Построение универсального представления графа потока управления для статического анализа исходного кода

ЧелГУ
Зубов М.В, Пустыгин А.Н., Старцев Е.В.

Статический анализ

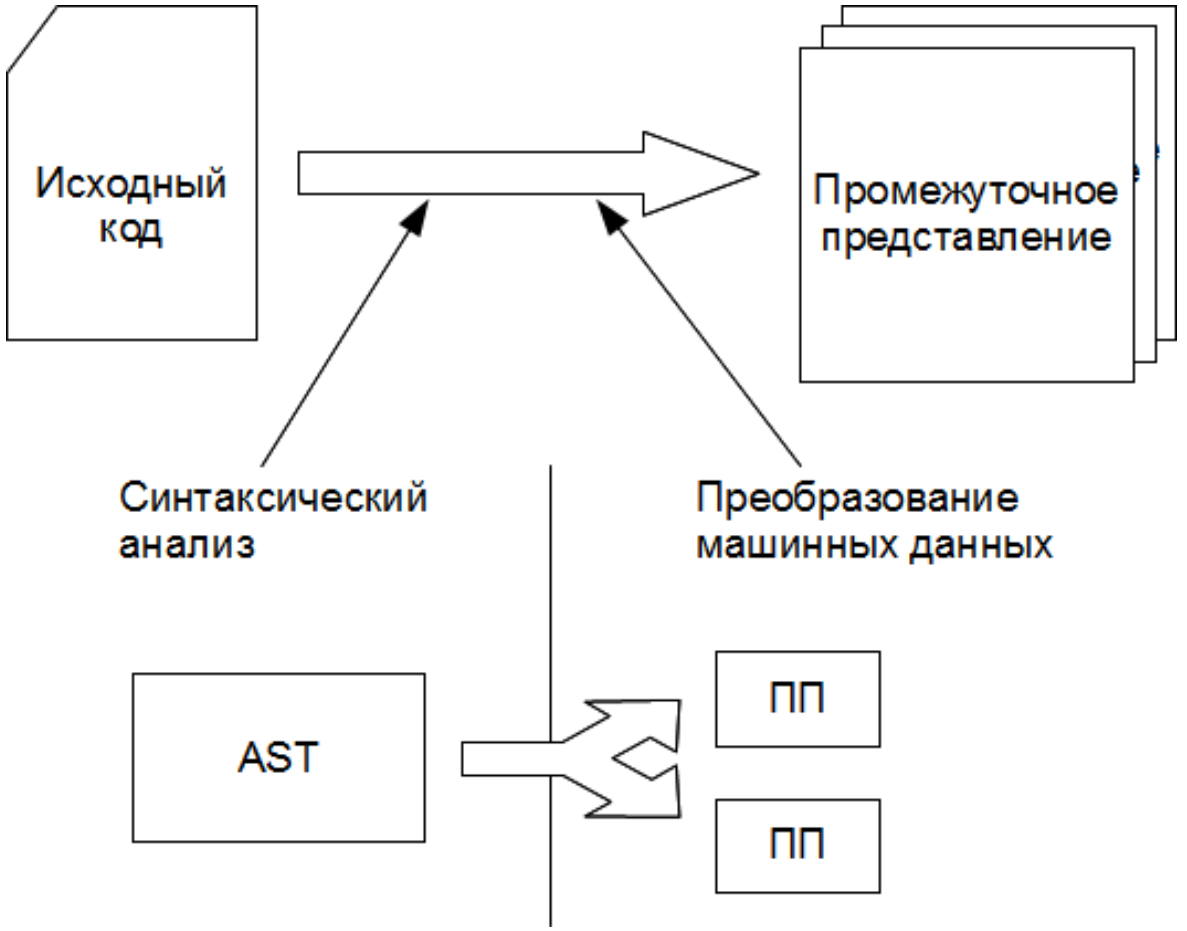
- способ анализа программного продукта по его тексту до запуска его исполняемого кода. Используется статический анализ исходного текста, поэтому программные инструменты инструменты пригодны для анализа ПО с открытым исходным текстом

Промежуточное представление исходного текста

- это текстовый набор данных, над которым выполняется анализ. ПП получается из исходного кода с целью сокращения затрат ресурсов при последующей обработке, в первую очередь трудозатрат на разработку инструментов

Наиболее технологичный путь получения ПП из AST, внутреннего набор данных компилятора (тогда AST — не промежуточное представление по определению)

Получение промежуточного представления из исходного текста



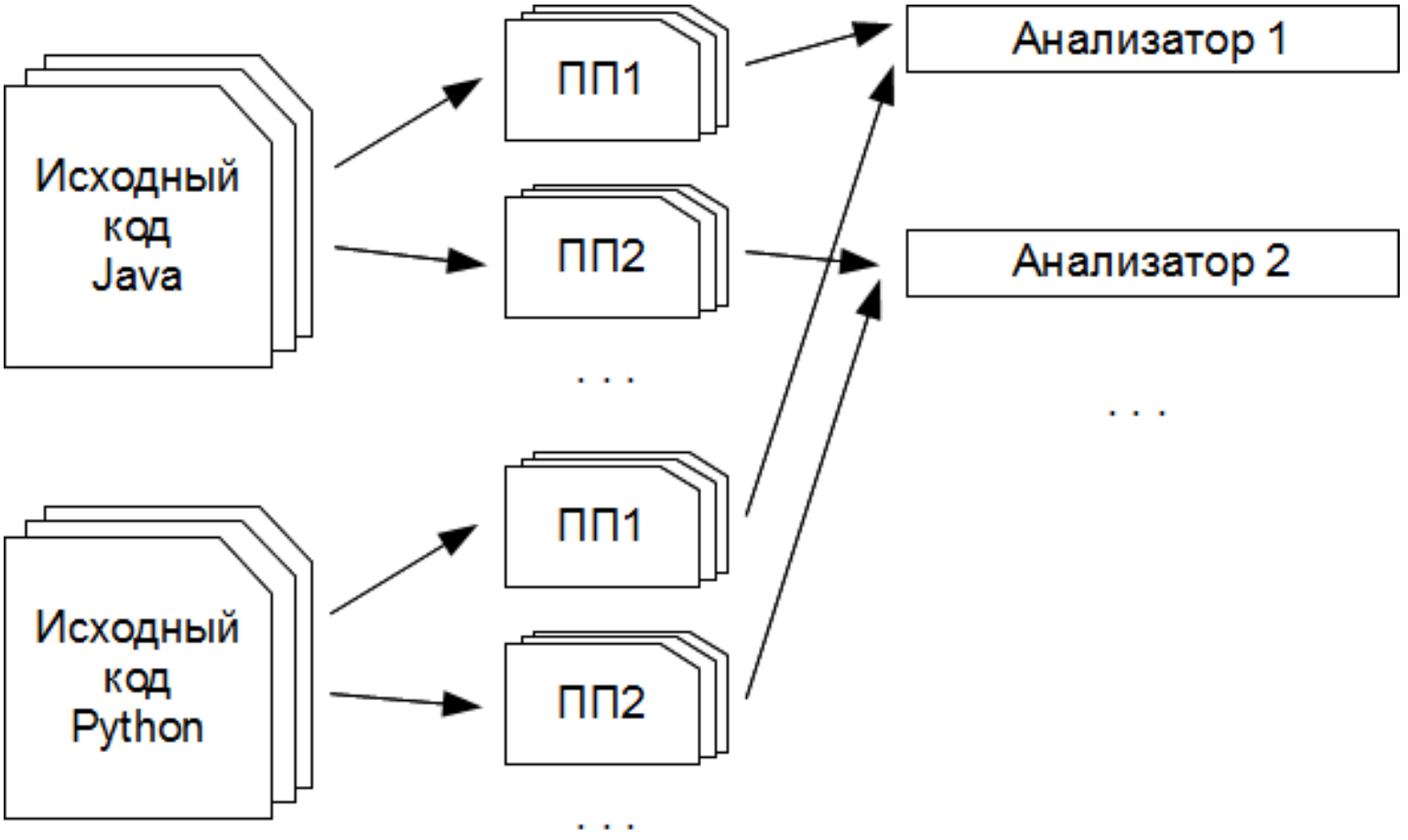
Универсальные многоуровневые представления в анализе исходного текста

- промежуточные представления, описывающие родственные синтаксические конструкции разных языков наиболее близким, то есть общим способом. Строятся из представлений предыдущего уровня путем сохранения только необходимые для конкретного анализа данных

Предназначены для упрощения конкретного анализа «прореживанием» предыдущего ПП

Универсальный (единый) формат позволяет использовать единый программный инструмент для исходных текстов на разных языках.

Универсальные многоуровневые представления



Универсальное классовое представление (UCR)

для Python и Java предназначено для построения и анализа диаграммы классов по исходному тексту проекта на одном из этих языков или проектов на обоих языках одновременно

Позволяет применять унифицированный инструмент анализа диаграммы классов для исходного текста Python и Java

Граф потока управления (CFG)

ориентированный граф, описывающий множество путей исполнения текста, предназначенный для определения трасс достижимости на этапе статического анализа и решения связанных задач (определение порядка вызовов, построения срезов по заданным атрибутам и пр.)

В данной работе применяется для анализа исходного текста

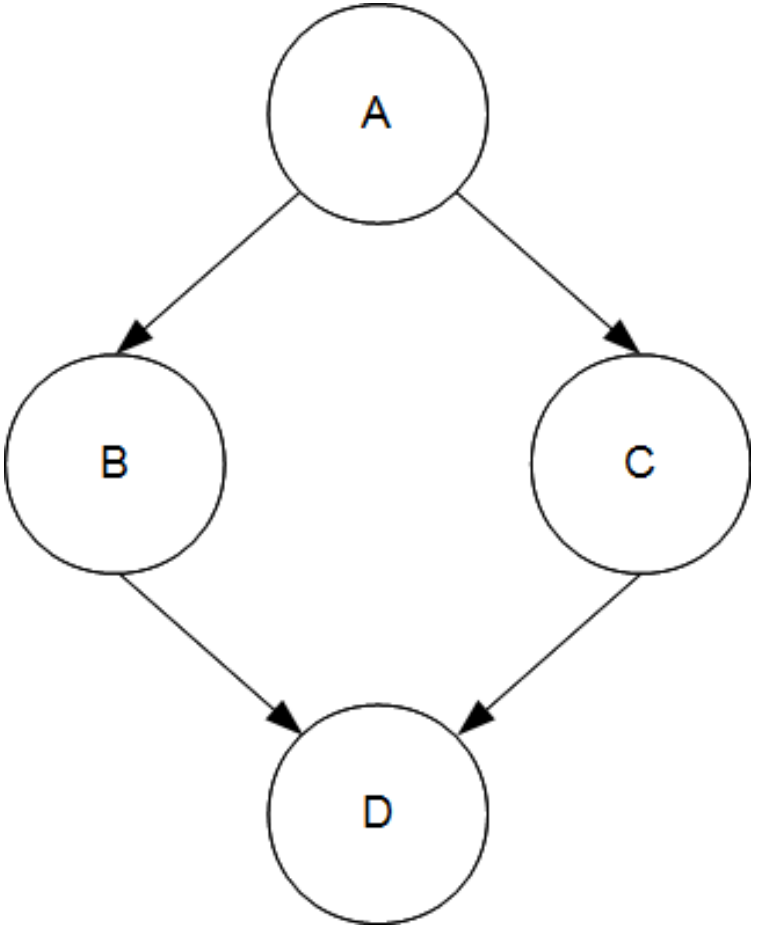
Ограничения принятой модели CFG

- Граф строится только для функции или метода, с возможностью связывания отдельных частей друг с другом
- Вызовы из метода пока не учитываются
- Участки последовательного кода без ветвления представляются одним узлом
- Представление - ориентированный граф, предназначенный для анализа известными методами
- Текстовая форма основана на открытом формате GraphML (graphml.graphdrawing.org), позволяющем описывать в XML графы любого вида

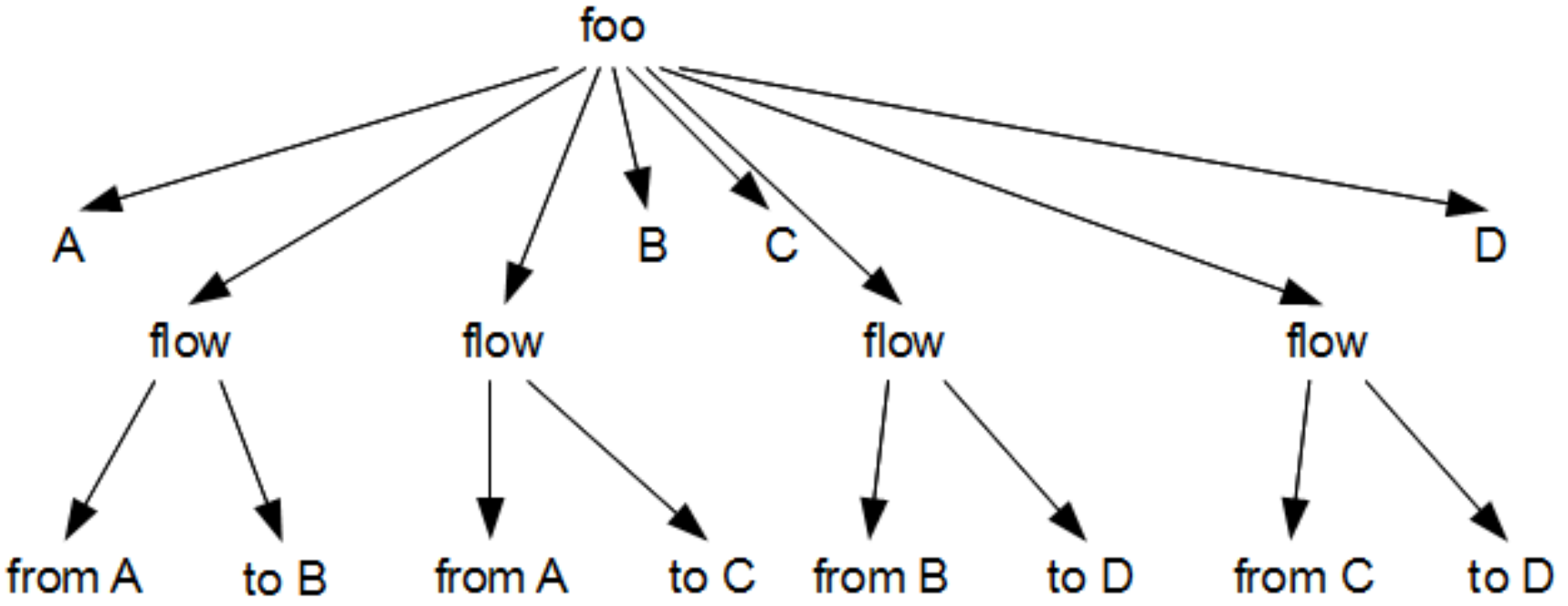
Пример кода Java для иллюстрации формата CFG

```
    public void foo() {
A:      this.count++;
        if (this.enough){
B:          bar();
        } else {
C:          fooBar();
        }
D:  }
```

Граф потока управления для примера кода



Представление примера в виде дерева



Представление примера в виде текста XML

```
– <Method name="foo">
    <Block id="A"/>
    <Flow from_id="A" to_id="A_if"/>
    <If id="A_if"/>
    <Flow from_id="A_if" to_id="B"/>
    <Block id="B"/>
    <Flow from_id="A_if" to_id="C"/>
    <Block id="C"/>
    <Flow from_id="B" to_id="D"/>
    <Flow from_id="C" to_id="D"/>
    <Block type="Exit" id="D"/>
  </Method>
```

Визуализация CFG

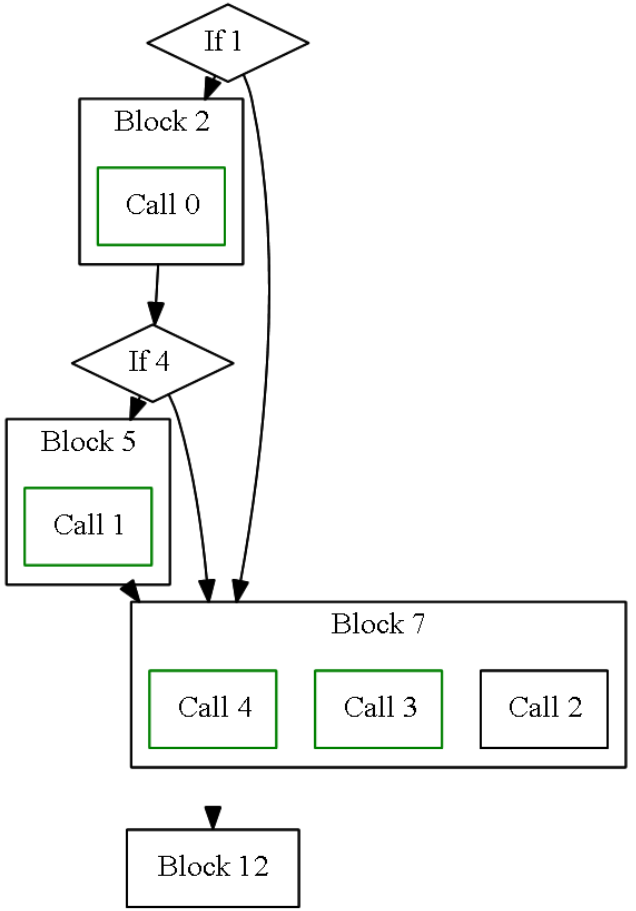
- Используется формат dot из пакета graphviz (www.graphviz.org)
- На каждый метод создается отдельное изображение

Пример исходного кода для визуализации (Python)

```
def set_color_formatter(logger=None, **kw):
    if logger is None:
        logger = logging.getLogger()
        if not logger.handlers:
            logging.basicConfig()
    format_msg = logger.handlers[0].formatter._fmt
    fmt = ColorFormatter(format_msg, **kw)
    fmt.colorfilters.append(xxx_cyan)
    logger.handlers[0].setFormatter(fmt)
```

Из класса logilab.common.logging_ext.ColorFormatter, Logilab-Common

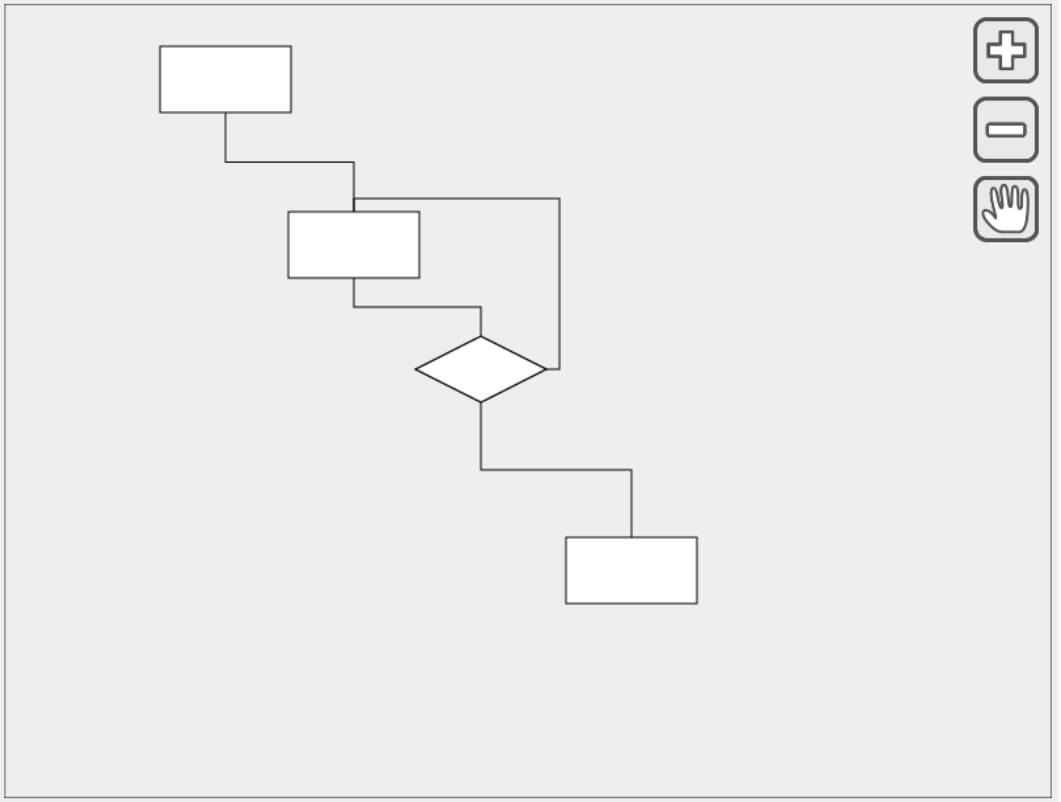
Визуализация CFG примера кода утилитой dot



Пути анализа CFG

- Интерактивный визуализатор CFG с взаимной навигацией по коду и классам
- построение срезов (слайсинг) для сокращения количества данных в представлении путем
- Построение диаграммы последовательности в стиле UML
- Поиск трасс между двумя указанными функциями

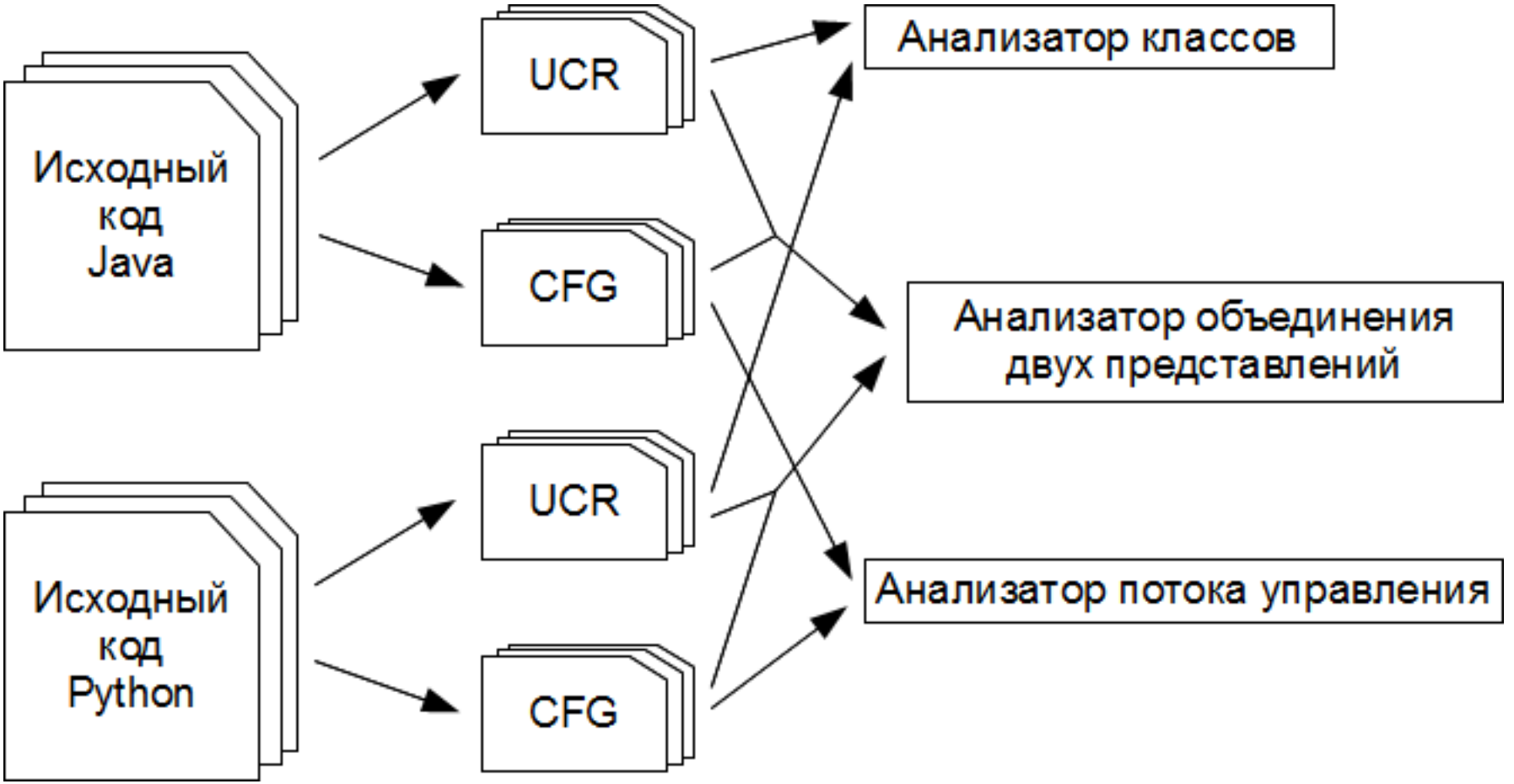
Прототип интерактивного визуализатора CFG



- Main
  - main()

```
        public static void main(String[] args) throws SQLException, ClassNotFoundException, IOException {
//              Runtime rt = Runtime.getRuntime();
//              Process p = rt.exec("/home/mz/export_my");
//              BufferedReader reader = new BufferedReader(new InputStreamReader(p.getInputStream()));
//              String line;
//              while ((line = reader.readLine()) != null)
//                      System.out.println(line);
                System.out.println(Locale.getDefault() + " " + Locale.ENGLISH);
                //Locale.setDefault(Locale.ENGLISH);
//              Map env = System.getenv();
//              for (String key : env.keySet())
//                      System.out.println(key + " - " + env.get(key));

                Class.forName ("oracle.jdbc.driver.OracleDriver");

        Connection conn = DriverManager.getConnection
            ("jdbc:oracle:thin:@//localhost:1521/XE", "mzubov", "123");
                        // @//machineName:port/SID,   userid,  password
        try {
        Statement stmt = conn.createStatement();
```

Совместное использование промежуточных представлений. Анализ объединения представлений



Механизм совместного анализа CFG и UCR

В CFG указывается id класса из UCR, к которому принадлежит метод

- метод идентифицируется по сигнатуре

  Функции python, не принадлежащие классам, в анализе не участвуют

- Общие id проставляются в текстовом XML-файлах CFG и UCR

Представление связей в схемах XML

https://github.com/exbluesbreaker/csu-code-analysis/blob/master/data/cfg.xsd

```
30  <xs:complexType name="Class">
31          <xs:complexContent>
32                  <xs:extension base="BaseElement">
33                    <xs:sequence>
34                      <xs:element name="Modifier" type="ModifierType" minOccurs="0" maxOccurs="unbounded"/>
35                      <xs:element name="Attr" type="Attribute" minOccurs="0" maxOccurs="unbounded"/>
36                      <xs:element name="Method" type="Method" minOccurs="0" maxOccurs="unbounded"/>
37                      <xs:element name="Parent" type="ParentClass" minOccurs="0" maxOccurs="unbounded"/>
38                      <xs:element name="Template" type="Template" minOccurs="0" maxOccurs="unbounded"/>
39                    </xs:sequence>
40                    <xs:attribute name="id" type="xs:integer"/>
41                    <xs:attribute name="filename" type="xs:string"/>
42            </xs:extension>
43    </xs:complexContent>
44  </xs:complexType>
```

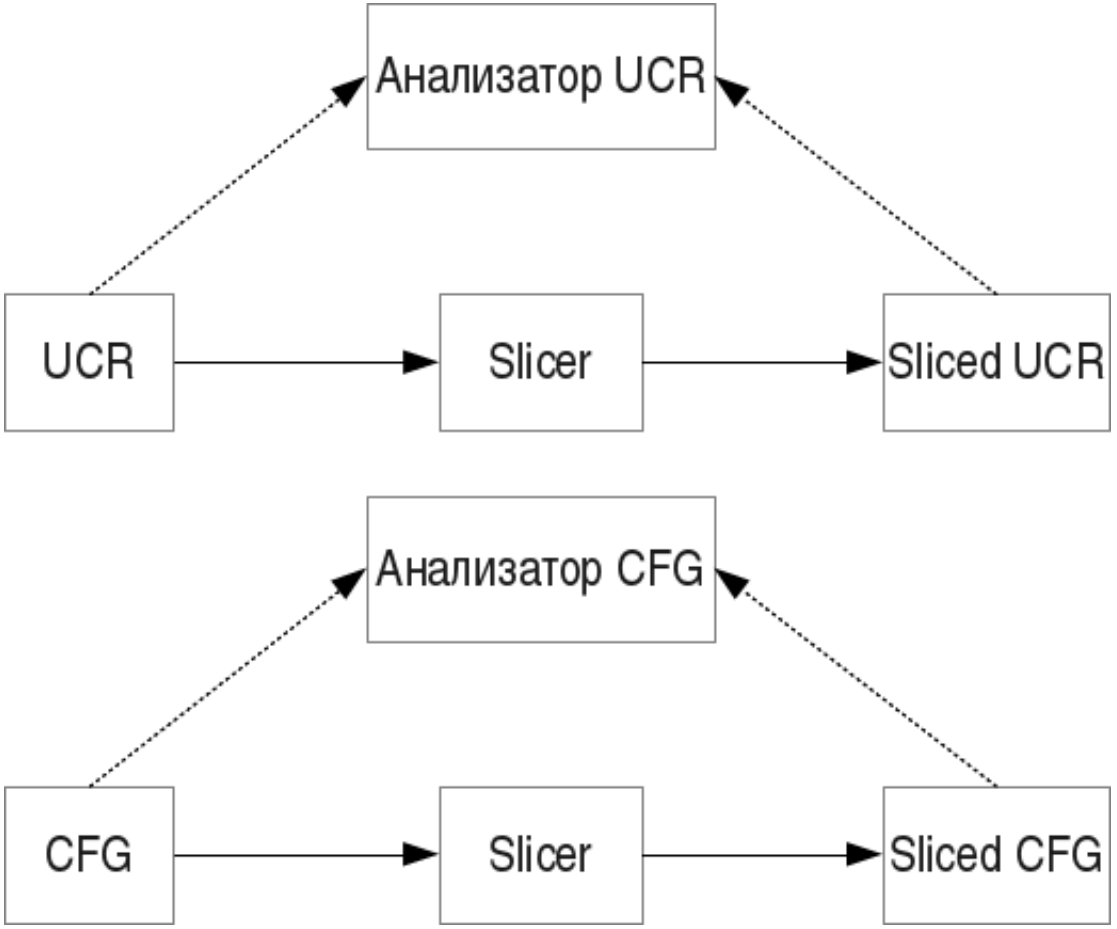
https://github.com/exbluesbreaker/csu-code-analysis/blob/master/data/classes.xsd

```
14  <xs:complexType name="Method">
15    <xs:choice minOccurs="0" maxOccurs="unbounded">
16      <xs:element name="TryExcept" type="TryExcept"/>
17      <xs:element name="TryFinally" type="TryFinally"/>
18      <xs:element name="With" type="With"/>
19      <xs:element name="Block" type="Block"/>
20      <xs:element name="Flow" type="Flow"/>
21      <xs:element name="For" type="For"/>
22      <xs:element name="If" type="If"/>
23      <xs:element name="While" type="While"/>
24    </xs:choice>
25    <xs:attribute name="name" type="xs:string"/>
26    <xs:attribute name="parent_class" type="xs:string"/>
27    <xs:attribute name="label" type="xs:string"/>
28    <xs:attribute name="cfg_id" type="xs:integer"/>
29    <xs:attribute name="ucr_id" type="xs:integer" use="optional"/>
30  </xs:complexType>
```

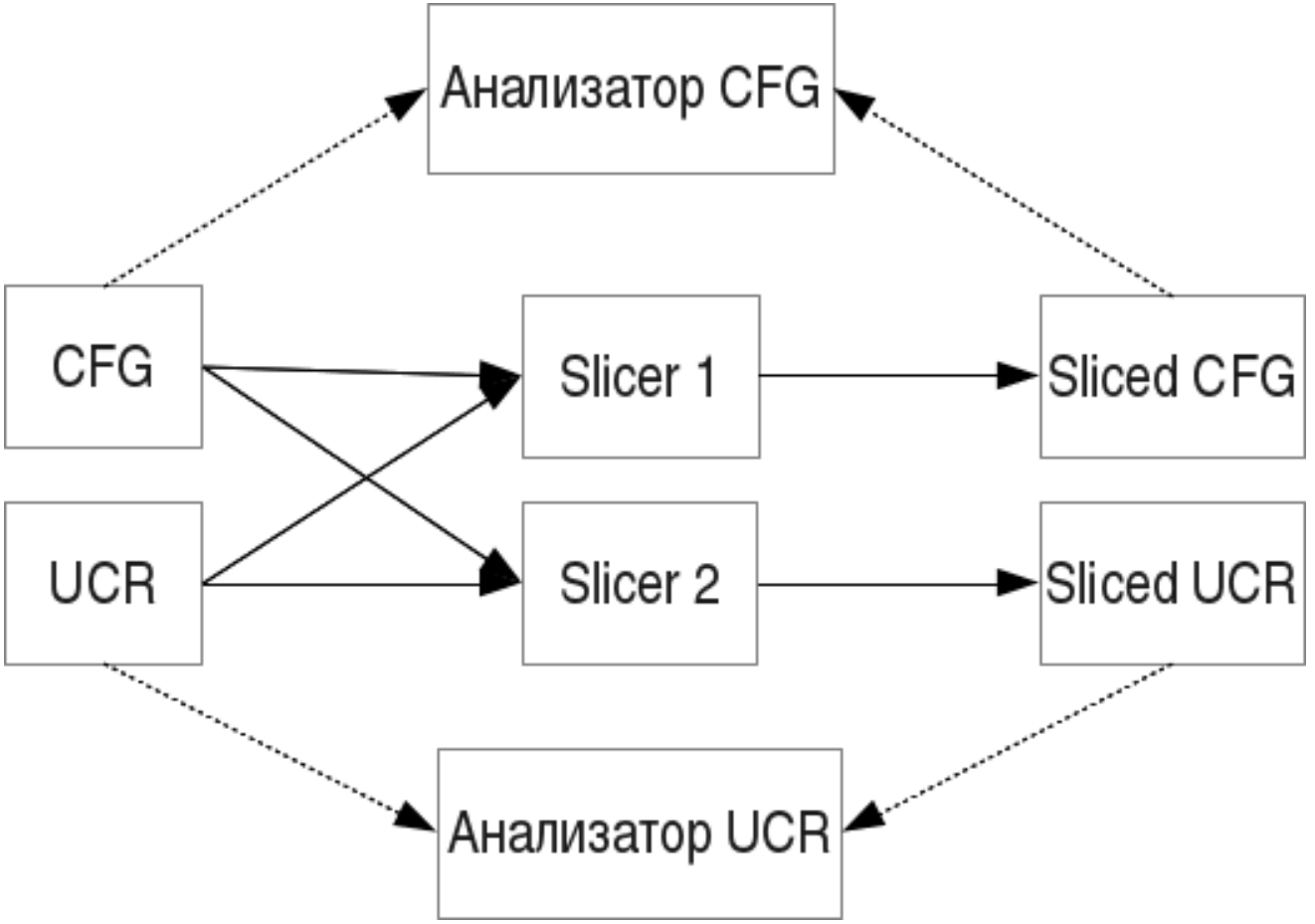
Совместное использование промежуточных представлений для слайсинга

- Метод уменьшения количества данных в представлении при решении частной задачи анализа
- Результат слайсинга остается валидным представлением (не изменяется формат данных)

Задача построения срезов для UCR и CFG по отдельности



Слайсинг на основе нескольких представлений



Слайсинг на основе нескольких представлений

- Слайсинг одного из промежуточных представлений исходного кода с использованием других промежуточных представлений этого же кода
- Например, при слайсинге CFG с использованием UCR может быть использована информация о принадлежности методов из CFG классам из UCR

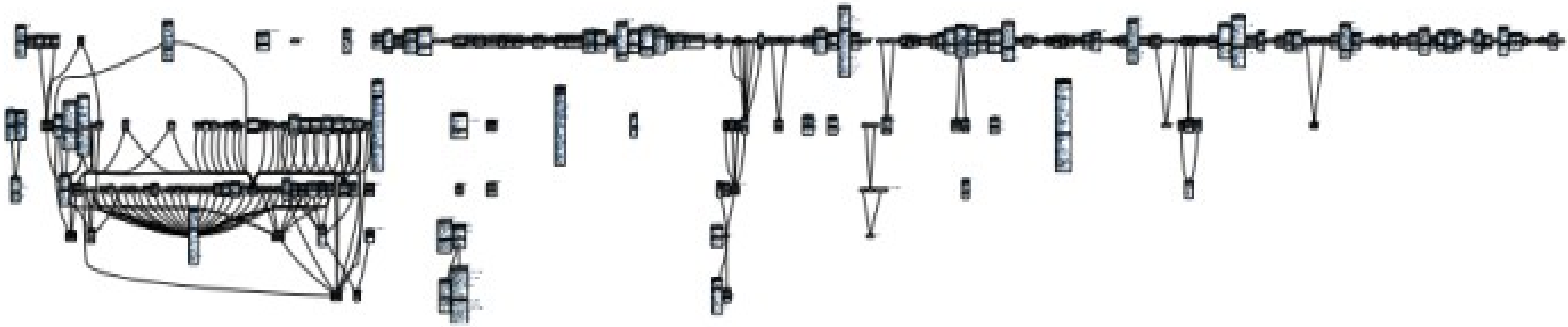
Построенные для имеющихся промежуточных представлений срезы

- Слайсинг UCR:
  - Дерево наследования для заданного класса
- Слайсинги CFG:
  - Вызываемые методы для заданного метода
  - Вызывающие методы для заданного метода
- Слайсинги UCR с использованием CFG
  - Классы, экземпляры которых создаются в методах заданного класса
- Слайсинги CFG с использованием UCR
  - Методы или функции, создающие экземпляр заданного класса

Пример слайсинга UCR с использованием CFG (исходный текст python, библиотека logilab.astng)

- Выполняется слайсинг для класса FromMixinImport из модуля logilab.astng.mixins.
- В результате остается интересующий класс и классы, экземпляры которых создаются в его методах.
- 2 варианта представления результатов:
  - Указываются связанные классы-родители из диаграммы классов,
  - Указываются только интересующие классы (более строгий слайсинг)

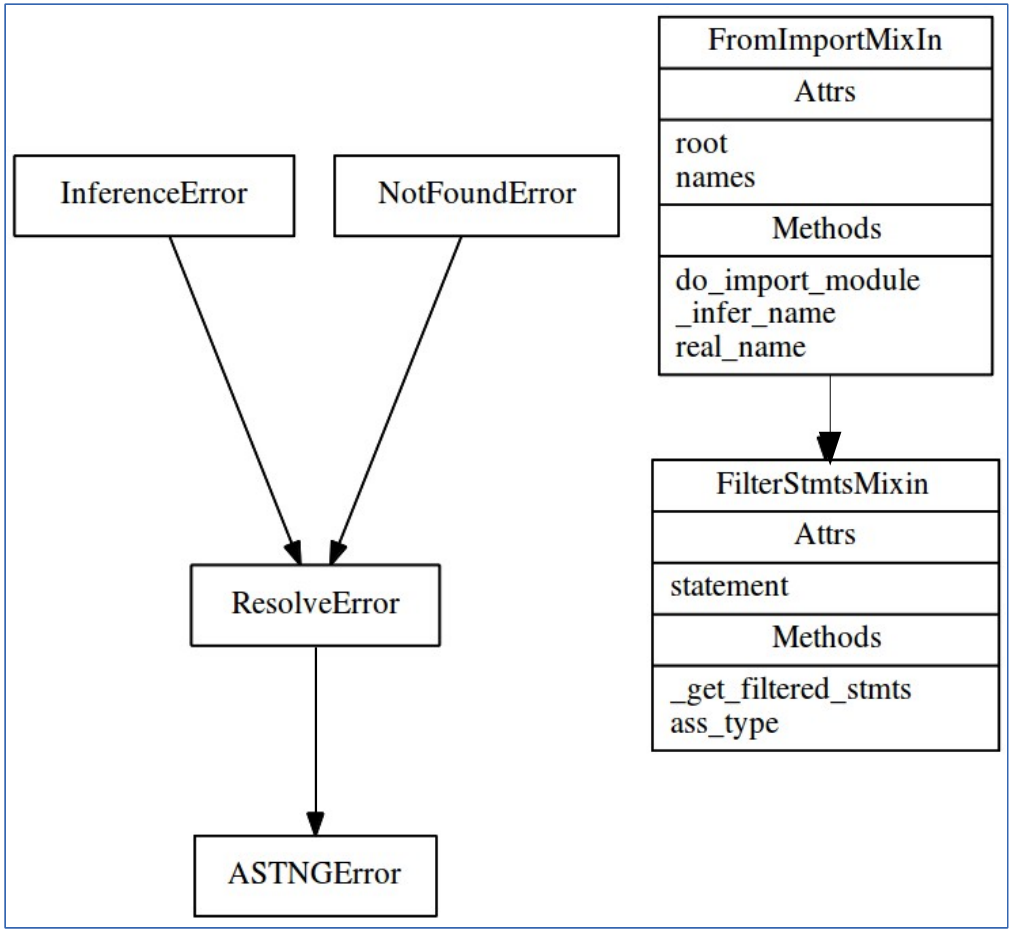
UCR logilab до выполнения слайсинга



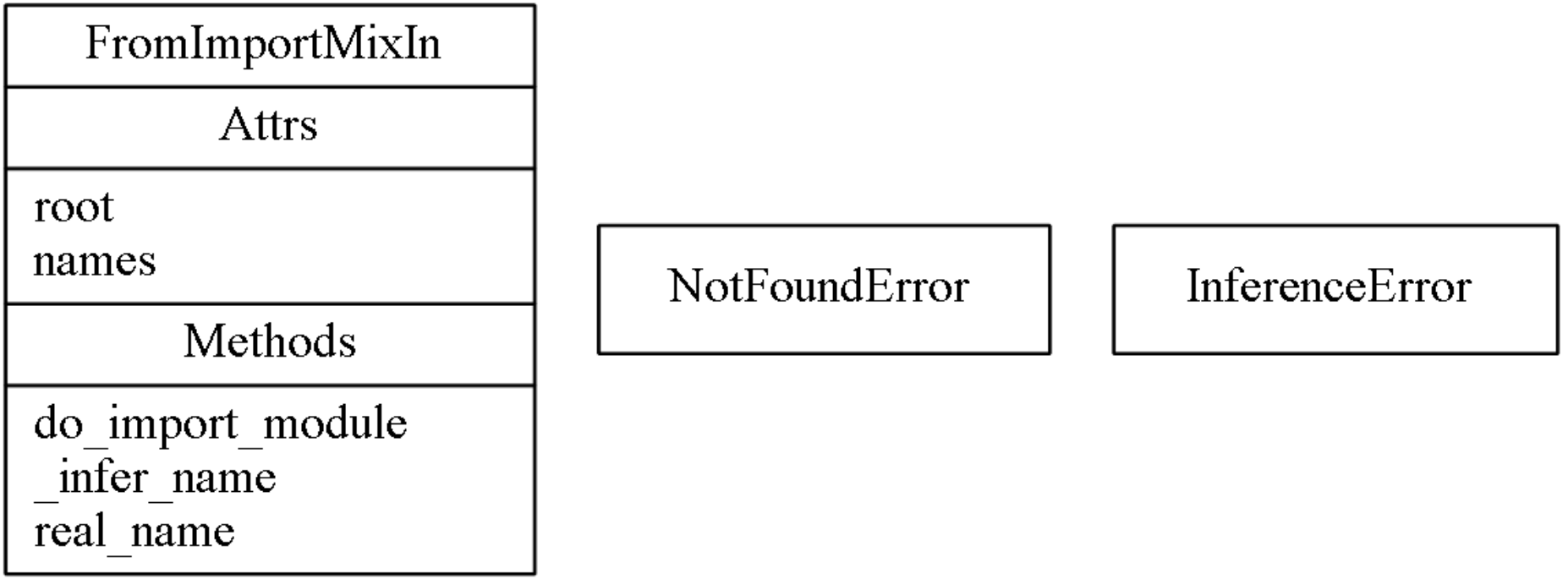
Исходный код класса для примера

```
class FromImportMixIn(FilterStmtsMixin):
    """MixIn for From and Import Nodes"""

    def _infer_name(self, frame, name):
        return name

    def do_import_module(self, modname):
        """return the ast for a module whose name is <modname> imported by <self>
        """
        # handle special case where we are on a package node importing a module
        # using the same name as the package, which may end in an infinite loop
        # on relative imports
        # XXX: no more needed ?
        mymodule = self.root()
        level = getattr(self, 'level', None) # Import as no level
        # XXX we should investigate deeper if we really want to check
        # importing itself: modname and mymodule.name be relative or absolute
        if mymodule.relative_to_absolute_name(modname, level) == mymodule.name:
            # FIXME: we used to raise InferenceError here, but why ?
            return mymodule
        try:
            return mymodule.import_module(modname, level=level)
        except ASTNGBuildingException:
            raise InferenceError(modname)
        except SyntaxError, ex:
            raise InferenceError(str(ex))

    def real_name(self, asname):
        """get name from 'as' name"""
        for name, _asname in self.names:
            if name == '*':
                return asname
            if not _asname:
                name = name.split('.', 1)[0]
                _asname = name
            if asname == _asname:
                return name
        raise NotFoundError(asname)
```

Результат слайсинга (с наследованием)



Результат слайсинга (без наследования)



Спасибо за внимание!

https://github.com/exbluesbreaker/csu-code-analysis