Компиляция сертифицированных F*-программ в робастные Web-приложения

Автор: Полубелова Марина Игоревна, 646 гр.
Научный руководитель: к. ф.-м. н., доцент Григорьев С. В.
Рецензент: разработчик ПО ООО “ИнтеллиДжей Лабс” Мордвинов Д. А.

Санкт-Петербургский государственный университет
Кафедра системного программирования

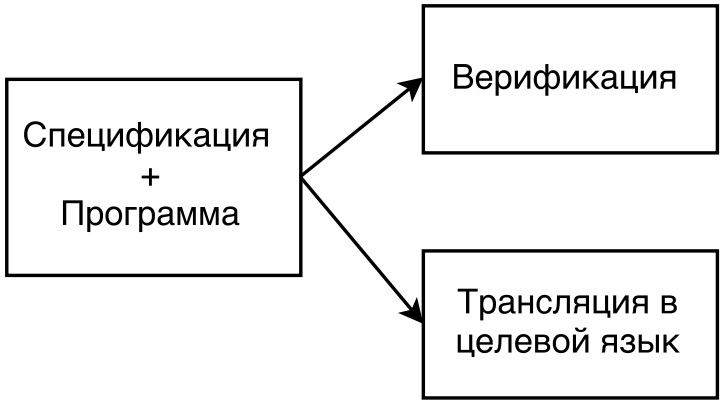
13 июня 2017г.

Криптографические протоколы

Разработка криптографических протоколов является актуальной и сложной задачей

- Протокол TLS используется для защищенной передачи данных между узлами в сети Интернет
  - Веб-браузеры
  - Электронная почта
  - Обмен мгновенными сообщениями
- Библиотека OpenSSL разрабатывается с 1998 года и все еще остается подвержена хакерским атакам
- Многие атаки используют ошибки, допущенные в программном обеспечении

Сертифицированное программирование



- Позволяет доказывать, что программа соответствует своему формальному описанию
- Инструменты для создания сертифицированных программ
  - Coq, Agda, F*, Idris и др.

Проект Everest

Целью проекта является создание высокопроизводительной, соответствующей спецификациям, реализации протокола HTTPS

- Проект miTLS посвящен верификации протокола TLS
- Проект HACL* посвящен верификации криптографических примитивов
- Язык программирования F*, ориентированный на верификацию программ
- KreMLin — компилятор из подмножества языка F* в C
- Язык программирования Vale и инструмент Dafny для создания и верификации криптографических примитивов на ассемблере

Язык F*

- Функциональный язык программирования, ориентированный на верификацию программ
- Обладает богатой системой типов
  - Уточняющие и зависимые типы
  - Эффект вычисления функции и т.д.
- Позволяет доказывать многие свойства программы
- **F* backends**
  - OCaml
  - C* $\rightarrow$ KreMLin $\rightarrow$ C

Существующие инструменты для компиляции программ на OCaml в JavaScript

- js_of_ocaml
  - Компиляция происходит на уровне байт-кода
  - Не предназначен для чтения сгенерированного кода
  - Многие OCaml-библиотеки содержат вставки кода на C
- BuckleScript
  - Компиляция происходит на уровне лямбда-выражений
  - Читабельность сгенерированного кода
  - Использование внутреннего представления для некоторых структур данных

Инструмент Flow

- Инструмент для статической проверки типов в JavaScript-программах
- Поддержана спецификация ECMAScript 6
- Почему не язык программирования TypeScript?
  - Flow имеет более точную и богатую систему типов, чем TypeScript
  - Для Flow: аннотация типов легко убирается из программы
  - Логика работы системы типов Flow близка к той, что используется в функциональных языках программирования

Постановка задачи

Цель: разработать инструмент для компиляции сертифицированных F*-программ в робастные Веб-приложения на JavaScript

- Сформулировать правила трансляции с языка F* на JavaScript, гарантирующие сохранение аннотаций типов
- Выполнить реализацию предложенного подхода
- Провести экспериментальное исследование реализованного инструмента

Описание подхода

$$\text{.fst} \xrightarrow{1} \text{F* AST} \xrightarrow{2} \text{ML AST} \xrightarrow{3} \text{Flow AST} \xrightarrow{4} \text{.flow} \xrightarrow{5} \text{.js}$$

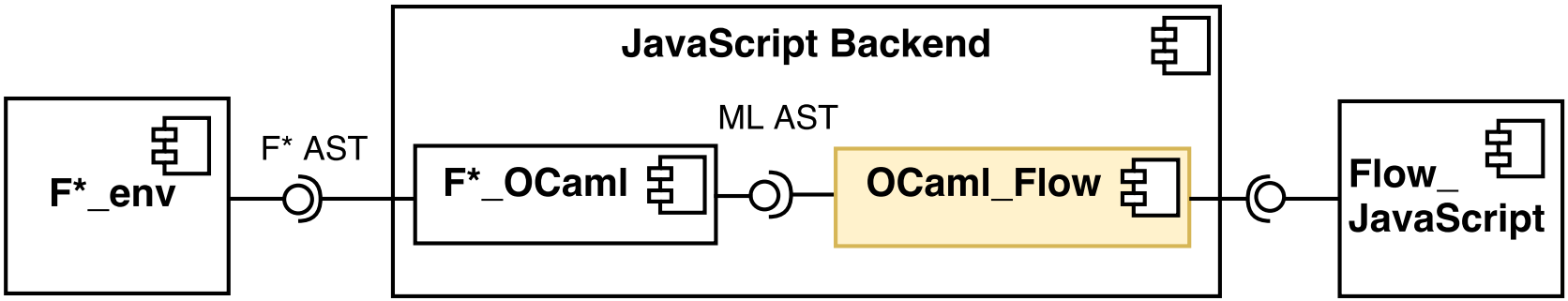
- **Шаг 1:** построение F* AST исходной программы
- **Шаг 2:** построение ML AST из F* AST (удаление зависимых и уточняющих типов, ghost-вычислений)
- **Шаг 3:** трансляция ML AST в Flow AST с сохранением аннотаций для типов
- **Шаг 4:** получение программы, которая может быть проверена инструментом Flow
- **Шаг 5:** преобразование Flow-программы в JavaScript-программу (удаляется информация о типах)

Правила трансляции языка OCaml в JavaScript

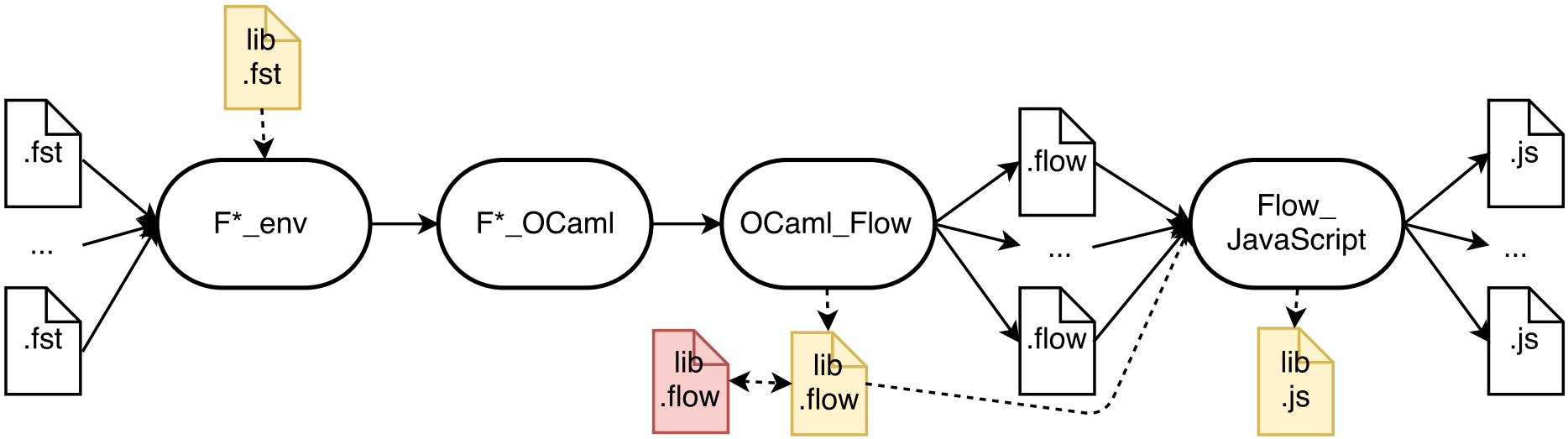
Сформулированы правила трансляции для подмножества языка OCaml в JavaScript

- Константы
- Выражения
- Выражения для сопоставления с образцом
- Типы

Архитектура инструмента



Процесс построения JavaScript-приложения из F*-программы



Экспериментальное исследование

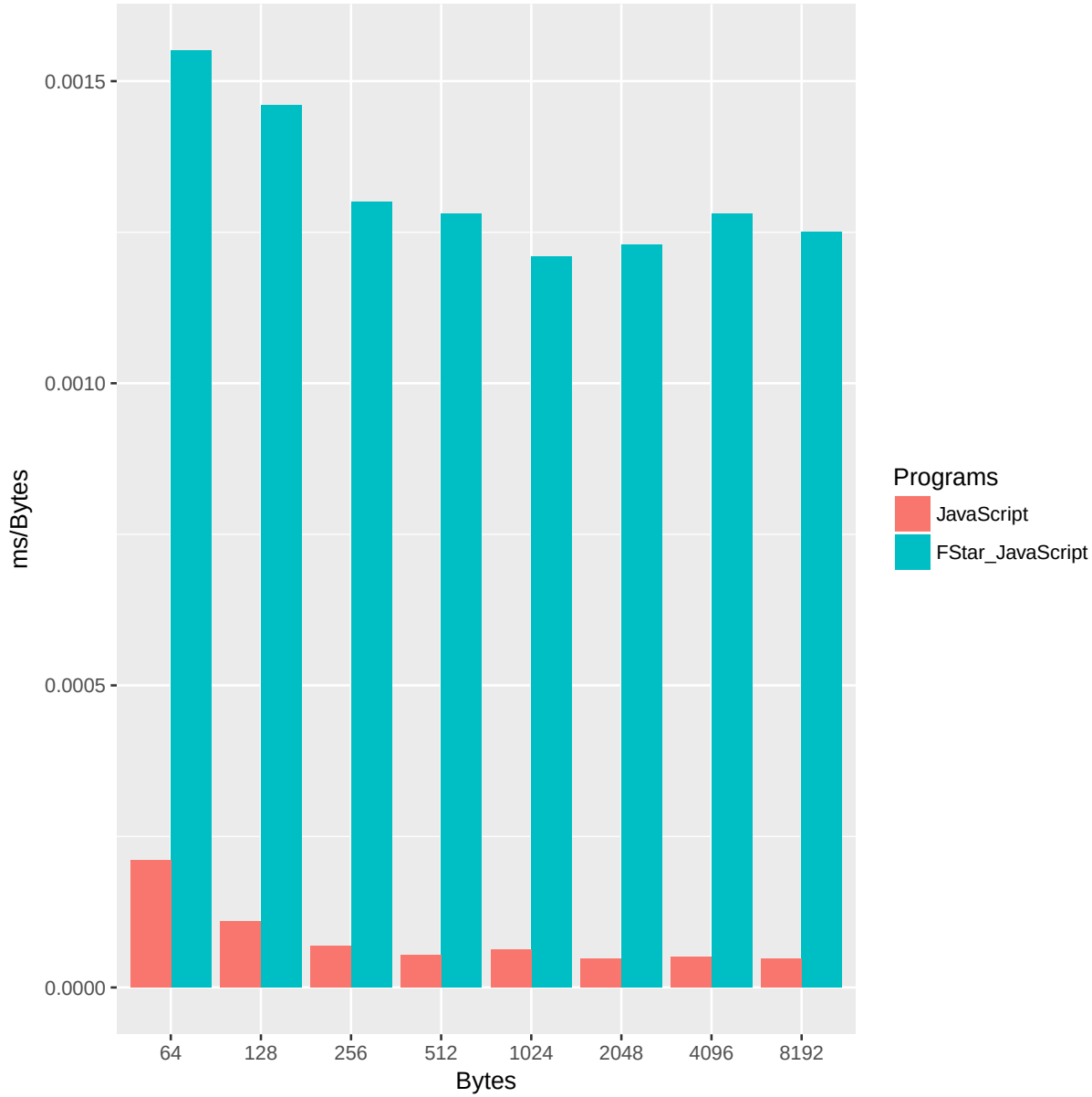
- Тестовые данные: реализация алгоритма ChaCha20 из библиотеки HACL*
- Для верификации алгоритма использовалось около 37 библиотечных и вспомогательных модулей
- Описательная характеристика основного модуля

	F*	F*_OCaml	F*_Flow
Кол-во верхнеуровневых функций	32	32	32
Кол-во строк кода	786	422	316

- Выполнялись замеры времени работы функции

$$\mathit{encrypt}\ \texttt{plaintext key nonce counter} \rightarrow \texttt{cyphertext}$$

График сравнения времени выполнения программ



Результаты

- Сформулированы правила трансляции с языка F* на JavaScript
- Выполнена реализация предложенного подхода на языке F*
- Проведено экспериментальное исследование реализованного инструмента на примерах из криптографической библиотеки HACL*