INNOPOLIS UNIVERSITY

Применение технологии XSLT для генерации Control Flow Graph из AST для языка Simple

Павлов Андрей B23-SD-01

and.pavlov@innopolis.university

Оглавление

1. Задача

- Применение технологии XSLT для автоматического преобразования XML-представления AST программ на языке Simple в интерактивный граф потока управления (CFG).
- Оценка применимости XSLT для решения данной задачи: удобство обхода дерева, гибкость форматирования и возможность визуализации.

2. Методология

Преобразование выполняется в два этапа:

1. Генерация DOT-кода – шаблон generate-dot создаёт описание графа в формате Graphviz.
2. Рендеринг – полученный DOT передаётся библиотеке Viz.js, которая отображает SVG в браузере.

Обход AST основан на рекурсивном применении шаблонов. Для каждого типа оператора определён свой шаблон в режиме `form`, который создаёт узел с соответствующей формой, цветом и текстовой меткой.

- **Линейные участки** обрабатываются шаблоном `seqBody`, который последовательно соединяет узлы рёбрами.
- **Условные операторы** (`If`, `ElseIf`) порождают разветвления с рёбрами `true`/`false`.
- **Циклы** (`For`, `While`) строят петли: условие проверяется перед телом, а после тела поток возвращается к условию. Для `For` с итератором дополнительно генерируются узлы для получения перечислителя и вызова `MoveNext()`.
- **Прерывания** (`Break`, `Return`) направляют поток к выходу из текущего блока или функции.
- **Выражения** и **типы** преобразуются в строки с помощью шаблонов `text-form` и `type-form`, что позволяет компактно и читаемо отображать сложные конструкции (бинарные операции, литералы массивов/кортежей, вызовы функций).

3. Технология

- **XSLT 1.0** с расширением **EXSLT** (`exslt:node-set`) для работы с временными фрагментами дерева.
- Преобразование выполняется **в браузере Firefox** (поддерживает XSLT-процессор).
- Выходной HTML-документ содержит **встроенный CSS и JavaScript**, который использует библиотеку **Viz.js** для визуализации DOT-графа в формате SVG.

4. Результат

Разработан **единый XML-документ**, содержащий как само AST (в элементе `<Program>`), так и все XSLT-шаблоны для его преобразования. При открытии этого файла в браузере автоматически генерируется HTML-страница с интерактивным CFG.

Тестирование, проводимое на наборе различных примеров, показало, что:

- **Все основные конструкции** языка Simple корректно отображаются.
- **Рёбра** правильно помечаются (`true/false`) для условий.
- **Функции** выделяются в отдельные подграфы с собственной точкой входа и выхода.
- **Сложные выражения** (бинарные операции, обращения к элементам) преобразуются в читаемые метки.

Выводы: XSLT показала удобство преобразования AST в CFG. Однако из-за ограничений технологии (отсутствие изменяемых переменных, словарей и возможности отладки в реальном времени) реализация преобразования сложных конструкций становится трудоёмким процессом, требующим многократных проверок и исправления ошибок.

5. Реализация

В этом разделе описываются ключевые XSLT-шаблоны, их назначение и взаимодействие. Все шаблоны входят в единый XML-документ и используют пространство имён `xsl:stylesheet`. Основная логика разделена на несколько групп:

1. **Точка входа** – шаблон, соответствующий корню документа (`match="/"`).
2. **Генерация DOT** – шаблон `generate-dot`, строящий описание графа в формате Graphviz.
3. **Вспомогательные шаблоны** – для формирования рёбер, имён узлов, визуализации узлов-выходов из функций, последовательных обходов, преобразования выражений и типов в читаемые строки.
4. **Шаблоны узлов** – для каждого типа оператора создаётся узел с определённой формой, цветом и меткой.

5.1. Точка входа и генерация DOT-кода

Шаблон `match="/"` генерирует полный HTML-документ, в который встраивается результат работы `generate-dot`. Этот результат помещается в тег `<script id="dot-source">` как текст DOT. Далее JavaScript-библиотека Viz.js выполняет рендеринг.

Шаблон `generate-dot` создаёт заголовок `digraph CFG {`, определяет глобальные узлы `entry` и `exit`, после чего обрабатывает:

- Глобальные переменные (`VariableDeclaration`) – вызывается шаблон `seqRootVar`, который последовательно создаёт узлы для каждой переменной и соединяет их с `entry`.
- Функции (кроме `main`) – применяется `apply-templates` с параметром `prev`, который передаёт имя предыдущего узла для построения рёбер. Для каждой функции создаётся отдельный подграф с собственным входом и выходом (суффикс `_exit`).
- Функцию `main` – обрабатывается отдельно в режиме `main`, где начало функции соединяется с последним глобальным узлом, а выход – с глобальным `exit`.

```
        <xsl:template name="generate-dot">
            <xsl:text>digraph CFG {&#13;&#10;</xsl:text>
            <xsl:text>&#x09;entry [label="Program entry", shape=ellipse, fillcolor=darkgreen,
style=filled];&#13;&#10;</xsl:text>
            <xsl:text>&#x09;exit [label="Program exit", shape=ellipse, fillcolor=darkgreen,
style=filled];&#13;&#10;</xsl:text>

            <xsl:variable name="nodes" select="root/Program/*[self::VariableDeclaration]"/>

            <xsl:call-template name="seqRootVar">
                <xsl:with-param name="nodes" select="$nodes"/>
                <xsl:with-param name="i" select="1"/>
                <xsl:with-param name="prev" select="'entry'"/>
            </xsl:call-template>

            <xsl:apply-templates
```

```
            select="root/Program/*[
                not(self::VariableDeclaration)
                and not(self::FunctionDeclaration[Id/@name='main'])
            ]">
            <xsl:with-param name="prev" select="''"/>
        </xsl:apply-templates>

        <xsl:variable name="prev">
            <xsl:choose>
                <xsl:when test="$nodes[last()]/Id/@name">
                    <xsl:value-of select="$nodes[last()]/Id/@name"/>
                </xsl:when>
                <xsl:otherwise>
                    <xsl:text>entry</xsl:text>
                </xsl:otherwise>
            </xsl:choose>
        </xsl:variable>

        <xsl:choose>
            <xsl:when test="root/Program/FunctionDeclaration[Id/@name='main']">
                <xsl:apply-templates
                    select="root/Program/FunctionDeclaration[
                        (self::FunctionDeclaration[Id/@name='main'])
                    ]" mode="main">
                    <xsl:with-param name="prev" select="$prev"/>
                </xsl:apply-templates>
            </xsl:when>

            <xsl:otherwise>
                <xsl:value-of select="concat('&#x09;', $prev, ' -> exit;&#13;&#10;')"/>
            </xsl:otherwise>
        </xsl:choose>

        <xsl:text>}</xsl:text>
    </xsl:template>
```

5.2. Вспомогательные шаблоны

5.2.1. Генерация уникальных имён узлов

Шаблон `name` создаёт уникальное имя для каждого узла на основе имени функции (`fun`) и локального имени элемента с добавлением `generate-id()`. Это гарантирует отсутствие коллизий в графе.

```
    <xsl:template name="name">
        <xsl:param name="node"/>
        <xsl:param name="fun"/>
        <xsl:value-of select="concat($fun, '_', local-name($node), '_', generate-id($node))"/>
    </xsl:template>
```

5.2.2. Создание ребра

Шаблон `emit-edge` принимает имена узлов `from` и `to`, а также опциональную метку. Он формирует строку вида `from -> to [label="метка"];`.

```
    <xsl:template name="emit-edge">
        <xsl:param name="from"/>
        <xsl:param name="to"/>
        <xsl:param name="label" select="''"/>
```

```
        <xsl:text>&#x09;</xsl:text>
        <xsl:value-of select="$from"/>
        <xsl:text> -> </xsl:text>
        <xsl:value-of select="$to"/>

        <xsl:if test="$label != ''">
            <xsl:text> [label=&quot;</xsl:text>
            <xsl:value-of select="$label"/>
            <xsl:text>&quot;]</xsl:text>
        </xsl:if>

        <xsl:text>;&#13;&#10;</xsl:text>
    </xsl:template>
```

5.2.3. Визуализация узла-выхода

Изначально у нас все узлы-выходы из функций скрыты. Если вдруг, каким-либо образом, у нас происходит выход из функции, мы его показываем на графе.

```
    <xsl:template name="emit-exit-filled">
      <xsl:param name="node"/>

      <xsl:if test="contains($node, '_exit')">
          <xsl:text>&#x09;</xsl:text>
          <xsl:value-of select="$node"/>
          <xsl:text> [style=filled];&#13;&#10;</xsl:text>
      </xsl:if>
    </xsl:template>
```

5.2.4. Шаблоны режима `form`

Для каждого типа оператора определён шаблон с `mode="form"`. Эти шаблоны генерируют строку вида:

```
имя_узла [label="текст", shape=..., style=..., fillcolor=...];
```

```
    <xsl:template match="PrintStatement" mode="form">
      <xsl:text>[label="print </xsl:text>
      <xsl:for-each select="*[not(self::Span)]">
          <xsl:if test="position() > 1"><xsl:text>, </xsl:text></xsl:if>
          <xsl:apply-templates select="." mode="text-form"/>
      </xsl:for-each>
      <xsl:text>" shape=parallelogram fillcolor=green style=filled];&#13;&#10;</xsl:text>
    </xsl:template>


    <xsl:template match="ReturnStatement" mode="form">
        <xsl:param name="node"/>

        <xsl:text>[</xsl:text>
        <xsl:text>label="Return</xsl:text>
        <xsl:if test="*">
            <xsl:text>: </xsl:text>
            <xsl:apply-templates select="*" mode="text-form"/>
        </xsl:if>
        <xsl:text>" shape=box style="filled,rounded" fillcolor=lightcoral];&#13;&#10;</xsl:text>
    </xsl:template>

    <xsl:template match="IfStatement|ElseIfStatement" mode="form">
```

```
            <xsl:text>[label="</xsl:text>
            <xsl:choose>
                <xsl:when test="Binary">
                    <xsl:apply-templates select="Binary" mode="text-form"/>
                </xsl:when>
                <xsl:otherwise>
                    <xsl:apply-templates select="*[not(self::Body) and not(self::ElseIfStatement) and
not(self::ElseBody)][1]" mode="text-form"/>
                </xsl:otherwise>
            </xsl:choose>
            <xsl:text>" shape=diamond style=filled fillcolor=yellow];&#13;&#10;</xsl:text>
        </xsl:template>


        <xsl:template match="ForStatement" mode="form">
            <xsl:variable name="iter_label" select="concat(Id/@name, '_iterator')"/>
            <xsl:choose>

                <xsl:when test="not(RightExpression) and not(LeftExpression)">
                    <xsl:text>[label=&quot;true&quot; shape=diamond style=filled
fillcolor=yellow];&#13;&#10;</xsl:text>
                </xsl:when>

                <xsl:otherwise>
                    <xsl:text>[label="</xsl:text>
                    <xsl:choose>
                        <xsl:when test="RightExpression">
                            <xsl:value-of select="Id/@name"/>
                            <xsl:text> = </xsl:text>
                            <xsl:choose>
                                <xsl:when test="@reverse='true'">
                                    <xsl:apply-templates select="RightExpression/*" mode="text-form"/>
                                </xsl:when>
                                <xsl:otherwise>
                                    <xsl:apply-templates select="LeftExpression/*" mode="text-form"/>
                                </xsl:otherwise>
                            </xsl:choose>
                        </xsl:when>
                        <xsl:otherwise>
                            <xsl:value-of select="$iter_label"/>
                            <xsl:text> = </xsl:text>
                            <xsl:apply-templates select="LeftExpression/*" mode="text-form"/>
                            <xsl:text>.GetEnumerator()</xsl:text>
                        </xsl:otherwise>
                    </xsl:choose>
                    <xsl:text>" shape=box style=filled fillcolor=white];&#13;&#10;</xsl:text>
                </xsl:otherwise>
            </xsl:choose>
        </xsl:template>


        <xsl:template match="WhileStatement" mode="form">
            <xsl:text>[label="</xsl:text>
            <xsl:apply-templates select="*[not(self::Body)][1]" mode="text-form"/>
            <xsl:text>" shape=diamond style=filled fillcolor=yellow];&#13;&#10;</xsl:text>
        </xsl:template>


        <xsl:template match="Binary|Unary" mode="form">
            <xsl:apply-templates mode="text-form" select="."/>
```

```
</xsl:template>

<xsl:template match="Assignment" mode="form">

    <xsl:text>[label="</xsl:text>
    <xsl:apply-templates select="Left" mode="text-form"/>
    <xsl:text> = </xsl:text>
    <xsl:apply-templates select="Right" mode="text-form"/>
    <xsl:text>" shape=</xsl:text>
    <xsl:choose>
        <xsl:when test="Right/Read">
            <xsl:text>parallelogram</xsl:text>
        </xsl:when>
        <xsl:otherwise>
            <xsl:text>box</xsl:text>
        </xsl:otherwise>
    </xsl:choose>
    <xsl:text> style=filled fillcolor=</xsl:text>
    <xsl:choose>
        <xsl:when test="Right/Read">
            <xsl:text>green</xsl:text>
        </xsl:when>
        <xsl:otherwise>
            <xsl:text>white</xsl:text>
        </xsl:otherwise>
    </xsl:choose>
    <xsl:text>];&#13;&#10;</xsl:text>
</xsl:template>

<xsl:template match="YieldStatement" mode="form">
    <xsl:text>[label="Yield</xsl:text>
    <xsl:if test="*">
        <xsl:text>: </xsl:text>
        <xsl:apply-templates select="*" mode="text-form"/>
    </xsl:if>
    <xsl:text>" shape=box style=filled fillcolor=lightblue];&#13;&#10;</xsl:text>
</xsl:template>


<xsl:template match="BreakStatement" mode="form">
    <xsl:text>[label="Break" shape=box style=filled fillcolor=orange];&#13;&#10;</xsl:text>
</xsl:template>


<xsl:template match="CallStatement" mode="form">
    <xsl:apply-templates select="FunctionCall" mode="form"/>
</xsl:template>


<xsl:template match="FunctionCall" mode='form'>
    <xsl:text> [label="</xsl:text>
    <xsl:apply-templates select="." mode="text-form"/>
    <xsl:text>" shape=box fillcolor=paleturquoise style="filled,rounded"];&#13;&#10;</xsl:text>
</xsl:template>


<xsl:template match="VariableDeclaration" mode="form">
    <xsl:text>[label="</xsl:text>
    <xsl:if test="@constant = 'true'">
        <xsl:text>const </xsl:text>
```

```
        </xsl:if>
        <xsl:value-of select="Id/@name"/>

        <xsl:variable name="text">
            <xsl:apply-templates select="*[self::TypeReference or self::ListType or
                self::ArrayType or self::TupleType]" mode="type-form"/>
        </xsl:variable>

        <xsl:if test="string-length($text) > 0">
        <xsl:text>: </xsl:text>
            <xsl:value-of select="$text"/>
        </xsl:if>
        <xsl:text> = </xsl:text>
        <xsl:choose>
            <xsl:when test="Initializer">
                <xsl:apply-templates select="Initializer/*" mode="text-form"/>
            </xsl:when>
            <xsl:otherwise>
                <xsl:text>?</xsl:text>
            </xsl:otherwise>
        </xsl:choose>
        <xsl:text>" shape=box style=filled fillcolor=white];&#13;&#10;</xsl:text>

    </xsl:template>
```

5.2.5. Шаблоны режима `text-form`

Эти шаблоны преобразуют узлы AST в строки, которые затем вставляются в метки узлов.

```
    <xsl:template match="Literal" mode="text-form">
        <xsl:value-of select="@value"/>
    </xsl:template>


    <xsl:template match="ListLiteral" mode="text-form">
        <xsl:text>{</xsl:text>
        <xsl:for-each select="*[not(self::Span)]">
            <xsl:if test="position() > 1"><xsl:text>, </xsl:text></xsl:if>
            <xsl:apply-templates select="." mode="text-form"/>
        </xsl:for-each>
        <xsl:text>}</xsl:text>
    </xsl:template>


    <xsl:template match="TupleLiteral" mode="text-form">
        <xsl:text>(</xsl:text>
        <xsl:for-each select="Element">
            <xsl:if test="position() > 1"><xsl:text>, </xsl:text></xsl:if>
            <xsl:variable name="elemName" select="@name"/>
            <xsl:if test="$elemName != ''">
                <xsl:value-of select="$elemName"/>
                <xsl:text>: </xsl:text>
            </xsl:if>
            <xsl:apply-templates select="*" mode="text-form"/>
        </xsl:for-each>
        <xsl:text>)</xsl:text>
    </xsl:template>
```

```
<xsl:template match="ArrayLiteral" mode="text-form">
    <xsl:variable name="children" select="*[not(self::Span)]"/>
    <xsl:variable name="count" select="count($children)"/>
    <xsl:choose>
        <xsl:when test="$count = 0">
            <xsl:text>[]</xsl:text>
        </xsl:when>
        <xsl:otherwise>
            <xsl:text>[</xsl:text>
            <xsl:call-template name="format-array-elements">
                <xsl:with-param name="elements" select="$children"/>
                <xsl:with-param name="pos" select="1"/>
            </xsl:call-template>
            <xsl:text>]</xsl:text>
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>


<xsl:template name="format-array-elements">
    <xsl:param name="elements"/>
    <xsl:param name="pos"/>
    <xsl:param name="length" select="1"/>

    <xsl:variable name="current" select="$elements[$pos]"/>
    <xsl:variable name="text">
        <xsl:apply-templates select="$current" mode="text-form"/>
    </xsl:variable>
    <xsl:value-of select="$text"></xsl:value-of>
    <xsl:variable name="next_len">
        <xsl:choose>
            <xsl:when test="$length + string-length($text) + 2 > 30">
                <xsl:value-of select="0"/>
            </xsl:when>
            <xsl:otherwise>
                <xsl:value-of select="$length + string-length($text) + 2"></xsl:value-of>
            </xsl:otherwise>
        </xsl:choose>
    </xsl:variable>

    <xsl:if test="$pos &lt; count($elements)">
        <xsl:text>, </xsl:text>
        <xsl:if test="$next_len = 0">
            <xsl:text>\n</xsl:text>
        </xsl:if>
        <xsl:call-template name="format-array-elements">
            <xsl:with-param name="elements" select="$elements"/>
            <xsl:with-param name="pos" select="$pos + 1"/>
            <xsl:with-param name="length" select="$next_len"/>
        </xsl:call-template>
    </xsl:if>
</xsl:template>


<xsl:template match="DynamicArrayLiteral" mode="text-form">
    <xsl:text>[</xsl:text>
    <xsl:apply-templates select="./*[1]" mode="text-form"/>
    <xsl:text> => </xsl:text>
    <xsl:apply-templates select="./*[last()]" mode="text-form"/>
    <xsl:text>]</xsl:text>
```

```
        </xsl:template>


        <xsl:template match="Reference" mode="text-form">
            <xsl:value-of select="@name"/>
        </xsl:template>


        <xsl:template match="TupleElement" mode="text-form">
            <xsl:apply-templates select="Primary/*" mode="text-form"/>
            <xsl:variable name="type" select="Primary/Reference/@type"/>
            <xsl:variable name="index" select="Index/@value"/>
            <xsl:variable name="field">
                <xsl:call-template name="fieldName">
                    <xsl:with-param name="type" select="$type"/>
                    <xsl:with-param name="index" select="number($index)"/>
                </xsl:call-template>
            </xsl:variable>

            <xsl:choose>
                <xsl:when test="$field != ''">
                    <xsl:text>.</xsl:text>
                    <xsl:value-of select="$field"/>
                </xsl:when>
                <xsl:otherwise>
                    <xsl:text>.</xsl:text>
                    <xsl:value-of select="$index"/>
                </xsl:otherwise>
            </xsl:choose>
        </xsl:template>


        <xsl:template match="ArrayElement" mode="text-form">
            <xsl:apply-templates select="*[1]" mode="text-form"/>
            <xsl:text>[</xsl:text>
            <xsl:apply-templates select="*[2]" mode="text-form"/>
            <xsl:text>]</xsl:text>
        </xsl:template>


        <xsl:template name="fieldName">
            <xsl:param name="type"/>
            <xsl:param name="index"/>

            <xsl:variable name="typeDecl" select="$types[Id/@name = $type]"/>
            <xsl:if test="$typeDecl">
                <xsl:variable name="tupleType" select="$typeDecl/TupleType"/>
                <xsl:if test="$tupleType">
                    <xsl:variable name="element" select="$tupleType/Element[position() =
number($index)]"/>
                    <xsl:value-of select="$element/Id/@name"/>
                </xsl:if>
            </xsl:if>
        </xsl:template>


        <xsl:template match="Binary" mode="text-form">
            <xsl:variable name="leftOp" select="LeftOperand/*/Operator/@name"/>
            <xsl:variable name="rightOp" select="RightOperand/*/Operator/@name"/>
            <xsl:variable name="leftNeedsParens"
```

```
                select="local-name(LeftOperand/*) = 'Binary'
                        and not($leftOp = '*')
                        and not($leftOp = '/')
                        and not($leftOp = '\\')"/>
            <xsl:variable name="rightNeedsParens"
                select="local-name(RightOperand/*) = 'Binary'
                        and not($rightOp = '*')
                        and not($rightOp = '/')
                        and not($rightOp = '\\')"/>

            <xsl:if test="$leftNeedsParens">
                <xsl:text>(</xsl:text>
            </xsl:if>
            <xsl:apply-templates select="LeftOperand" mode="text-form">
                <xsl:with-param name="first" select="'1'"/>
            </xsl:apply-templates>
            <xsl:if test="$leftNeedsParens">
                <xsl:text>)</xsl:text>
            </xsl:if>

            <xsl:text> </xsl:text>
            <xsl:value-of select="Operator/@name"/>
            <xsl:text> </xsl:text>

            <xsl:if test="$rightNeedsParens">
                <xsl:text>(</xsl:text>
            </xsl:if>
            <xsl:apply-templates select="RightOperand" mode="text-form">
                <xsl:with-param name="first" select="'1'"/>
            </xsl:apply-templates>
            <xsl:if test="$rightNeedsParens">
                <xsl:text>)</xsl:text>
            </xsl:if>
        </xsl:template>


        <xsl:template match="Unary" mode="text-form">
            <xsl:value-of select="Operator/@name"/>
            <xsl:choose>
                <xsl:when test="Literal|Reference">
                    <xsl:apply-templates select="*[not(self::Span) and not(self::Operator)]" mode="text-
form"/>
                </xsl:when>
                <xsl:otherwise>
                    <xsl:text>(</xsl:text>
                    <xsl:apply-templates select="*[not(self::Span) and not(self::Operator)]" mode="text-
form"/>
                    <xsl:text>)</xsl:text>
                </xsl:otherwise>
            </xsl:choose>
        </xsl:template>


        <xsl:template match="Read" mode="text-form">
            <xsl:text>read</xsl:text>
        </xsl:template>


        <xsl:template match="FunctionCall" mode="text-form">
            <xsl:variable name="children" select="*[not(self::Span)]"/>
```

```
            <xsl:if test="count($children) > 0">
                <xsl:apply-templates select="$children[1]" mode="text-form"/>
                <xsl:text>(</xsl:text>
                <xsl:for-each select="$children[position() > 1]">
                    <xsl:if test="position() > 1"><xsl:text>, </xsl:text></xsl:if>
                    <xsl:apply-templates select="." mode="text-form"/>
                </xsl:for-each>
                <xsl:text>)</xsl:text>
            </xsl:if>
        </xsl:template>


        <xsl:template match="YieldStatement" mode="text-form">
            <xsl:text>yield</xsl:text>
            <xsl:if test="*">
                <xsl:text> </xsl:text>
                <xsl:apply-templates select="*" mode="text-form"/>
            </xsl:if>
        </xsl:template>


        <xsl:template match="Reference" mode="type-form">
            <xsl:value-of select="@name"/>
        </xsl:template>


        <xsl:template match="FunctionLiteral" mode="text-form">
            <xsl:text>func </xsl:text>
            <xsl:text>(</xsl:text>
            <xsl:call-template name="seqFunParams">
                <xsl:with-param name="nodes" select="FunctionSignature/Parameter"/>
                <xsl:with-param name="i" select="1"/>
            </xsl:call-template>
            <xsl:text>)</xsl:text>
            <xsl:if test="FunctionSignature/ReturnType">
                <xsl:text>: </xsl:text>
                <xsl:apply-templates select="FunctionSignature/ReturnType/TypeReference" mode="type-
form"/>
            </xsl:if>
            <xsl:text> => </xsl:text>
            <xsl:apply-templates select="*[not(self::FunctionSignature)]" mode="text-form"/>
        </xsl:template>


        <xsl:template match="TypeReference" mode="type-form">
            <xsl:value-of select="Id/@name"/>
        </xsl:template>


        <xsl:template match="ArrayType" mode="type-form">
            <xsl:text>[</xsl:text>
            <xsl:apply-templates select="*[self::TypeReference or self::TupleType or self::ListType or
self::ArrayType]" mode="type-form"/>
            <xsl:text>]</xsl:text>
        </xsl:template>


        <xsl:template match="ListType" mode="type-form">
            <xsl:text>{</xsl:text>
```

```
            <xsl:apply-templates select="*[self::TypeReference or self::TupleType or self::ListType or
self::ArrayType]" mode="type-form"/>
            <xsl:text>}</xsl:text>
        </xsl:template>


        <xsl:template match="TupleType" mode="type-form">
            <xsl:text>(</xsl:text>
            <xsl:for-each select="Element">
                <xsl:if test="position() > 1"><xsl:text>, </xsl:text></xsl:if>
                <xsl:if test="Id/@name">
                    <xsl:value-of select="Id/@name"/>
                    <xsl:text>: </xsl:text>
                </xsl:if>
                <xsl:apply-templates select="*[self::TypeReference or self::ListType or self::ArrayType or
                    self::TupleType]" mode="type-form"/>
            </xsl:for-each>
            <xsl:text>)</xsl:text>
        </xsl:template>


        <xsl:template match="FunctionType" mode="type-form">
            <xsl:text>func (</xsl:text>
            <xsl:call-template name="seqFunParams">
                <xsl:with-param name="nodes" select="FunctionSignature/Parameter"/>
                <xsl:with-param name="i" select="1"/>
            </xsl:call-template>
            <xsl:text>)</xsl:text>
            <xsl:variable name="text">
                 <xsl:apply-templates select="FunctionSignature/ReturnType/*" mode="type-form"/>
            </xsl:variable>
            <xsl:if test="string-length($text) > 0">
                <xsl:text>: </xsl:text>
                <xsl:value-of select="$text"/>
            </xsl:if>
        </xsl:template>
```

5.2.6. Шаблоны последовательных обходов

Данные шаблоны вызывают, практически, после каждого прохода рекурсивно вызывают себя после обработки каждого узла.

`seqRootVar` используется для вывода всех глобальных переменных и последовательного их соединения.

`seqFunParams` используется для вывода параметров, которые принимает конкретная функция.

`seqBody` используется для обработки всех переданных узлов и их последовательного соединения.

```
          <xsl:template name="seqRootVar">
            <xsl:param name="nodes"/>
            <xsl:param name="i"/>
            <xsl:param name="prev"/>

            <xsl:if test="$i &lt;= count($nodes)">
                <xsl:variable name="cur" select="$nodes[$i]"/>
                <xsl:apply-templates select="$cur">
```

```
                <xsl:with-param name="prev" select="$prev"/>
            </xsl:apply-templates>
            <xsl:call-template name="seqRootVar">
                <xsl:with-param name="nodes" select="$nodes"/>
                <xsl:with-param name="i" select="$i + 1"/>
                <xsl:with-param name="prev" select="$cur/Id/@name"/>
            </xsl:call-template>
        </xsl:if>
    </xsl:template>

    <xsl:template name="seqFunParams">
        <xsl:param name="nodes"/>

        <xsl:for-each select="$nodes">
            <xsl:if test="position() > 1">
                <xsl:text>, </xsl:text>
            </xsl:if>
            <xsl:value-of select="concat(Id/@name, ': ')"/>
            <xsl:apply-templates select="*" mode="type-form"/>
        </xsl:for-each>

    </xsl:template>

    <xsl:template name="seqBody">
        <xsl:param name="nodes"/>
        <xsl:param name="i"/>
        <xsl:param name="fun"/>
        <xsl:param name="fun_exit"/>
        <xsl:param name="next_el" select="''"/>
        <xsl:param name="break" select="$fun_exit"/>
        <xsl:param name="label" select="''"/>


        <xsl:variable name="cur" select="$nodes[$i]"/>

        <xsl:if test="$i &lt; count($nodes)">

            <xsl:variable name="next" select="$nodes[$i + 1]"/>

            <xsl:variable name="next_name">
                <xsl:call-template name="name">
                    <xsl:with-param name="node" select="$next"/>
                    <xsl:with-param name="fun" select="$fun"/>
                </xsl:call-template>
            </xsl:variable>

            <xsl:variable name="cur_name">
                <xsl:call-template name="name">
                    <xsl:with-param name="node" select="$cur"/>
                    <xsl:with-param name="fun" select="$fun"/>
                </xsl:call-template>
            </xsl:variable>

            <xsl:value-of select="concat('&#x09;', $cur_name, ' ')"/>
            <xsl:apply-templates select="$cur" mode="form"/>

            <xsl:if test="$i = 1 and $fun != 'main'">
                <xsl:call-template name="emit-edge">
                    <xsl:with-param name="from" select="$fun"/>
                    <xsl:with-param name="to" select="$cur_name"/>
```

```
            <xsl:with-param name="label" select="$label"/>
        </xsl:call-template>
    </xsl:if>

    <xsl:choose>

        <xsl:when test="local-name($cur)='IfStatement'">

            <xsl:apply-templates select="$cur">
                <xsl:with-param name="fun" select="$fun"/>
                <xsl:with-param name="fun_exit" select="$fun_exit"/>
                <xsl:with-param name="cur" select="$cur_name"/>
                <xsl:with-param name="next" select="$next_name"/>
                <xsl:with-param name="break" select="$break"/>
            </xsl:apply-templates>

            <xsl:call-template name="seqBody">
                <xsl:with-param name="nodes" select="$nodes"/>
                <xsl:with-param name="i" select="$i+1"/>
                <xsl:with-param name="fun" select="$fun"/>
                <xsl:with-param name="fun_exit" select="$fun_exit"/>
                <xsl:with-param name="next_el" select="$next_el"/>
                <xsl:with-param name="break" select="$break"/>
            </xsl:call-template>

        </xsl:when>

        <xsl:when test="local-name($cur)='ForStatement'">
            <xsl:choose>
                <xsl:when test="$next_el != ''">
                    <xsl:apply-templates select="$cur">
                        <xsl:with-param name="fun" select="$fun"/>
                        <xsl:with-param name="fun_exit" select="$fun_exit"/>
                        <xsl:with-param name="cur" select="$cur_name"/>
                        <xsl:with-param name="next" select="$next_name"/>
                        <xsl:with-param name="next_el" select="$next_name"/>
                    </xsl:apply-templates>
                </xsl:when>
                <xsl:otherwise>
                    <xsl:apply-templates select="$cur">
                        <xsl:with-param name="fun" select="$fun"/>
                        <xsl:with-param name="fun_exit" select="$fun_exit"/>
                        <xsl:with-param name="cur" select="$cur_name"/>
                        <xsl:with-param name="next" select="$next_name"/>
                        <xsl:with-param name="next_el" select="$next_name"/>
                    </xsl:apply-templates>
                </xsl:otherwise>
            </xsl:choose>

            <xsl:call-template name="seqBody">
                <xsl:with-param name="nodes" select="$nodes"/>
                <xsl:with-param name="i" select="$i+1"/>
                <xsl:with-param name="fun" select="$fun"/>
                <xsl:with-param name="fun_exit" select="$fun_exit"/>
                <xsl:with-param name="next_el" select="$next_el"/>
                <xsl:with-param name="break" select="$break"/>
            </xsl:call-template>
        </xsl:when>

        <xsl:when test="local-name($cur)='WhileStatement'">
```

```
                    <xsl:apply-templates select="$cur">
                        <xsl:with-param name="fun" select="$fun"/>
                        <xsl:with-param name="fun_exit" select="$fun_exit"/>
                        <xsl:with-param name="cur" select="$cur_name"/>
                        <xsl:with-param name="next" select="$next_name"/>
                        <xsl:with-param name="break" select="$break"/>
                    </xsl:apply-templates>
                    <xsl:call-template name="seqBody">
                        <xsl:with-param name="nodes" select="$nodes"/>
                        <xsl:with-param name="i" select="$i+1"/>
                        <xsl:with-param name="fun" select="$fun"/>
                        <xsl:with-param name="fun_exit" select="$fun_exit"/>
                        <xsl:with-param name="next_el" select="$next_el"/>
                        <xsl:with-param name="break" select="$break"/>
                    </xsl:call-template>
                </xsl:when>

                <xsl:when test="local-name($cur)='ReturnStatement'">
                    <xsl:call-template name="emit-edge">
                        <xsl:with-param name="from" select="$cur_name"/>
                        <xsl:with-param name="to" select="$fun_exit"/>
                    </xsl:call-template>

                    <xsl:if test="$fun_exit != 'exit'">
                        <xsl:call-template name="emit-exit-filled">
                            <xsl:with-param name="node" select="$fun_exit"/>
                        </xsl:call-template>
                    </xsl:if>
                </xsl:when>

                <xsl:when test="local-name($cur)='BreakStatement'">
                    <xsl:call-template name="emit-edge">
                        <xsl:with-param name="from" select="$cur_name"/>
                        <xsl:with-param name="to" select="$break"/>
                    </xsl:call-template>
                </xsl:when>

                <xsl:otherwise>

                    <xsl:call-template name="emit-edge">
                        <xsl:with-param name="from" select="$cur_name"/>
                        <xsl:with-param name="to" select="$next_name"/>
                    </xsl:call-template>

                    <xsl:call-template name="seqBody">
                        <xsl:with-param name="nodes" select="$nodes"/>
                        <xsl:with-param name="i" select="$i+1"/>
                        <xsl:with-param name="fun" select="$fun"/>
                        <xsl:with-param name="fun_exit" select="$fun_exit"/>
                        <xsl:with-param name="next_el" select="$next_el"/>
                        <xsl:with-param name="break" select="$break"/>
                    </xsl:call-template>

                </xsl:otherwise>

            </xsl:choose>

        </xsl:if>

        <xsl:if test="$i = count($nodes)">
```

```
<xsl:variable name="cur_name">
    <xsl:call-template name="name">
        <xsl:with-param name="node" select="$cur"/>
        <xsl:with-param name="fun" select="$fun"/>
    </xsl:call-template>
</xsl:variable>

<xsl:value-of select="concat('&#x09;', $cur_name, ' ')"/>
<xsl:apply-templates select="$cur" mode="form"/>

<xsl:if test="$i = 1 and $fun != 'main'">

    <xsl:call-template name="emit-edge">
        <xsl:with-param name="from" select="$fun"/>
        <xsl:with-param name="to" select="$cur_name"/>
        <xsl:with-param name="label" select="$label"/>
    </xsl:call-template>

</xsl:if>

<xsl:choose>

    <xsl:when test="local-name($cur)='IfStatement'">

        <xsl:apply-templates select="$cur">
            <xsl:with-param name="fun" select="$fun"/>
            <xsl:with-param name="fun_exit" select="$fun_exit"/>
            <xsl:with-param name="cur" select="$cur_name"/>
            <xsl:with-param name="next" select="$next_el"/>
            <xsl:with-param name="next_el" select="$next_el"/>
            <xsl:with-param name="break" select="$break"/>
        </xsl:apply-templates>

    </xsl:when>

    <xsl:when test="local-name($cur)='ForStatement'">
        <xsl:variable name="nxt">
            <xsl:choose>
                <xsl:when test="$next_el != ''">
                    <xsl:value-of select="$next_el"></xsl:value-of>
                </xsl:when>
                <xsl:otherwise>
                    <xsl:value-of select="$fun_exit"></xsl:value-of>
                </xsl:otherwise>
            </xsl:choose>
        </xsl:variable>
        <xsl:apply-templates select="$cur">
            <xsl:with-param name="fun" select="$fun"/>
            <xsl:with-param name="fun_exit" select="$fun_exit"/>
            <xsl:with-param name="cur" select="$cur_name"/>
            <xsl:with-param name="next" select="$nxt"/>
            <xsl:with-param name="next_el" select="$nxt"/>
        </xsl:apply-templates>
    </xsl:when>

    <xsl:when test="local-name($cur)='WhileStatement'">
        <xsl:apply-templates select="$cur">
            <xsl:with-param name="fun" select="$fun"/>
            <xsl:with-param name="fun_exit" select="$fun_exit"/>
```

```
                        <xsl:with-param name="cur" select="$cur_name"/>
                        <xsl:with-param name="next" select="$next_name"/>
                        <xsl:with-param name="break" select="$break"/>
                    </xsl:apply-templates>
                </xsl:when>

                <xsl:when test="local-name($cur)='ReturnStatement'">
                    <xsl:call-template name="emit-edge">
                        <xsl:with-param name="from" select="$cur_name"/>
                        <xsl:with-param name="to" select="$fun_exit"/>
                    </xsl:call-template>
                    <xsl:if test="$fun_exit != 'exit'">
                        <xsl:call-template name="emit-exit-filled">
                            <xsl:with-param name="node" select="$fun_exit"/>
                        </xsl:call-template>
                    </xsl:if>
                </xsl:when>

                <xsl:when test="local-name($cur)='BreakStatement'">
                    <xsl:call-template name="emit-edge">
                        <xsl:with-param name="from" select="$cur_name"/>
                        <xsl:with-param name="to" select="$break"/>
                    </xsl:call-template>

                    <xsl:if test="contains($break, '_exit')">
                        <xsl:call-template name="emit-exit-filled">
                            <xsl:with-param name="node" select="$break"/>
                        </xsl:call-template>
                    </xsl:if>
                </xsl:when>

                <xsl:otherwise>

                    <xsl:value-of select="concat('&#x09;', $cur_name, ' -> ')"/>
                    <xsl:choose>
                        <xsl:when test="$next_el != ''">
                            <xsl:value-of select="$next_el"></xsl:value-of>
                            <xsl:text>;&#13;&#10;</xsl:text>
                        </xsl:when>
                        <xsl:otherwise>
                            <xsl:value-of select="$fun_exit"></xsl:value-of>
                            <xsl:text>;&#13;&#10;</xsl:text>
                            <xsl:if test="$fun_exit != 'exit'">
                                <xsl:call-template name="emit-exit-filled">
                                    <xsl:with-param name="node" select="$fun_exit"/>
                                </xsl:call-template>
                            </xsl:if>
                        </xsl:otherwise>
                    </xsl:choose>

                </xsl:otherwise>

            </xsl:choose>

        </xsl:if>

    </xsl:template>
```

5.3. Шаблоны узлов

5.3.1. VariableDeclaration

```
    <xsl:template match="VariableDeclaration">
      <xsl:param name="prev"/>

      <xsl:text>&#x09;</xsl:text>
      <xsl:value-of select="Id/@name"/>
      <xsl:text> [label="</xsl:text>
      <xsl:if test="@constant = 'true'">
          <xsl:text>const </xsl:text>
      </xsl:if>
      <xsl:value-of select="Id/@name"/>

      <xsl:variable name="text">
          <xsl:apply-templates select="*[self::TypeReference or self::ListType or
              self::ArrayType or self::TupleType]" mode="type-form"/>
      </xsl:variable>

      <xsl:if test="string-length($text) > 0">
      <xsl:text>: </xsl:text>
          <xsl:value-of select="$text"/>
      </xsl:if>
      <xsl:text> = </xsl:text>

      <xsl:choose>
          <xsl:when test="Initializer">
              <xsl:apply-templates select="Initializer/*" mode="text-form"/>
          </xsl:when>
          <xsl:otherwise>
              <xsl:text>?</xsl:text>
          </xsl:otherwise>
      </xsl:choose>

      <xsl:text>" fillcolor=white shape=box];&#13;&#10;</xsl:text>

      <xsl:if test="$prev != ''">
          <xsl:call-template name="emit-edge">
              <xsl:with-param name="from" select="$prev"/>
              <xsl:with-param name="to" select="Id/@name"/>
          </xsl:call-template>
      </xsl:if>
    </xsl:template>
```

5.3.2. IfStatement и ElseIfStatement

```
    <xsl:template match="IfStatement">
      <xsl:param name="fun"/>
      <xsl:param name="fun_exit"/>
      <xsl:param name="cur"/>
      <xsl:param name="next"/>
      <xsl:param name="break" select="$fun_exit"/>


      <!-- If -->
       <xsl:choose>

          <xsl:when test="Body/*[not(self::TypeDeclaration)]">
              <xsl:call-template name="seqBody">
                  <xsl:with-param name="nodes" select="Body/*[not(self::TypeDeclaration)]"/>
```

```
                <xsl:with-param name="i" select="1"/>
                <xsl:with-param name="fun" select="$cur"/>
                <xsl:with-param name="fun_exit" select="$fun_exit"/>
                <xsl:with-param name="next_el" select="$next"/>
                <xsl:with-param name="label" select="'true'"/>
                <xsl:with-param name="break" select="$break"/>
            </xsl:call-template>
        </xsl:when>

        <xsl:otherwise>
            <xsl:value-of select="concat('&#x09;', $cur, ' -> ')"/>
            <xsl:choose>
                <xsl:when test="$next  != ''">
                    <xsl:value-of select="$next"/>
                </xsl:when>
                <xsl:otherwise>
                    <xsl:value-of select="$fun_exit"/>
                </xsl:otherwise>
            </xsl:choose>
            <xsl:text> [label=&quot;true&quot;];&#13;&#10;</xsl:text>
        </xsl:otherwise>

    </xsl:choose>

    <!-- ElseIf -->
    <xsl:call-template name="seqIfElse">
        <xsl:with-param name="nodes" select="./ElseIfStatement"/>
        <xsl:with-param name="i" select="1"/>
        <xsl:with-param name="fun" select="$cur"/>
        <xsl:with-param name="init" select="$cur"/>
        <xsl:with-param name="fun_exit" select="$fun_exit"/>
        <xsl:with-param name="next" select="$next"/>
        <xsl:with-param name="break" select="$break"/>
    </xsl:call-template>

    <xsl:variable name="last_cond">
        <xsl:choose>
            <xsl:when test="count(ElseIfStatement) > 0">
                <xsl:variable name="last_elseif" select="./ElseIfStatement[last()]"/>
                <xsl:call-template name="name">
                    <xsl:with-param name="node" select="$last_elseif"/>
                    <xsl:with-param name="fun" select="$cur"/>
                </xsl:call-template>
            </xsl:when>
            <xsl:otherwise>
                <xsl:value-of select="$cur"/>
            </xsl:otherwise>
        </xsl:choose>
    </xsl:variable>

    <!-- Else -->
    <xsl:choose>
        <xsl:when test="count(ElseBody) > 0">

            <xsl:call-template name="seqBody">
                <xsl:with-param name="nodes" select="./ElseBody/*[not(self::TypeDeclaration)]"/>
                <xsl:with-param name="i" select="1"/>
                <xsl:with-param name="fun" select="$last_cond"/>
                <xsl:with-param name="fun_exit" select="$fun_exit"/>
                <xsl:with-param name="next_el" select="$next"/>
```

```
                    <xsl:with-param name="label" select="'false'"/>
                    <xsl:with-param name="break" select="$break"/>
                </xsl:call-template>

            </xsl:when>

            <xsl:otherwise>

                <xsl:call-template name="emit-edge">
                    <xsl:with-param name="from" select="$last_cond"/>
                    <xsl:with-param name="to" select="$next"/>
                    <xsl:with-param name="label" select="'false'"/>
                </xsl:call-template>

            </xsl:otherwise>
        </xsl:choose>
    </xsl:template>

    <!-- ELSEIF STATEMENT -->
    <xsl:template match="ElseIfStatement">
        <xsl:param name="fun"/>
        <xsl:param name="fun_exit"/>
        <xsl:param name="cur"/>
        <xsl:param name="next"/>
        <xsl:param name="break" select="$fun_exit"/>

        <xsl:choose>
            <xsl:when test="count(Body/*) > 0">
                <xsl:call-template name="seqBody">
                    <xsl:with-param name="nodes" select="Body/*[not(self::TypeDeclaration)]"/>
                    <xsl:with-param name="i" select="1"/>
                    <xsl:with-param name="fun" select="$cur"/>
                    <xsl:with-param name="fun_exit" select="$fun_exit"/>
                    <xsl:with-param name="next_el" select="$next"/>
                    <xsl:with-param name="label" select="'true'"/>
                    <xsl:with-param name="break" select="$break"/>
                </xsl:call-template>
            </xsl:when>
            <xsl:otherwise>
                <xsl:call-template name="emit-edge">
                    <xsl:with-param name="from" select="$cur"/>
                    <xsl:with-param name="to" select="$next"/>
                    <xsl:with-param name="label" select="'true'"/>
                </xsl:call-template>
            </xsl:otherwise>
        </xsl:choose>
    </xsl:template>

    <xsl:template name="seqIfElse">
        <xsl:param name="nodes"/>
        <xsl:param name="i"/>
        <xsl:param name="init"/>
        <xsl:param name="fun"/>
        <xsl:param name="fun_exit"/>
        <xsl:param name="next"/>
        <xsl:param name="break" select="$fun_exit"/>

        <xsl:if test="count($nodes) &gt;= $i">
            <xsl:variable name="cur" select="$nodes[$i]"/>
```

```
            <xsl:variable name="cur_name">
                <xsl:call-template name="name">
                    <xsl:with-param name="node" select="$cur"/>
                    <xsl:with-param name="fun" select="$init"/>
                </xsl:call-template>
            </xsl:variable>

            <xsl:value-of select="concat('&#x09;', $cur_name, ' ')"/>
            <xsl:apply-templates select="$cur" mode="form"/>

            <xsl:call-template name="emit-edge">
                <xsl:with-param name="from" select="$fun"/>
                <xsl:with-param name="to" select="$cur_name"/>
                <xsl:with-param name="label" select="'false'"/>
            </xsl:call-template>

            <xsl:apply-templates select="$cur">
                <xsl:with-param name="fun" select="$fun"/>
                <xsl:with-param name="fun_exit" select="$fun_exit"/>
                <xsl:with-param name="cur" select="$cur_name"/>
                <xsl:with-param name="next" select="$next"/>
                <xsl:with-param name="break" select="$break"/>
            </xsl:apply-templates>

            <xsl:call-template name="seqIfElse">
                <xsl:with-param name="nodes" select="$nodes"/>
                <xsl:with-param name="i" select="$i + 1"/>
                <xsl:with-param name="fun" select="$cur_name"/>
                <xsl:with-param name="init" select="$init"/>
                <xsl:with-param name="fun_exit" select="$fun_exit"/>
                <xsl:with-param name="break" select="$break"/>
            </xsl:call-template>
        </xsl:if>

    </xsl:template>
```

5.3.3. ForStatement

```
    <xsl:template match="ForStatement">
        <xsl:param name="fun"/>
        <xsl:param name="fun_exit"/>
        <xsl:param name="cur"/>
        <xsl:param name="next"/>
        <xsl:param name="next_el" select="''"/>

        <xsl:variable name="inf" select="not(RightExpression) and not(LeftExpression)"/>
        <xsl:choose>
            <xsl:when test="$inf">
                <xsl:choose>
                    <xsl:when test="Body/*[not(self::TypeDeclaration)]">
                        <xsl:call-template name="seqBody">
                            <xsl:with-param name="nodes" select="Body/*[not(self::TypeDeclaration)]"/>
                            <xsl:with-param name="i" select="1"/>
                            <xsl:with-param name="fun" select="$cur"/>
                            <xsl:with-param name="fun_exit" select="$fun_exit"/>
                            <xsl:with-param name="next_el" select="$cur"/>
                            <xsl:with-param name="break" select="$next"/>
                            <xsl:with-param name="label" select="''"/>
                        </xsl:call-template>
                    </xsl:when>
```

```
                <xsl:otherwise>
                    <xsl:call-template name="emit-edge">
                        <xsl:with-param name="from" select="$cur"/>
                        <xsl:with-param name="to" select="$cur"/>
                    </xsl:call-template>
                </xsl:otherwise>
            </xsl:choose>
        </xsl:when>

        <xsl:otherwise>
            <xsl:variable name="cond_name" select="concat($cur, '_cond')"/>

            <xsl:variable name="iter_label" select="concat(Id/@name, '_iterator')"/>

            <xsl:value-of select="concat('&#x09;', $cond_name, ' [label=&quot;')"/>
            <xsl:choose>
                <xsl:when test="not(RightExpression)">
                    <xsl:value-of select="concat($iter_label, '.MoveNext()')"/>
                </xsl:when>

                <xsl:otherwise>
                    <xsl:variable name="op">
                        <xsl:choose>
                            <xsl:when test="@reverse='true'">>=</xsl:when>
                            <xsl:otherwise>&lt;=</xsl:otherwise>
                        </xsl:choose>
                    </xsl:variable>
                    <xsl:value-of select="Id/@name"/>
                    <xsl:value-of select="concat(' ', $op, ' ')"/>

                    <xsl:variable name="right_text">
                        <xsl:choose>
                            <xsl:when test="@reverse='true'">
                                <xsl:apply-templates select="LeftExpression" mode="text-form"/>
                            </xsl:when>
                            <xsl:otherwise>
                                <xsl:apply-templates select="RightExpression" mode="text-form"/>
                            </xsl:otherwise>
                        </xsl:choose>
                    </xsl:variable>

                    <xsl:value-of select="$right_text"/>
                </xsl:otherwise>
            </xsl:choose>

            <xsl:text>&quot; shape=diamond style=filled fillcolor=yellow];&#13;&#10;</xsl:text>


            <xsl:call-template name="emit-edge">
                <xsl:with-param name="from" select="$cur"/>
                <xsl:with-param name="to" select="$cond_name"/>
            </xsl:call-template>

            <xsl:variable name="inc">
                <xsl:element name="Assignment">
                    <xsl:element name="Left">
                        <xsl:element name="Reference">
                            <xsl:attribute name="name">
                                <xsl:value-of select="Id/@name"/>
```

```
                                </xsl:attribute>
                            </xsl:element>
                        </xsl:element>

                        <xsl:element name="Right">
                            <xsl:element name="Binary">
                                <xsl:element name="Operator">
                                    <xsl:choose>
                                        <xsl:when test="@reverse='false'">
                                            <xsl:attribute name="name">+</xsl:attribute>
                                        </xsl:when>
                                        <xsl:otherwise>
                                            <xsl:attribute name="name">-</xsl:attribute>
                                        </xsl:otherwise>
                                    </xsl:choose>
                                </xsl:element>
                                <xsl:element name="LeftOperand">
                                    <xsl:element name="Reference">
                                        <xsl:attribute name="name">
                                            <xsl:value-of select="Id/@name"/>
                                        </xsl:attribute>
                                    </xsl:element>
                                </xsl:element>
                                <xsl:element name="RightOperand">
                                    <xsl:element name="Literal">
                                        <xsl:attribute name="value">1</xsl:attribute>
                                    </xsl:element>
                                </xsl:element>
                            </xsl:element>
                        </xsl:element>
                    </xsl:element>
                </xsl:variable>

                <xsl:variable name="body-nodes">
                    <xsl:for-each select="Body/*[not(self::TypeDeclaration)]">
                        <xsl:copy-of select="."/>
                    </xsl:for-each>
                    <xsl:if test="RightExpression">
                        <xsl:copy-of select="$inc"/>
                    </xsl:if>
                </xsl:variable>
                <xsl:variable name="nodes" select="exslt:node-set($body-nodes)/*"/>

                <xsl:if test="Guard">
                    <xsl:variable name="guard_name" select="concat($cur, '_guard')"/>

                    <xsl:if test="not(RightExpression)">
                        <xsl:variable name="iter_name" select="concat($cur, '_iter')"/>
                        <xsl:value-of select="concat('&#x09;', $iter_name, ' [label=&quot;')"/>
                        <xsl:value-of select="concat(Id/@name, ' = ', $iter_label, '.Current')"/>
                        <xsl:text>&quot; shape=box style=filled fillcolor=white];&#13;&#10;</xsl:text>
                    </xsl:if>

                    <xsl:value-of select="concat('&#x09;', $guard_name, ' [label=&quot;')"/>
                    <xsl:apply-templates select="Guard/Binary" mode="form"/>
                    <xsl:text>&quot; shape=diamond style=filled
fillcolor=yellow];&#13;&#10;</xsl:text>

                    <xsl:choose>
                        <xsl:when test="not(RightExpression)">
```

```
                    <xsl:variable name="iter_name" select="concat($cur, '_iter')"/>
                    <xsl:call-template name="emit-edge">
                        <xsl:with-param name="from" select="$cond_name"/>
                        <xsl:with-param name="to" select="$iter_name"/>
                        <xsl:with-param name="label" select="'true'"/>
                    </xsl:call-template>
                    <xsl:call-template name="emit-edge">
                        <xsl:with-param name="from" select="$iter_name"/>
                        <xsl:with-param name="to" select="$guard_name"/>
                    </xsl:call-template>
                </xsl:when>
                <xsl:otherwise>
                    <xsl:call-template name="emit-edge">
                        <xsl:with-param name="from" select="$cond_name"/>
                        <xsl:with-param name="to" select="$guard_name"/>
                        <xsl:with-param name="label" select="'true'"/>
                    </xsl:call-template>
                </xsl:otherwise>
            </xsl:choose>

            <xsl:call-template name="emit-edge">
                <xsl:with-param name="from" select="$cond_name"/>
                <xsl:with-param name="to" select="$next_el"/>
                <xsl:with-param name="label" select="'false'"/>
            </xsl:call-template>

            <xsl:call-template name="emit-exit-filled">
                <xsl:with-param name="node" select="$next_el"/>
            </xsl:call-template>

            <xsl:call-template name="seqBody">
                <xsl:with-param name="nodes" select="$nodes"/>
                <xsl:with-param name="i" select="1"/>
                <xsl:with-param name="fun" select="$guard_name"/>
                <xsl:with-param name="fun_exit" select="$fun_exit"/>
                <xsl:with-param name="next_el" select="$cond_name"/>
                <xsl:with-param name="label" select="'true'"/>
                <xsl:with-param name="break" select="$next"/>
            </xsl:call-template>

            <xsl:call-template name="emit-edge">
                <xsl:with-param name="from" select="$guard_name"/>
                <xsl:with-param name="to" select="$cond_name"/>
                <xsl:with-param name="label" select="'false'"/>
            </xsl:call-template>
        </xsl:if>

        <xsl:if test="not(Guard)">
            <xsl:choose>
                <xsl:when test="not(RightExpression)">
                    <xsl:variable name="iter_name" select="concat($cur, '_iter')"/>
                    <xsl:variable name="body_with_iter_rtf">
                        <xsl:element name="Assignment">
                            <xsl:element name="Left">
                                <xsl:element name="Reference">
                                    <xsl:attribute name="name">
                                        <xsl:value-of select="Id/@name"/>
                                    </xsl:attribute>
                                </xsl:element>
                            </xsl:element>
```

```
                                    <xsl:element name="Right">
                                        <xsl:element name="Reference">
                                            <xsl:attribute name="name">
                                                <xsl:value-of select="$iter_label"/>
                                                <xsl:text>.Current</xsl:text>
                                            </xsl:attribute>
                                        </xsl:element>
                                    </xsl:element>
                                </xsl:element>
                                <xsl:for-each select="Body/*[not(self::TypeDeclaration)]">
                                    <xsl:copy-of select="."/>
                                </xsl:for-each>
                            </xsl:variable>
                            <xsl:variable name="body_nodes" select="exslt:node-
set($body_with_iter_rtf)/*"/>
                            <xsl:call-template name="seqBody">
                                <xsl:with-param name="nodes" select="$body_nodes"/>
                                <xsl:with-param name="i" select="1"/>
                                <xsl:with-param name="fun" select="$cond_name"/>
                                <xsl:with-param name="fun_exit" select="$fun_exit"/>
                                <xsl:with-param name="next_el" select="$cond_name"/>
                                <xsl:with-param name="label" select="'true'"/>
                                <xsl:with-param name="break" select="$next"/>
                            </xsl:call-template>
                        </xsl:when>
                        <xsl:otherwise>
                            <xsl:call-template name="seqBody">
                                <xsl:with-param name="nodes" select="$nodes"/>
                                <xsl:with-param name="i" select="1"/>
                                <xsl:with-param name="fun" select="$cond_name"/>
                                <xsl:with-param name="fun_exit" select="$fun_exit"/>
                                <xsl:with-param name="next_el" select="$cond_name"/>
                                <xsl:with-param name="label" select="'true'"/>
                                <xsl:with-param name="break" select="$next"/>
                            </xsl:call-template>
                        </xsl:otherwise>
                    </xsl:choose>

                    <xsl:variable name="nxt">
                        <xsl:choose>
                            <xsl:when test="$next_el"><xsl:value-of select="$next_el"/></xsl:when>
                            <xsl:otherwise>
                                <xsl:value-of select="$fun_exit"/>
                            </xsl:otherwise>
                        </xsl:choose>
                    </xsl:variable>
                    <xsl:call-template name="emit-edge">
                        <xsl:with-param name="from" select="$cond_name"/>
                        <xsl:with-param name="to" select="$nxt"/>
                        <xsl:with-param name="label" select="'false'"/>
                    </xsl:call-template>

                    <xsl:call-template name="emit-exit-filled">
                        <xsl:with-param name="node" select="$nxt"/>
                    </xsl:call-template>
                </xsl:if>
            </xsl:otherwise>
        </xsl:choose>
```

```
    </xsl:template>
```

5.3.4. WhileStatement

```
    <xsl:template match="WhileStatement">
        <xsl:param name="fun"/>
        <xsl:param name="fun_exit"/>
        <xsl:param name="cur"/>
        <xsl:param name="next"/>
        <xsl:param name="break" select="$next"/>

        <xsl:choose>
            <xsl:when test="Body/*[not(self::TypeDeclaration)]">
                <xsl:call-template name="seqBody">
                    <xsl:with-param name="nodes" select="Body/*[not(self::TypeDeclaration)]"/>
                    <xsl:with-param name="i" select="1"/>
                    <xsl:with-param name="fun" select="$cur"/>
                    <xsl:with-param name="fun_exit" select="$fun_exit"/>
                    <xsl:with-param name="next_el" select="$cur"/>
                    <xsl:with-param name="break" select="$next"/>
                    <xsl:with-param name="label" select="'true'"/>
                </xsl:call-template>
            </xsl:when>

            <xsl:otherwise>
                <xsl:value-of select="concat('&#x09;', $cur, ' -> ', $cur)"/>
                <xsl:text> [label=&quot;true&quot;];&#13;&#10;</xsl:text>
            </xsl:otherwise>
        </xsl:choose>

        <xsl:call-template name="emit-edge">
            <xsl:with-param name="from" select="$cur"/>
            <xsl:with-param name="to" select="$next"/>
            <xsl:with-param name="label" select="'false'"/>
        </xsl:call-template>
    </xsl:template>
```

5.3.5. FunctionDeclaration

```
    <xsl:template match="FunctionDeclaration">
        <xsl:variable name="name" select="Id/@name"/>
        <xsl:text>&#x09;</xsl:text>
        <xsl:value-of select="$name"/>
        <xsl:text> [label="fun </xsl:text>
        <xsl:value-of select="$name"/>
        <xsl:text>(</xsl:text>
        <xsl:call-template name="seqFunParams">
            <xsl:with-param name="nodes" select="FunctionSignature/Parameter"/>
            <xsl:with-param name="i" select="1"/>
        </xsl:call-template>
        <xsl:text>): </xsl:text>
        <xsl:choose>
            <xsl:when test="count(FunctionSignature/ReturnType) &gt; 0">
                <xsl:apply-templates select="FunctionSignature/ReturnType/*" mode="type-form"/>
            </xsl:when>
            <xsl:otherwise>
                <xsl:text>void</xsl:text>
            </xsl:otherwise>
        </xsl:choose>
        <xsl:text>", shape=ellipse, fillcolor=darkgreen, style=filled];&#13;&#10;</xsl:text>
```

```
            <xsl:text>&#x09;</xsl:text>
            <xsl:value-of select="concat($name, '_exit')"/>
            <xsl:text> [label="fun </xsl:text>
            <xsl:value-of select="$name"/>
            <xsl:text> exit", shape=ellipse, fillcolor=darkgreen, style=invis];&#13;&#10;</xsl:text>

            <xsl:choose>
                <xsl:when test="FunctionLiteral">
                    <xsl:variable name="retName" select="concat($name, '_return_literal')"/>
                    <xsl:value-of select="concat('&#x09;', $retName, ' [label=&quot;Return: ')" />
                    <xsl:apply-templates select="FunctionLiteral" mode="text-form"/>
                    <xsl:text>&quot; shape=box style="filled,rounded"
fillcolor=lightcoral];&#13;&#10;</xsl:text>
                    <xsl:value-of select="concat('&#x09;', $name, ' -> ', $retName, ';&#13;&#10;')"/>
                    <xsl:value-of select="concat('&#x09;', $retName, ' -> ', $name, '_exit',
';&#13;&#10;')"/>
                    <xsl:value-of select="concat('&#x09;', $name, '_exit
[style=&quot;filled&quot;];&#13;&#10;')"/>
                </xsl:when>
                <xsl:otherwise>
                    <xsl:choose>
                        <xsl:when test="Body/*[not(self::TypeDeclaration)]">
                            <xsl:call-template name="seqBody">
                                <xsl:with-param name="nodes" select="Body/*[not(self::TypeDeclaration)]"/>
                                <xsl:with-param name="i" select="1"/>
                                <xsl:with-param name="fun" select="$name"/>
                                <xsl:with-param name="fun_exit" select="concat($name, '_exit')"/>
                            </xsl:call-template>
                        </xsl:when>

                        <xsl:otherwise>
                            <xsl:value-of select="concat('&#x09;', $name, ' -> ', $name,
'_exit;&#13;&#10;')"/>
                            <xsl:value-of select="concat('&#x09;', $name, '_exit
[style=&quot;filled&quot;];&#13;&#10;')"/>
                        </xsl:otherwise>
                    </xsl:choose>

                </xsl:otherwise>
            </xsl:choose>
        </xsl:template>

        <xsl:template match="FunctionDeclaration" mode="main">
            <xsl:param name="prev"/>

            <xsl:variable name="name" select="Id/@name"/>


            <xsl:value-of select="concat('&#x09;', $prev, ' -> ' )"/>
            <xsl:call-template name="name">
                <xsl:with-param name="node" select="Body/*[not(self::TypeDeclaration)][1]"/>
                <xsl:with-param name="fun" select="'main'"/>
            </xsl:call-template>
            <xsl:text>;&#13;&#10;</xsl:text>


            <xsl:call-template name="seqBody">
                <xsl:with-param name="nodes" select="Body/*[not(self::TypeDeclaration)]"/>
                <xsl:with-param name="i" select="1"/>
```

```
                <xsl:with-param name="fun" select="$name"/>
                <xsl:with-param name="fun_exit" select="'exit'"/>
            </xsl:call-template>
        </xsl:template>
```

6. Пример

Код на Simple

```
type student is (name: {Int}, surname: {Int}, GPA: Real);

const students: [ student ] := [
    (name: {73, 118, 97, 110}, surname: {73, 118, 97, 110, 111, 118}, GPA: 4.3),
    (name: {75, 115, 117, 115, 104, 97}, surname: {83, 109, 105, 114, 110, 111, 118, 97}, GPA: 4.8),
    (name: {76, 101, 115, 104, 97}, surname: {80, 111, 112, 111, 118, 105, 99, 104}, GPA: 3.3)
];

func main()
    for student_var in students if student_var.GPA >= 4.5 loop
        print student_var.name, student_var.surname;
    end;
end
```

XML представление для AST данного кода

```
<Program>
    <TypeDeclaration>
        <Id name="#001" />
        <ListType>
            <TypeReference>
                <Id name="Int" />
            </TypeReference>
        </ListType>
    </TypeDeclaration>
    <TypeDeclaration>
        <Id name="student" begin="1:6" end="1:13" />
        <TupleType>
            <Element>
                <Id name="name" />
                <ListType>
                    <TypeReference>
                        <Id name="Int" />
                    </TypeReference>
                </ListType>
            </Element>
            <Element>
                <Id name="surname" />
                <ListType>
                    <TypeReference>
                        <Id name="Int" />
                    </TypeReference>
                </ListType>
            </Element>
            <Element>
                <Id name="GPA" />
                <TypeReference>
```

```
                <Id name="Real" />
            </TypeReference>
        </Element>
    </TupleType>
</TypeDeclaration>
<TypeDeclaration>
    <Id name="#002" />
    <ArrayType>
        <TupleType>
            <Element>
                <Id name="name" />
                <ListType>
                    <TypeReference>
                        <Id name="Int" />
                    </TypeReference>
                </ListType>
            </Element>
            <Element>
                <Id name="surname" />
                <ListType>
                    <TypeReference>
                        <Id name="Int" />
                    </TypeReference>
                </ListType>
            </Element>
            <Element>
                <Id name="GPA" />
                <TypeReference>
                    <Id name="Real" />
                </TypeReference>
            </Element>
        </TupleType>
    </ArrayType>
</TypeDeclaration>
<VariableDeclaration constant="true" read="false" begin="3:2" end="7:3">
    <Id name="students" begin="3:8" end="3:16" />
    <ArrayType>
        <TupleType>
            <Element>
                <Id name="name" />
                <ListType>
                    <TypeReference>
                        <Id name="Int" />
                    </TypeReference>
                </ListType>
            </Element>
            <Element>
                <Id name="surname" />
                <ListType>
                    <TypeReference>
                        <Id name="Int" />
                    </TypeReference>
                </ListType>
            </Element>
            <Element>
                <Id name="GPA" />
                <TypeReference>
                    <Id name="Real" />
                </TypeReference>
            </Element>
```

```
            </TupleType>
        </ArrayType>
        <Initializer>
            <ArrayLiteral begin="3:33" end="7:3">
                <TupleLiteral begin="4:3" end="4:78">
                    <Element name="name">
                        <ListLiteral begin="4:10" end="4:28">
                            <Literal value="73" begin="4:10" end="4:12" />
                            <Literal value="118" begin="4:15" end="4:18" />
                            <Literal value="97" begin="4:20" end="4:22" />
                            <Literal value="110" begin="4:24" end="4:27" />
                        </ListLiteral>
                    </Element>
                    <Element name="surname">
                        <ListLiteral begin="4:39" end="4:67">
                            <Literal value="73" begin="4:39" end="4:41" />
                            <Literal value="118" begin="4:44" end="4:47" />
                            <Literal value="97" begin="4:49" end="4:51" />
                            <Literal value="110" begin="4:53" end="4:56" />
                            <Literal value="111" begin="4:58" end="4:61" />
                            <Literal value="118" begin="4:63" end="4:66" />
                        </ListLiteral>
                    </Element>
                    <Element name="GPA">
                        <Literal value="4.3" begin="4:74" end="4:77" />
                    </Element>
                </TupleLiteral>
                <TupleLiteral begin="5:3" end="5:98">
                    <Element name="name">
                        <ListLiteral begin="5:10" end="5:38">
                            <Literal value="75" begin="5:10" end="5:12" />
                            <Literal value="115" begin="5:15" end="5:18" />
                            <Literal value="117" begin="5:20" end="5:23" />
                            <Literal value="115" begin="5:25" end="5:28" />
                            <Literal value="104" begin="5:30" end="5:33" />
                            <Literal value="97" begin="5:35" end="5:37" />
                        </ListLiteral>
                    </Element>
                    <Element name="surname">
                        <ListLiteral begin="5:49" end="5:87">
                            <Literal value="83" begin="5:49" end="5:51" />
                            <Literal value="109" begin="5:54" end="5:57" />
                            <Literal value="105" begin="5:59" end="5:62" />
                            <Literal value="114" begin="5:64" end="5:67" />
                            <Literal value="110" begin="5:69" end="5:72" />
                            <Literal value="111" begin="5:74" end="5:77" />
                            <Literal value="118" begin="5:79" end="5:82" />
                            <Literal value="97" begin="5:84" end="5:86" />
                        </ListLiteral>
                    </Element>
                    <Element name="GPA">
                        <Literal value="4.8" begin="5:94" end="5:97" />
                    </Element>
                </TupleLiteral>
                <TupleLiteral begin="6:3" end="6:93">
                    <Element name="name">
                        <ListLiteral begin="6:10" end="6:33">
                            <Literal value="76" begin="6:10" end="6:12" />
                            <Literal value="101" begin="6:15" end="6:18" />
                            <Literal value="115" begin="6:20" end="6:23" />
```

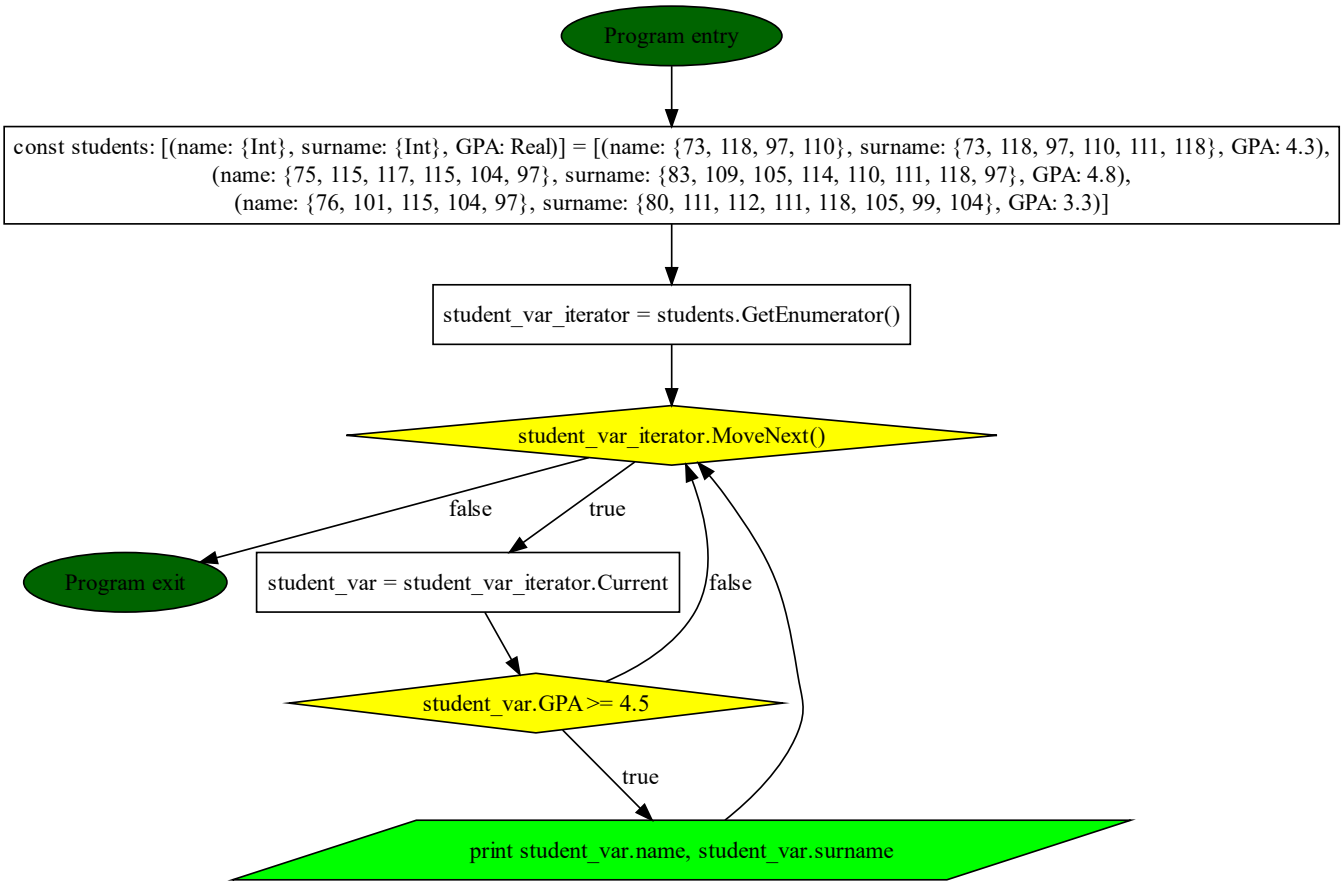
```
                        <Literal value="104" begin="6:25" end="6:28" />
                        <Literal value="97" begin="6:30" end="6:32" />
                    </ListLiteral>
                </Element>
                <Element name="surname">
                    <ListLiteral begin="6:44" end="6:82">
                        <Literal value="80" begin="6:44" end="6:46" />
                        <Literal value="111" begin="6:49" end="6:52" />
                        <Literal value="112" begin="6:54" end="6:57" />
                        <Literal value="111" begin="6:59" end="6:62" />
                        <Literal value="118" begin="6:64" end="6:67" />
                        <Literal value="105" begin="6:69" end="6:72" />
                        <Literal value="99" begin="6:74" end="6:76" />
                        <Literal value="104" begin="6:78" end="6:81" />
                    </ListLiteral>
                </Element>
                <Element name="GPA">
                    <Literal value="3.3" begin="6:89" end="6:92" />
                </Element>
            </TupleLiteral>
        </ArrayLiteral>
    </Initializer>
</VariableDeclaration>
<TypeDeclaration>
    <Id name="#003" />
    <FunctionType>
        <FunctionSignature />
    </FunctionType>
</TypeDeclaration>
<FunctionDeclaration begin="9:2" end="14:5">
    <Id name="main" begin="9:7" end="9:11" />
    <FunctionSignature />
    <Body>
        <ForStatement reverse="false" begin="10:3" end="12:6">
            <Id name="student_var" begin="10:7" end="10:18" />
            <LeftExpression>
                <Reference name="students" type="#002" begin="10:22" end="10:30" />
            </LeftExpression>
            <Guard>
                <Binary begin="10:34" end="10:56">
                    <Operator name="&gt;=" begin="10:50" end="10:52" />
                    <LeftOperand>
                        <TupleElement type="003" begin="10:34" end="10:49">
                            <Primary>
                                <Reference name="student_var" type="student" begin="10:34" end="10:45"
/>

                            </Primary>
                            <Index value="3" />
                        </TupleElement>
                    </LeftOperand>
                    <RightOperand>
                        <Literal value="4.5" begin="10:53" end="10:56" />
                    </RightOperand>
                </Binary>
            </Guard>
            <Body>
                <PrintStatement begin="11:4" end="11:48">
                    <TupleElement type="001" begin="11:10" end="11:26">
                        <Primary>
                            <Reference name="student_var" type="student" begin="11:10" end="11:21" />
```

```
                        </Primary>
                        <Index value="1" />
                    </TupleElement>
                    <TupleElement type="002" begin="11:28" end="11:47">
                        <Primary>
                            <Reference name="student_var" type="student" begin="11:28" end="11:39" />
                        </Primary>
                        <Index value="2" />
                    </TupleElement>
                </PrintStatement>
            </Body>
        </ForStatement>
    </Body>
</FunctionDeclaration>
</Program>
```

DOT CFG данного XML

```
digraph CFG {
        entry [label="Program entry", shape=ellipse, fillcolor=green, style=filled];
        exit [label="Program exit", shape=ellipse, fillcolor=green, style=filled];
        students [label="const students: [(name: {Int}, surname: {Int}, GPA: Real)] = [(name: {73, 118,
97, 110}, surname: {73, 118, 97, 110, 111, 118}, GPA: 4.3), \n(name: {75, 115, 117, 115, 104, 97},
surname: {83, 109, 105, 114, 110, 111, 118, 97}, GPA: 4.8), \n(name: {76, 101, 115, 104, 97}, surname:
{80, 111, 112, 111, 118, 105, 99, 104}, GPA: 3.3)]" fillcolor=white shape=box];
        entry -> students;
        students -> main_ForStatement_id0x90ba0a0;
        main_ForStatement_id0x90ba0a0 [label="student_var_iterator = students.GetEnumerator()" shape=box
style=filled fillcolor=white];
        main_ForStatement_id0x90ba0a0_cond [label="student_var_iterator.MoveNext()" shape=diamond
style=filled fillcolor=yellow];
        main_ForStatement_id0x90ba0a0 -> main_ForStatement_id0x90ba0a0_cond;
        main_ForStatement_id0x90ba0a0_iter [label="student_var = student_var_iterator.Current" shape=box
style=filled fillcolor=white];
        main_ForStatement_id0x90ba0a0_guard [label="student_var.GPA >= 4.5" shape=diamond style=filled
fillcolor=yellow];
        main_ForStatement_id0x90ba0a0_cond -> main_ForStatement_id0x90ba0a0_iter [label="true"];
        main_ForStatement_id0x90ba0a0_iter -> main_ForStatement_id0x90ba0a0_guard;
        main_ForStatement_id0x90ba0a0_cond -> exit [label="false"];
        main_ForStatement_id0x90ba0a0_guard_PrintStatement_id0x910b4c0 [label="print student_var.name,
student_var.surname" shape=parallelogram fillcolor=lightgreen style=filled];
        main_ForStatement_id0x90ba0a0_guard ->
main_ForStatement_id0x90ba0a0_guard_PrintStatement_id0x910b4c0 [label="true"];
        main_ForStatement_id0x90ba0a0_guard_PrintStatement_id0x910b4c0 ->
main_ForStatement_id0x90ba0a0_cond;
        main_ForStatement_id0x90ba0a0_guard -> main_ForStatement_id0x90ba0a0_cond [label="false"];
}
```

Визуализированный CFG



: