Предмет курса “Статический анализ ПО”

Марат Ахин Михаил Беляев

13 февраля 2018 г.

Введение

- Программное обеспечение — это **сложно**
- Насколько? Настолько, что поиск ошибок может занимать годы, десятилетия и, в некоторых случаях, вечность
- В программах случаются ошибки
- Программы могут работать медленно
- Нужны методы для того, чтобы судить о программах и их свойствах в отрыве от программиста

Для этого и служит анализ ПО

Анализ ПО

Анализ программного обеспечения, как и любой анализ, нужен для установления различных свойств ПО как объекта

- Является ли программа корректной?
- Соответствует ли она спецификации?
- Можно ли её оптимизировать?
- Насколько быстро она работает?
- Завершается ли она когда-нибудь?
- Соответствует ли она семантическим свойствам языка, на котором написана?
- Есть ли в ней мёртвый код?
- Является ли её результат детерминированным и от чего он зависит?
- ...

Кто мы такие

Кто мы?

Лаборатория верификации и анализа ПО кафедры КСПТ Политеха

`https://research.jetbrains.org/groups/pvalab`

`https://bitbucket.org/vorpal-research`

`https://github.com/vorpal-research`

Что мы знаем про анализ ПО?

Если вкратце, то достаточно.

`https://bitbucket.org/vorpal-research/borealis`

О чем этот курс?

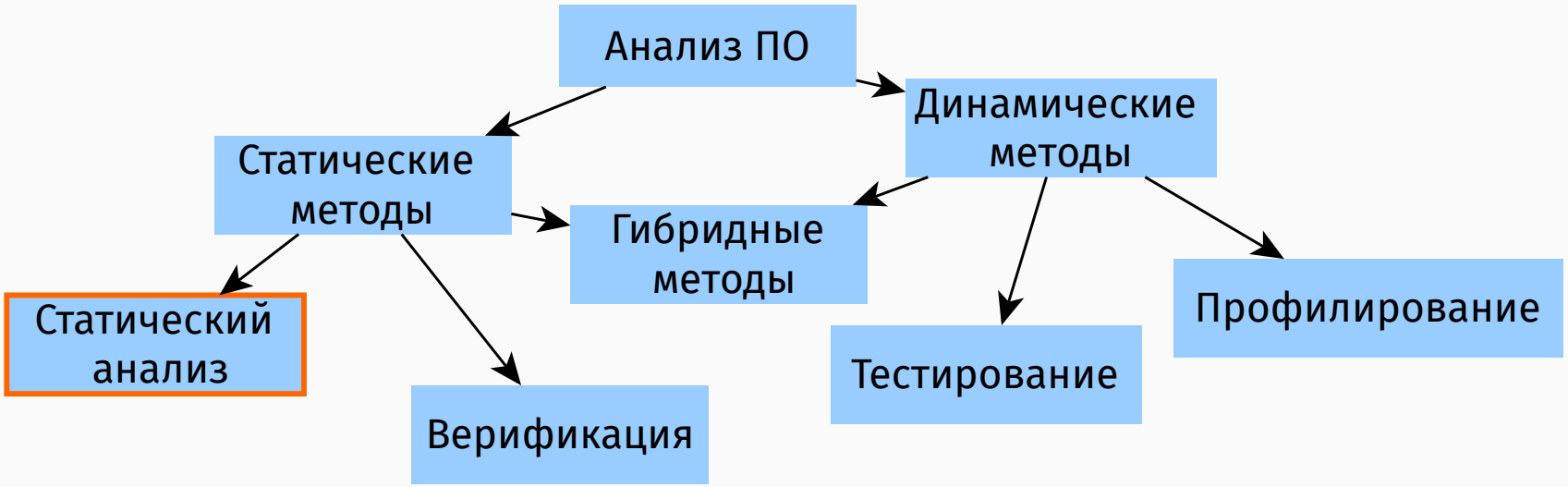
Этот курс про **статический анализ** ПО

Что это такое?

О чем этот курс?

Этот курс про **статический анализ** ПО

Что это такое?



Warning: классификаций много и все неверные, эта — не исключение

Что такое статический анализ?

Метод, который выполняется **статически**

- То есть, без запуска программы

Почему это важно?

- Динамические методы ничего вам не гарантируют
- Вы исследуете только те части программы, которые запустили
- Их нельзя использовать в процессе компиляции (нечего запускать)
- Бесконечные циклы существуют
- Некоторые свойства так проверить **вообще нельзя**
- К тому же, вы ими и так уже пользуетесь :-)

Любой ли статический метод — это статический анализ?

- Это терминологический вопрос
- Всё сложно
- Мы будем использовать ту терминологию, к которой привыкли

Статический анализ vs верификация ПО

Warning: субъективное восприятие терминологии авторами

Viewer discretion is advised

Верификация — формальное доказательство соответствия или несоответствия формального предмета верификации его формальному описанию. © Википедия

Статический анализ vs верификация ПО

Warning: субъективное восприятие терминологии авторами

Viewer discretion is advised

Верификация — формальное доказательство соответствия или несоответствия формального предмета верификации его формальному описанию. © Википедия

Ух, как всё **формально**

Верификация — это статический метод анализа ПО с целью **точного** определения каких-то свойств

Статический анализ vs верификация ПО

Warning: неформальные описания формальных (x3) вещей

Viewer discretion is advised

Метод верификации характеризуется двумя свойствами:

soundness и completeness

Soundness

Если вы доказали какое-то свойство, значит оно действительно было

Completeness

Если свойство имеет место, ваш метод может его доказать

Методы верификации от лучших домов Европы должны обладать и тем, и другим

Автоматическая верификация

Мы хотим всё с предыдущего слайда, и чтобы автоматически

Автоматическая верификация

Мы хотим всё с предыдущего слайда, и чтобы автоматически

Автоматическая верификация

Алан Тьюринг, 1936

Невозможно написать алгоритм, который говорит, завершается

Тьюринг-полная программа или нет

Автоматическая верификация

Алан Тьюринг, 1936

Невозможно написать алгоритм, который говорит, завершается Тьюринг-полная программа или нет

Генри Райс, 1952

Невозможно написать алгоритм, который определяет **любое** нетривиальное свойство Тьюринг-полной программы

Автоматическая верификация

Алан Тьюринг, 1936

Невозможно написать алгоритм, который говорит, завершается Тьюринг-полная программа или нет

Генри Райс, 1952

Невозможно написать алгоритм, который определяет **любое** нетривиальное свойство Тьюринг-полной программы


That's all Folks!

Что делать-то???

- Можно верифицировать Тьюринг-*неполную* модель

  Этим занимается **Model Checking**
- Можно дописывать недостающую информацию вручную

  См. **дедуктивную верификацию**
- Можно аппроксимировать (сверху, снизу или как получится)

  Вот об этом мы и будем тут говорить

Enter static analysis!

- Статический анализ — это всегда *аппроксимация*
- Он тоже обладает двумя свойствами:

  precision (точность) и **recall** (полнота).

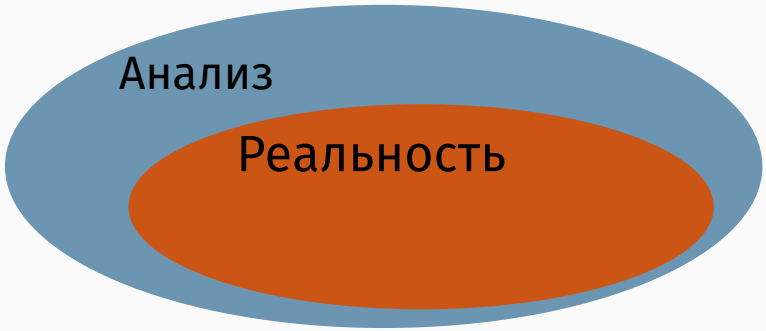
Precision

Доля истинных результатов среди всех полученных

Recall

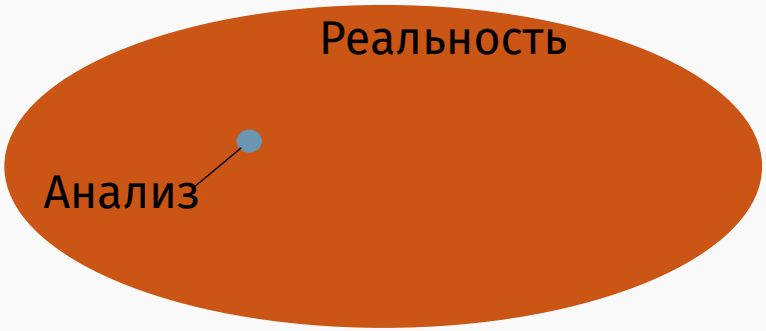
Доля обнаруженных результатов среди всех истинных

Полнота vs точность



Ориентация на точность vs ориентация на полноту

Полнота vs точность



Но может быть и вот так

Soundness & completeness vs precision & recall

В чём разница?

- Характеристики верификации — это *качественные* свойства
- Характеристики CA — это *количественные* свойства

Можно (неформально) сказать, что

- soundness = 100% precision
- completeness = 100% recall
- верификация — частный случай CA *(но это тссс)*

  Ученые в области верификации нас бы за это побили

Каким бывает статический анализ

- Сигнатурный поиск
- Анализ на основе типов
- Абстрактная интерпретация
- Символьное исполнение
- Анализ потока данных
- тысячи их!

Сигнатурный поиск

- Отличный метод анализа, просто превосходный
- Приносит пользу человечеству
- Самый распространённый вид анализа

  (на самом деле нет)

Сигнатурный поиск

- Отличный метод анализа, просто превосходный
- Приносит пользу человечеству
- Самый распространённый вид анализа

  (на самом деле нет)

- Им мы заниматься **не будем**
- Почему?

  Хороший сигнатурный анализатор — это 10K+ *сигнатур*.

  Среднюю сигнатуру любой из вас напишет за пару дней.

Каким бывает статический анализ

- ~~Сигнатурный поиск~~
- Анализ на основе типов
- Абстрактная интерпретация
- Символьное исполнение
- Анализ потока данных
- Тысячи их!

Всё оставшееся — это разный взгляд на одни и те же проблемы

> *За эту фразу адепты какой-нибудь абстрактной интерпретации нас бы тоже побили*

Структура конкретно этого курса

- Лекции
- Практика
- Экзамен

Лекционная часть

Идея лекций этого курса — показать конструирование анализаторов от простых к сложным
(к очень сложным, если успеем)

Создана на основе:

- Лекций по программному анализу университета Аархуса
  `http://cs.au.dk/~amoeller/spa`
- Материалов других курсов и книжек
- Собственного опыта, шишек и граблей

Практическая часть

На основе системы TIP всё того же университета Аархуса

`https://github.com/vorpal-research/TIP`

- **Что это такое?**
  - Tiny Imperative Programming Language
  - Язык и инфраструктура для его анализа, специально для обучения
  - Всё написано на Scala
- **Что нужно будет делать?**
  - Есть примеры кода на этом языке
  - Многие анализы там уже реализованы
  - Некоторые предстоит реализовать вам

FAQ по практической части

- **Почему не промышленные системы анализа?**

  Потому что в них все проблемы уже решены (или нет) и все методы выбраны, а это же самое интересное!
- **Почему не всем известные языки?**

  Анализатор любого сложного языка = 80% заморочек этого языка + 20% анализа
- **А зачем вообще учиться на игрушечной системе???**

  Любой анализ строится по одним и тем же принципам, любая модель языка — это всё равно модель
- **На чём будем разрабатывать?**

  На Scala. Чуть не забыл, нужно до первого занятия выучить Scala.
- **Почему императивный, а не функциональный язык?**

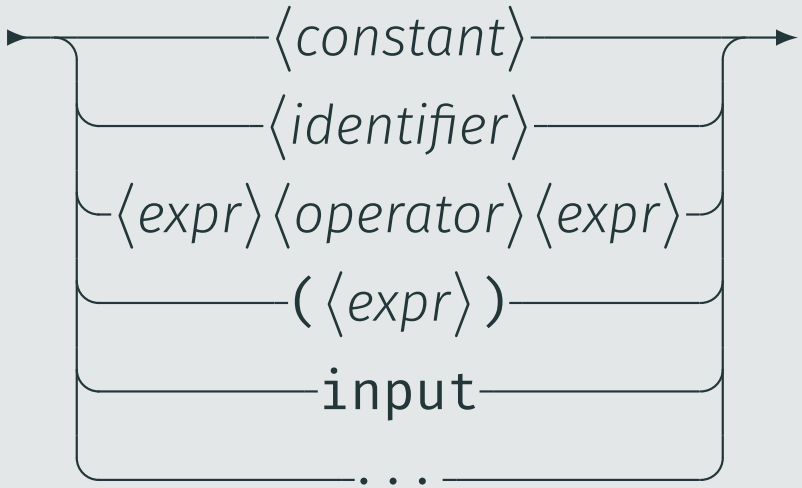
  Чисто функциональные языки **гораздо** проще анализировать

Язык TIP

- Примитивный процедурный язык
- C-подобный синтаксис
- Максимально упрощено всё, что анализу неинтересно
    - Нет структур и объектов
    - Нет глобальных переменных
- То, что интересно, оставлено
    - Есть указатели, нет арифметики над ними
    - Есть функции (и указатели на них)
    - Нет массивов (вот тут у ребят прокол)
- В дальнейшем будет расширяться (надеемся)

TIP: выражения (expressions)

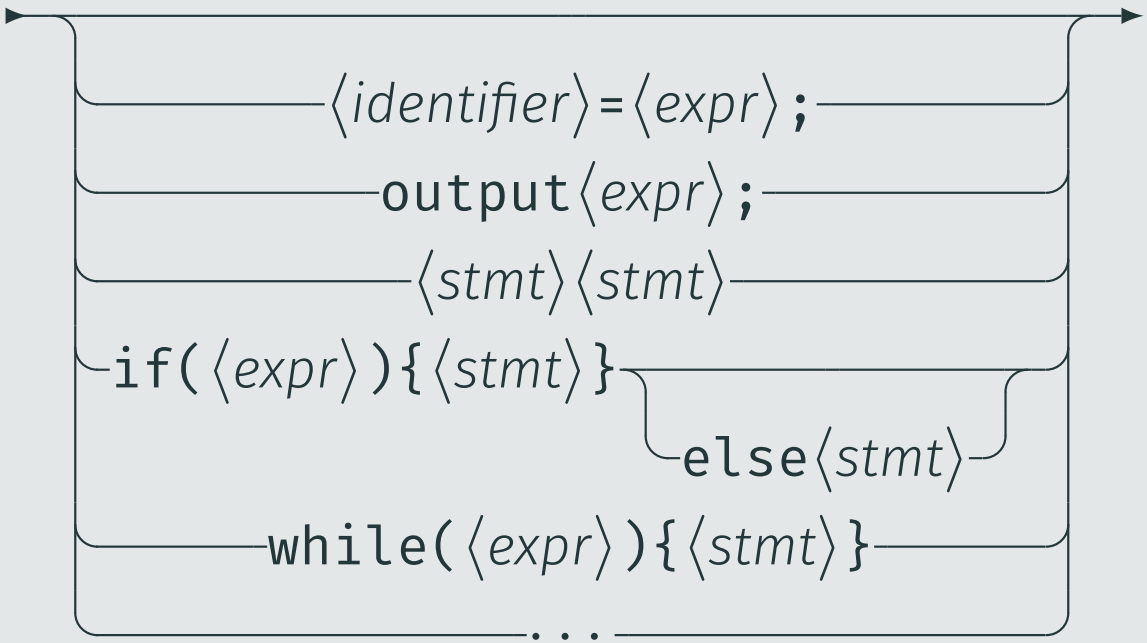
⟨*expr*⟩



- Только целые числа
  - Були кодируются как 0 и 1
- Доступные операторы: `+`, `-`, `*`, `/`, `>`, `==`
- Унарных и битовых операций нет
- `input` это некий абстрактный *ввод*

TIP: операторы (statements)

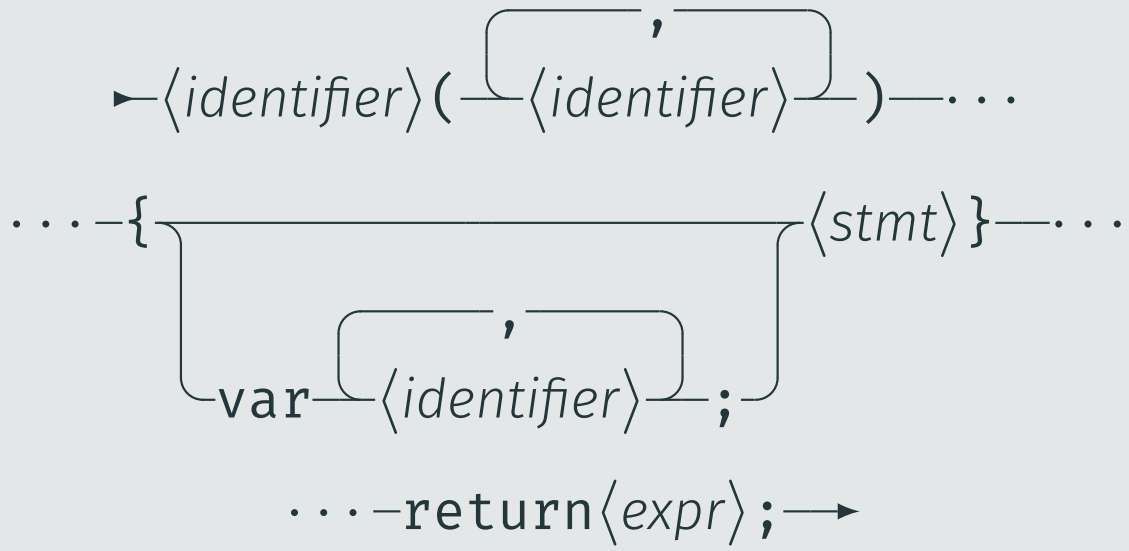
⟨*stmt*⟩



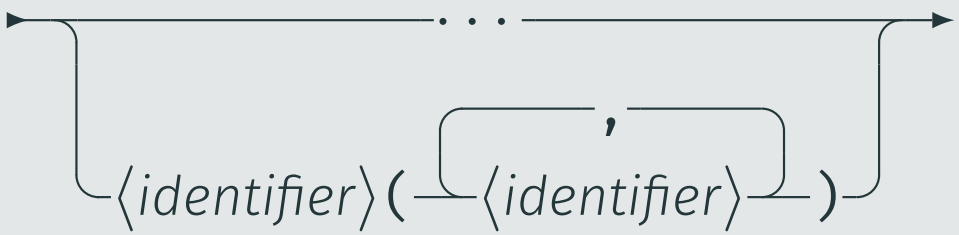
- Нельзя объявлять переменные где попало (как в C до 99 года)
- `output` это некий абстрактный *вывод*

TIP: функции

⟨*fun*⟩



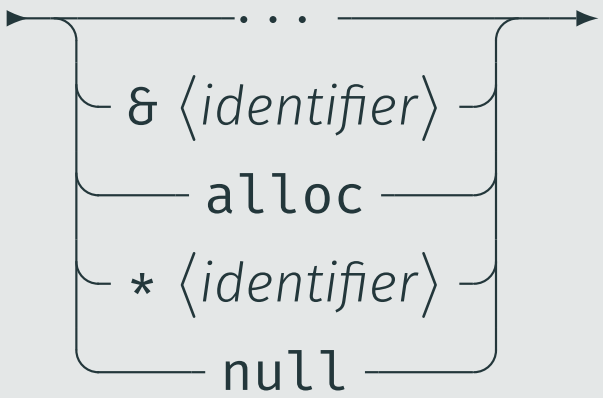
⟨*expr*⟩



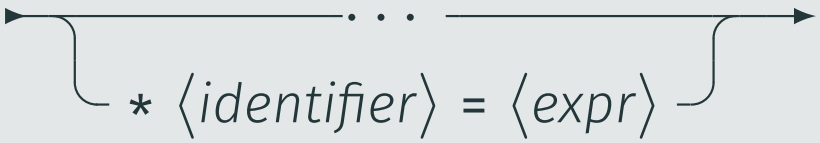
- Один `var`-блок, один `return`

TIP: указатели

⟨*expr*⟩



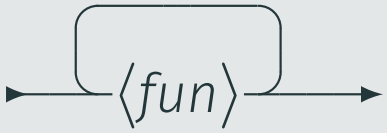
⟨*stmt*⟩



- Выделять объекты можно только по одному
- Нет арифметики указателей

TIP: программа

⟨*program*⟩



TIP: примеры

```
fib(i) {
  var j,k,h;
  j = 1; k = 1;
  while(i > 0) {
    i = i - 1;
    h = j + k;
    j = k;
    k = h;
  }
  return j;
}
```

TIP: примеры

```
main() {
  var a;
  a = input;
  output fib(a);
}
```

TIP: примеры

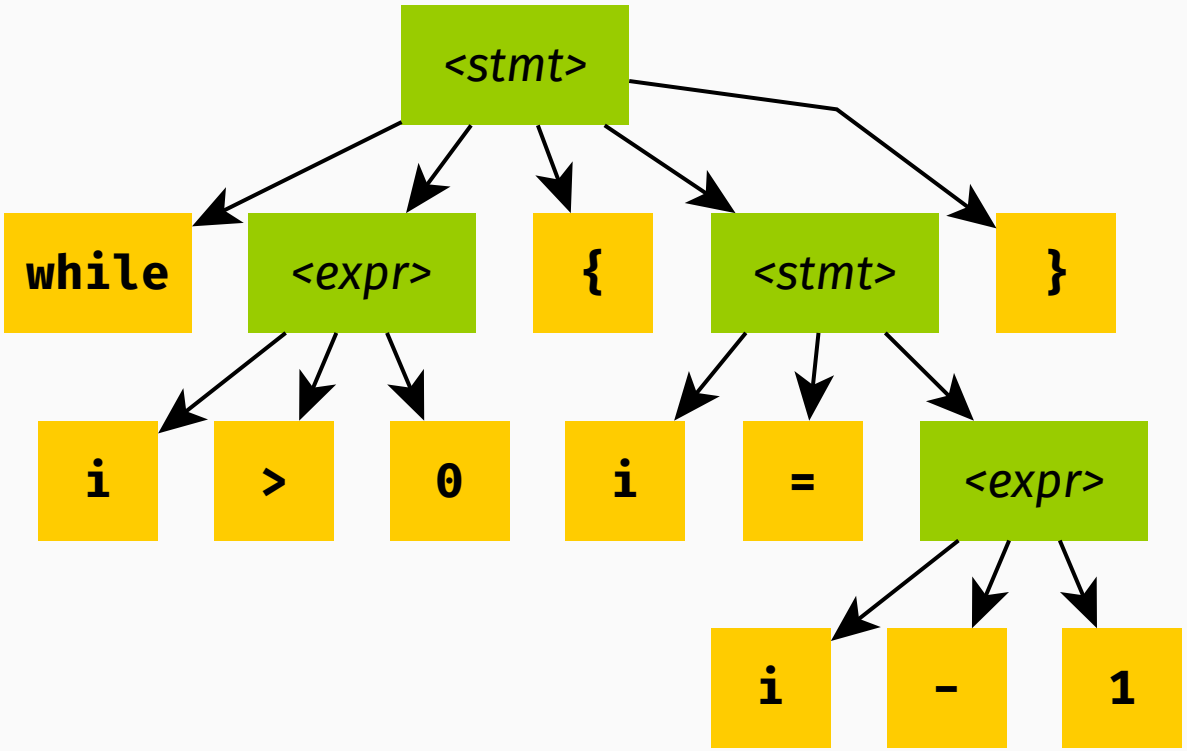
```
realloc(ptr) {
  if(ptr == null) {
  } else {
    *ptr = alloc;
    **ptr = 0;
  }
  return *ptr;
}
```

Модель языка

- Практически любой анализ реализуется на модели
- Исходный код — это тоже модель, но очень неудобная
- Самая простая модель — это *абстрактное синтаксическое дерево* (AST)

Абстрактное синтаксическое дерево

```
while(i > 0) {
  i = i - 1
}
```



Абстрактное синтаксическое дерево

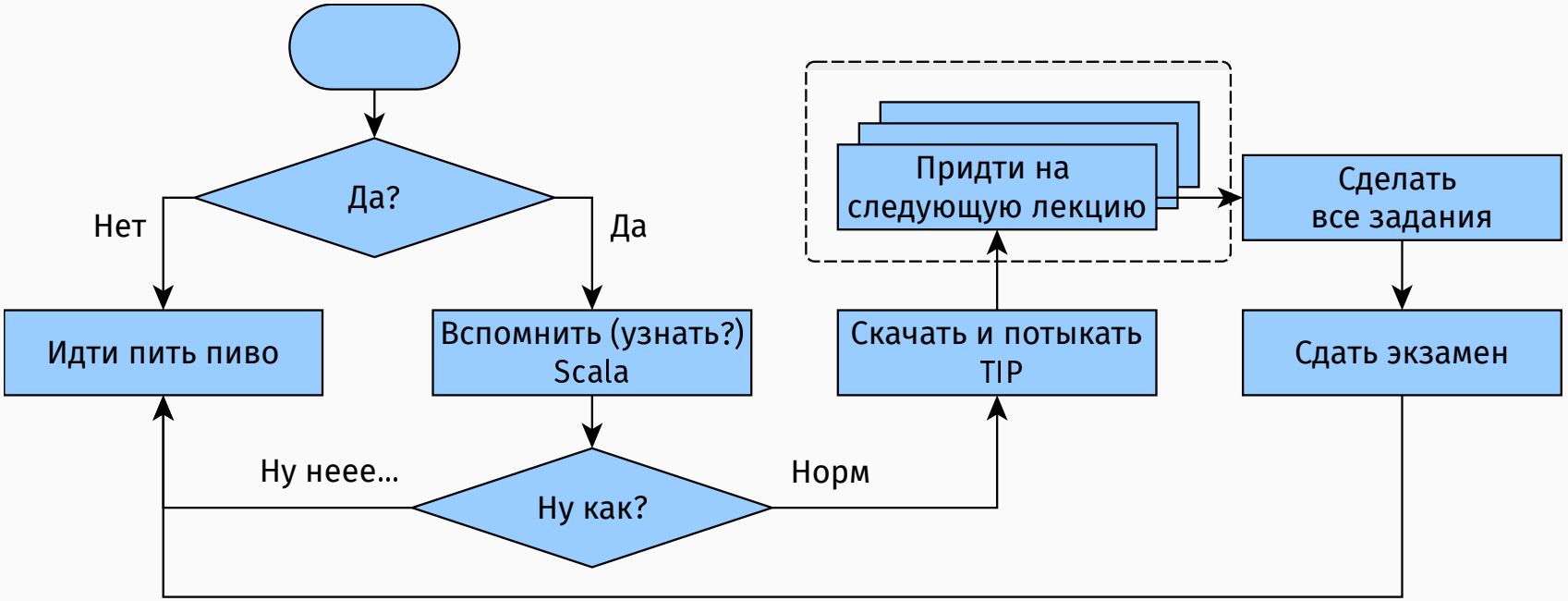
- Дерево разбора, получаемое в результате парсинга
- Обычно из него удаляется всякая ненужная ерунда
    - Пробелы
    - Комментарии
    - Артефакты грамматики
        - Нормальные формы
        - Форма рекурсии
        - Оптимизации парсинга
        - И так далее
- Получаем тот же исходный код, но в виде удобного дерева
- Анализ над деревом:
    - В ООП используются visitor’ы
    - В ФП используется pattern matching

Beyond AST

- Для простого анализа достаточно AST
- Разумеется, существуют более сложные модели
- То же AST можно дополнять всякими данными про семантику
    - Обычно такую модель называют ASG (*абстрактным семантическим графом*)
- Графы потока данных/управления
- Графы зависимостей
- Нормальные формы кода

Какие-то из этих моделей рассмотрим позже, какие-то не рассмотрим

Что дальше


Да?
Нет
Да
Идти пить пиво
Вспомнить (узнать?) Scala
Ну как?
Ну неее...
Норм
Скачать и потыкать TIP
Придти на следующую лекцию
Сделать все задания
Сдать экзамен

В следующей серии

Добавим в TIP систему типов и попытаемся поделать какой-никакой анализ

belyaev@kspt.icc.spbstu.ru

akhin@kspt.icc.spbstu.ru