map (C++)	цепочки	—	—
map (Go)	цепочки	—	8 пар ключ-значение в блоке (bucket)

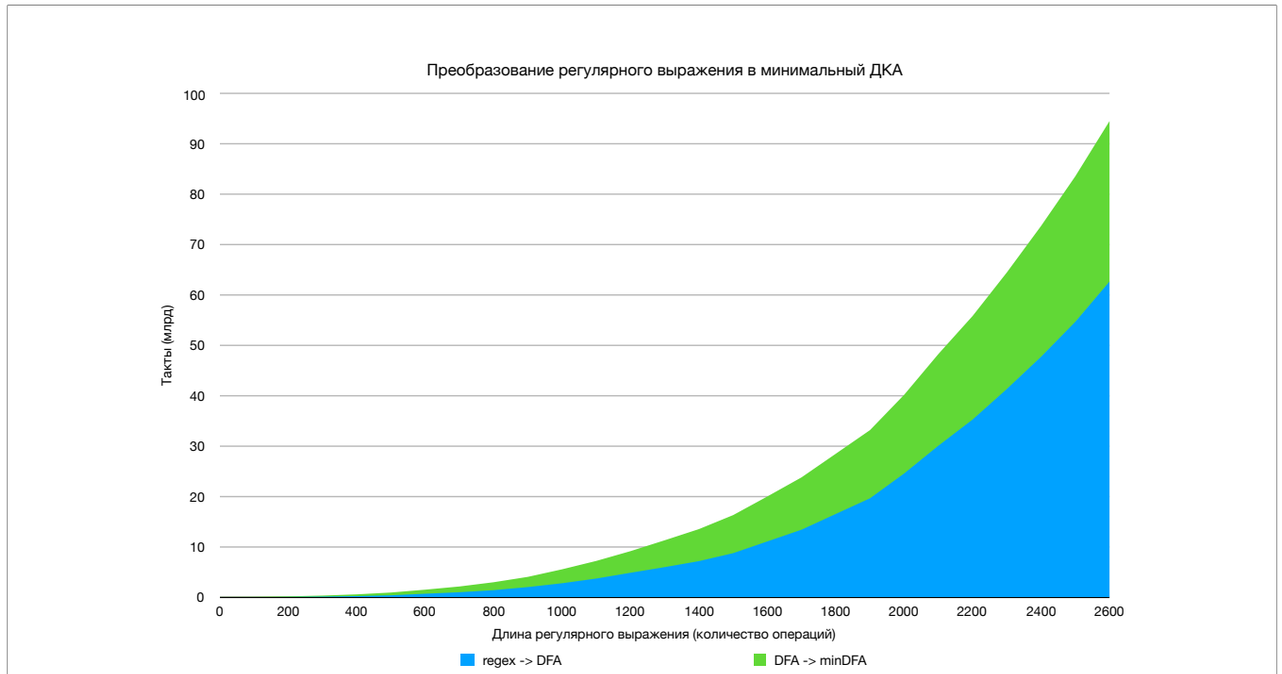
На слайде представлено сравнение популярных реализаций хеш-таблиц в языках C++ и Go. Далее будет приведено сравнение скорости их работы.



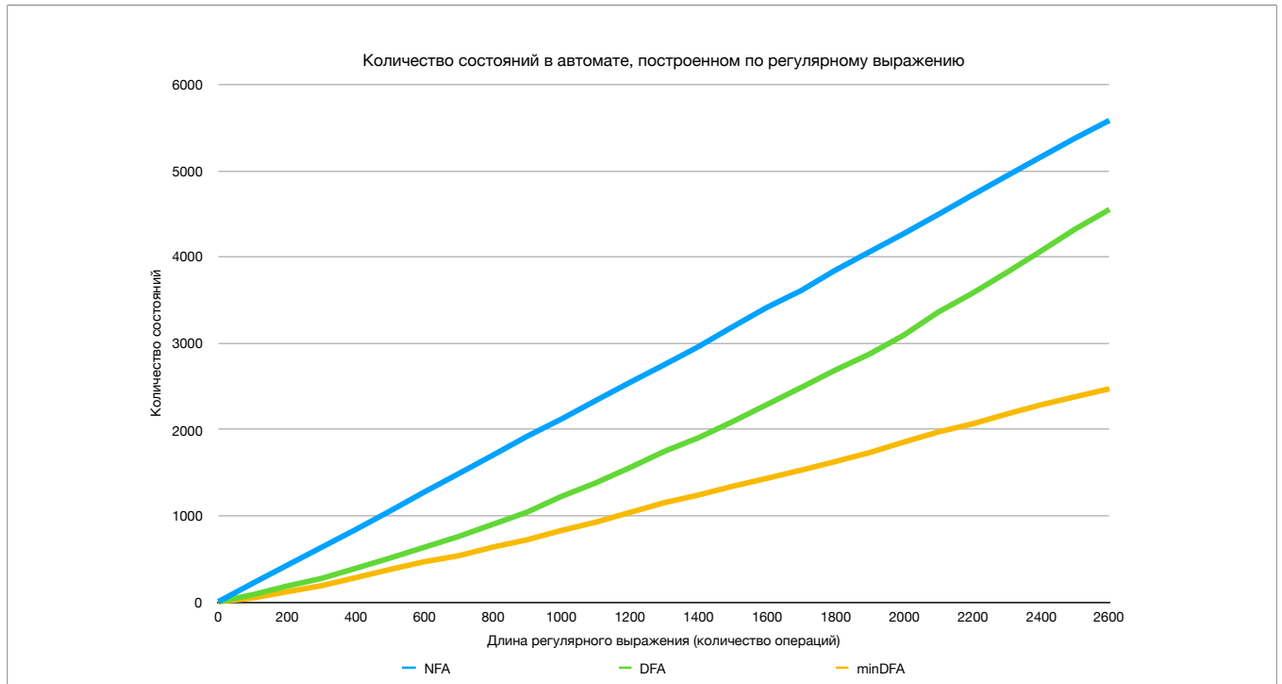
Здесь и далее можно примерно считать 1 млрд тактов равным 1 секунде. На слайде представлено сравнение скорости автомассива (autoarray) с промышленными реализациями. Важно отметить, что мы не затратили никаких усилий на оптимизацию, это наивная исследовательская реализация в лоб с тривиальным управлением памятью. Уже сейчас до нашей границы (1,8 млн вставок строк) автомассив в 1,5-2 раза обгоняет самую быструю хеш-таблицу в C++ (ska::flat_hash_map).



На слайде представлен график времени работы утилиты построения минимального автомата по словарю. Алгоритм Хопкрофта небыстрый: для словаря размером 2000 слов автомат будет строиться около 5 минут. На этапе построения интерпретатора это не так важно.



На слайде показан график времени выполнения модуля преобразования регулярного выражения в минимальный ДКА. Минимизация здесь занимает значительную часть времени, это особенно заметно на небольших регулярных выражениях (до 1 тыс).



На этом графике наблюдается линейная зависимость размера НКА от размера ДКА, хотя в теории говорится об экспоненциальной. Это связано с тем, что в построенном по регулярному выражению НКА совсем незначительное количество epsilon-переходов.

Апробация

- Автомассив (наивная исследовательская реализация)
  - Граница применимости: до 1,8 млн вставок строк
  - Обгоняет самую быструю хеш-таблицу в C++ в 1,5–2 раза
- Утилита построения минимального ДКА по словарю
  - Граница применимости: словари до нескольких тысяч слов
- Модуль построения ДКА по регулярному выражению
  - Минимизация занимает значительное время
  - Линейная зависимость размеров НКА от ДКА

По результатам тестирования определены границы применимости автомассива (1,8 млн строк) и утилиты построения минимального ДКА по словарю (несколько тысяч строк). Уже сейчас реализация автомассива в лоб обгоняет самую быструю хеш-таблицу в C++. Кроме того, поскольку в модуле преобразования регулярного выражения в автомат минимизация занимает значительное время, имеет смысл сделать её опциональной.

Результаты

- Проанализированы популярные реализации хеш-таблиц (map (Go) и 4 реализации в C++) и следующие алгоритмы минимизации ДКА: Хопкрофта, Глушкова, Дацюка-Михова
- Реализован ассоциативный массив в виде автоматного массива с хранением словаря в виде ДКА
- Разработана утилита построения ДКА по словарю служебных слов в двух вариантах: с минимизацией по алгоритму Хопкрофта (для больших словарей) и без минимизации автомата
- Реализован модуль для построения ДКА по регулярному выражению с разбором VPA и возможностью включения/выключения опции минимизации автомата
- Реализован модуль для синтаксического анализа языка Macer с разбором VPA
- Выполнена апробация созданных средств

В ходе данной работы были получены следующие результаты:
— проанализированы популярные реализации хеш-таблиц (map (Go) и 4 реализации в C++) и следующие алгоритмы минимизации ДКА: Хопкрофта, Глушкова, Дацюка-Михова;
— реализован ассоциативный массив в виде автоматного массива с хранением словаря в виде ДКА;
— разработана утилита построения ДКА по словарю служебных слов в двух вариантах (с минимизацией по алгоритму Хопкрофта и без минимизации автомата);
— реализован модуль для построения ДКА по регулярному выражению с разбором VPA и возможностью включения/выключения опции минимизации автомата;
— реализован модуль для синтаксического анализа языка Macer с разбором VPA;
— выполнена апробация созданных средств.