Семантика

Лекция №10

Цель

Обрабатывать все правила семантики до начала выполнения кода.

Синтаксис и семантика языков программирования

Синтаксис

Способ разбора лексем в «предложения» языка

1. Правила КС-грамматики
2. Приоритет и ассоциативность операторов

Семантика

???

Синтаксис и семантика языков программирования

Синтаксис

Способ разбора лексем в «предложения» языка

1. Правила КС-грамматики
2. Приоритет и ассоциативность операторов

Семантика

Смысл и поведение языковых конструкций.

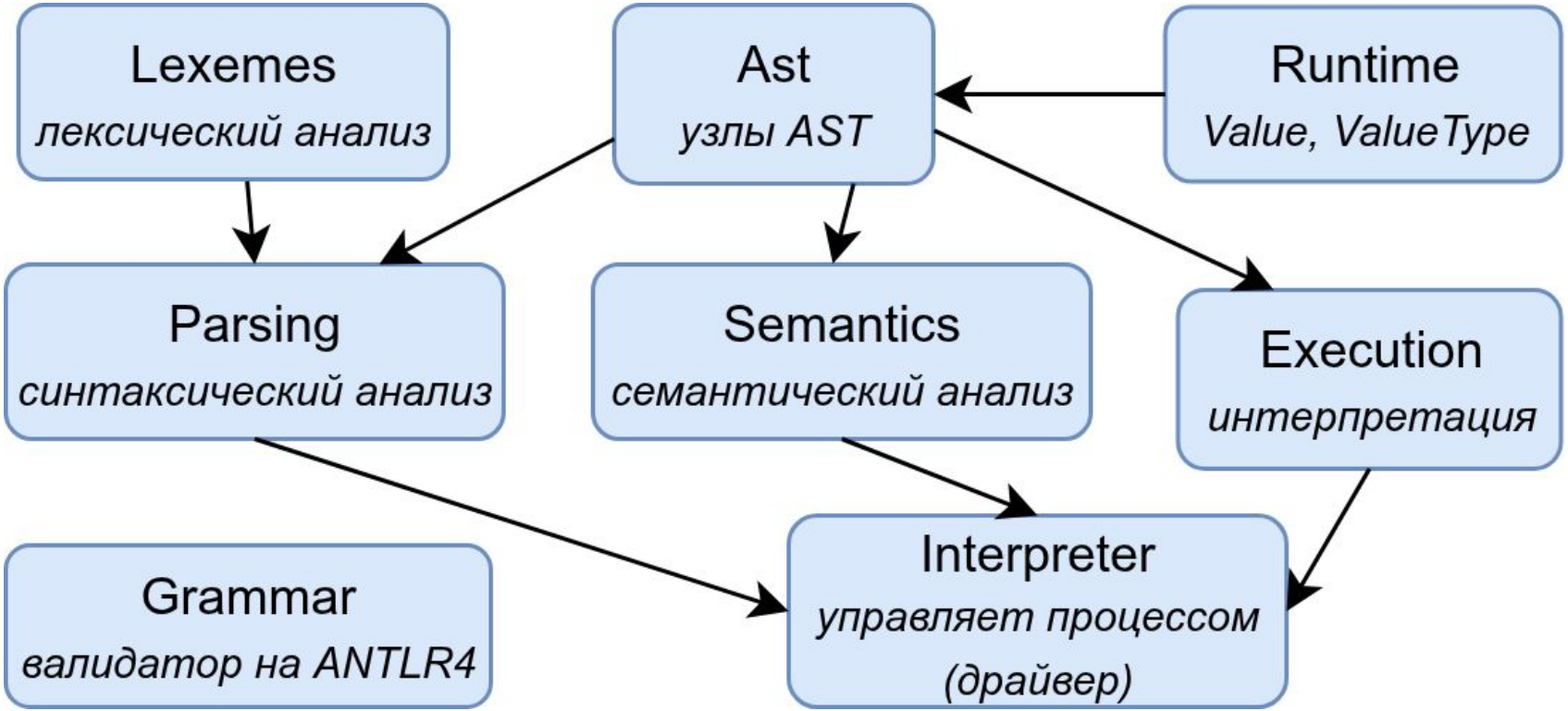
1. Правила работы с типами данных
2. Способ отображения имён на узлы AST
   - англ. name resolution / symbol linking
3. Контекстно-зависимые правила
   - Запрет break и continue вне циклов

Интерпретатор языка Tiger

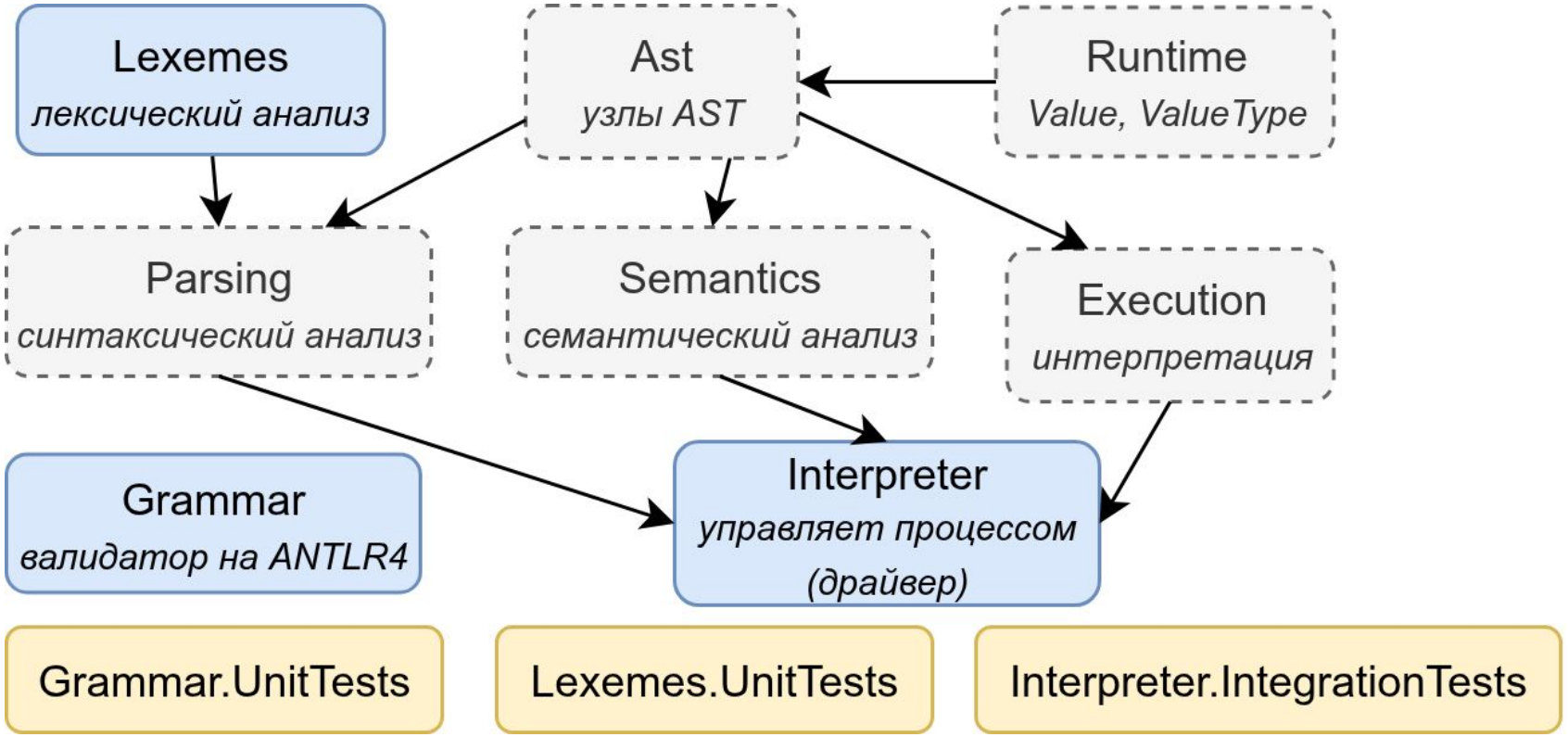
https://sourcecraft.dev/sshambir-public/pstiger

Ветка	Что добавляет
05_if_else	Ветвления
06_functions	Пользовательские функции и процедуры
07_loops	Циклы for и while, выражение break

Модули проекта PsTiger

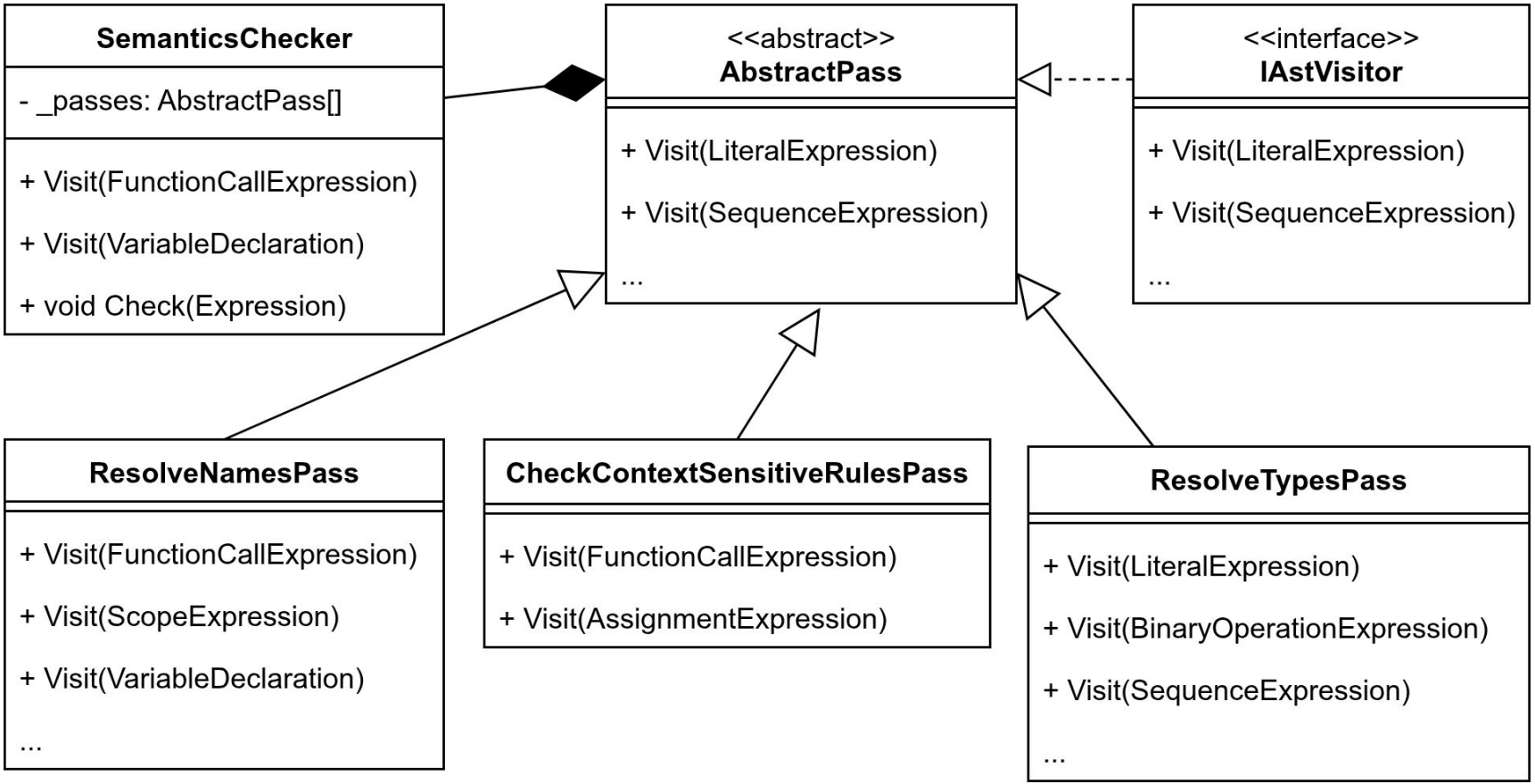


Тесты проекта PsTiger



Проверка семантики

Обработка в несколько проходов



Обработка в несколько проходов

```
_passes = [
    new ResolveNamesPass(globalSymbols),
    new CheckContextSensitiveRulesPass(),
    new ResolveTypesPass(),
];
public void Check(Expression program) {
    foreach (AbstractPass pass in _passes) {
        program.Accept(pass);
    }
}
```

Атрибутные грамматики

Формализация правил семантики

Правила грамматики:

$E \rightarrow T \qquad E \rightarrow E + T \qquad E \rightarrow E - T$

Формализация правил семантики

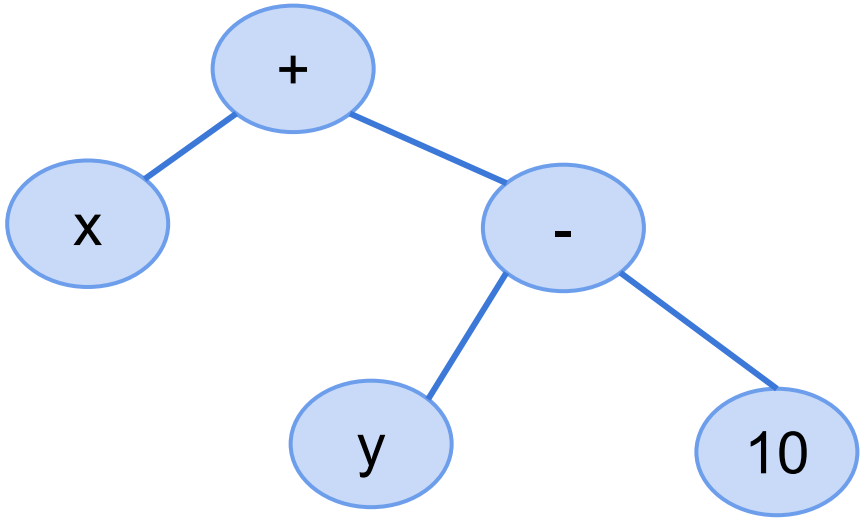
Правила грамматики:

```
E → T     E → E + T     E → E - T
```

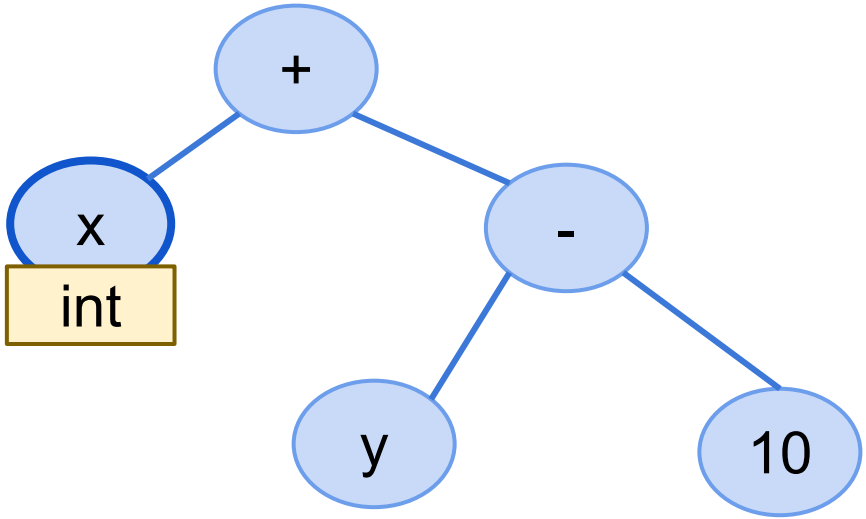
Правило семантики:

```
type(left + right) =
    if type(left) == int && type(right) == int
    then int
    else error
```

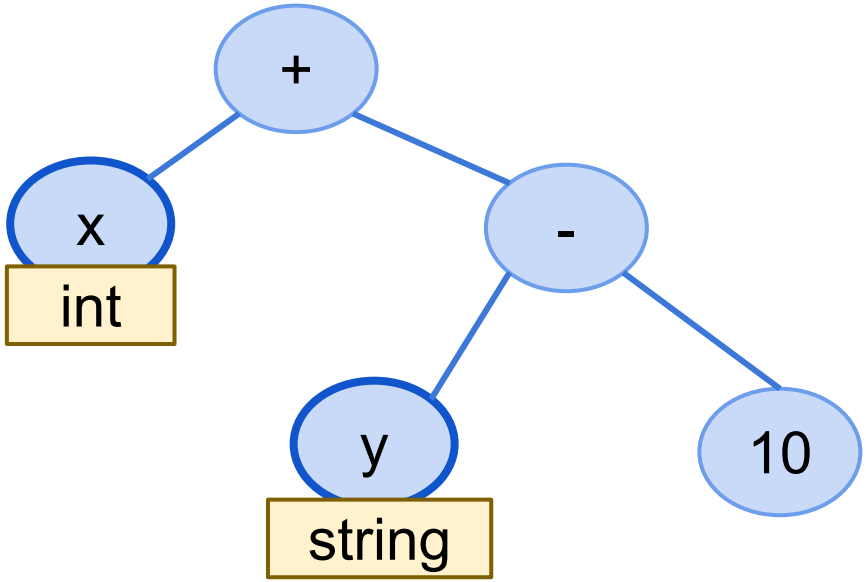
Вычисление атрибута “type”



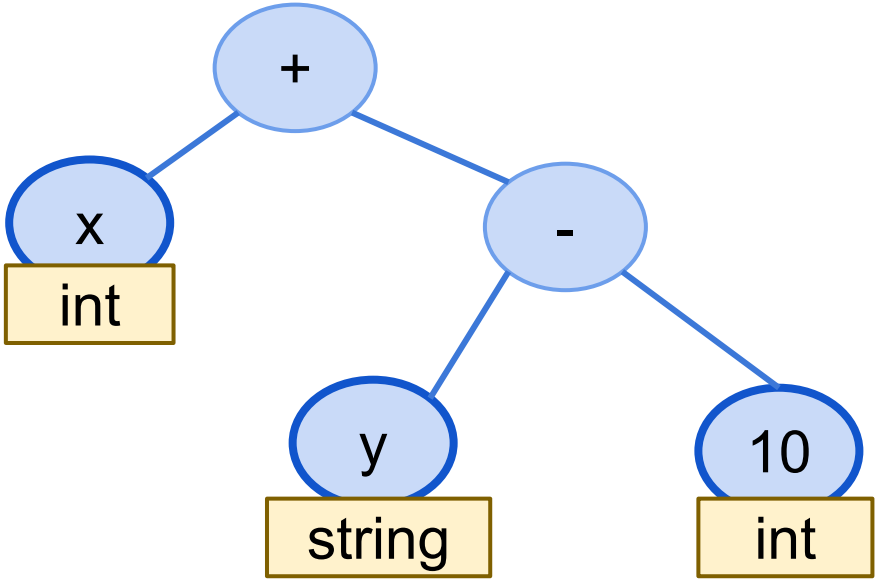
Вычисление атрибута “type”



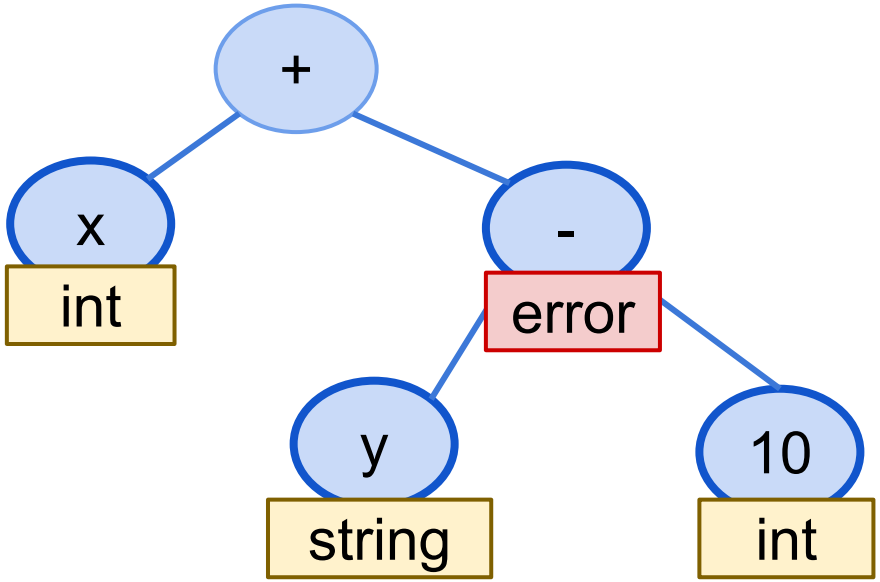
Вычисление атрибута “type”



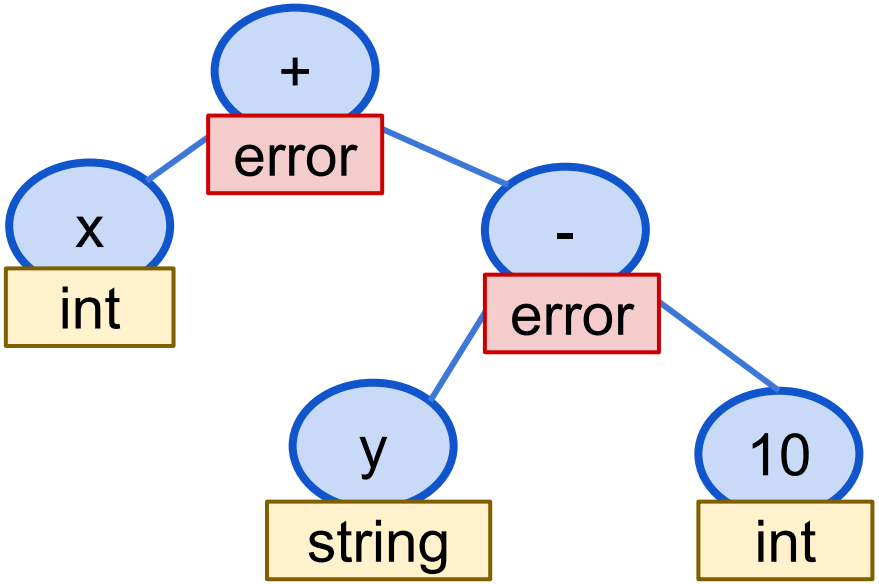
Вычисление атрибута “type”



Вычисление атрибута “type”



Вычисление атрибута “type”

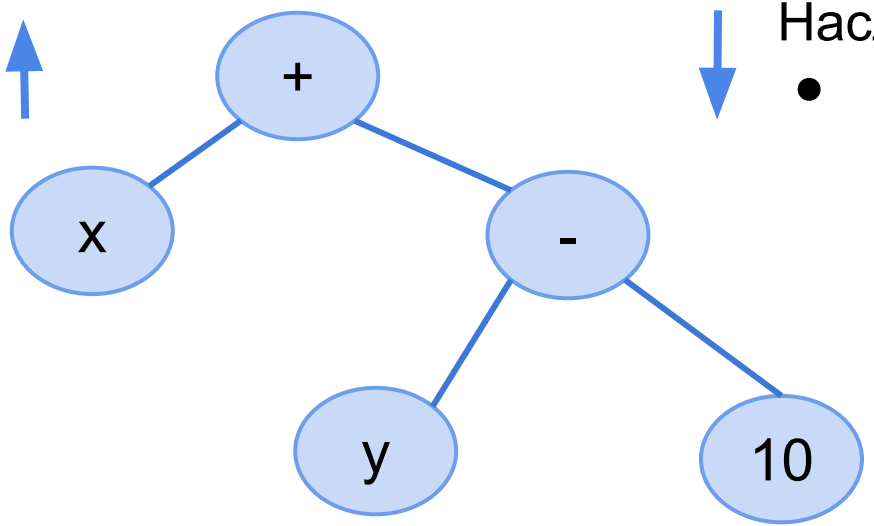


Виды атрибутов

1. Синтезируемые — от терминалов к начальному символу
2. Наследуемые — от начального символа к терминалам

Синтезируемые:

- type
- value

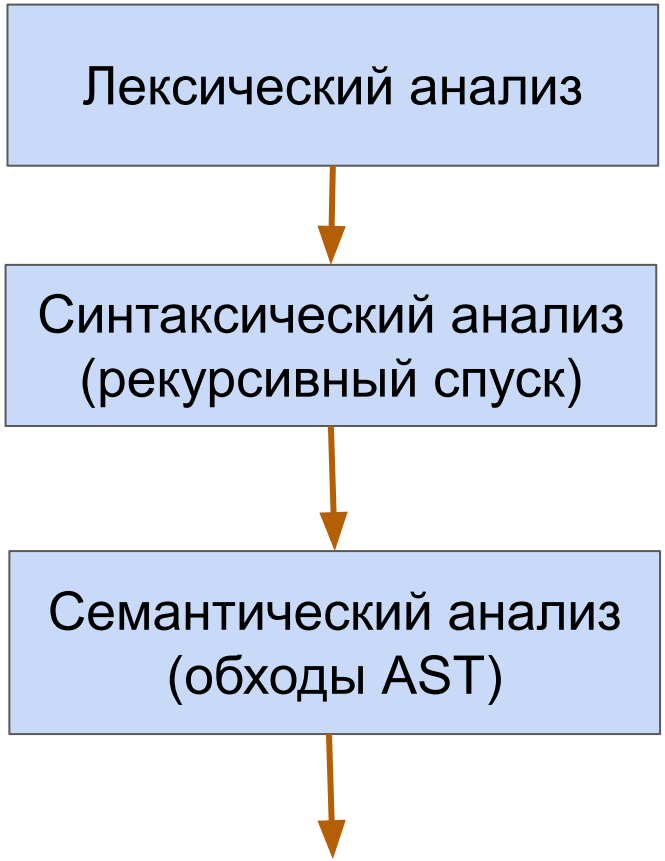


Наследуемые:

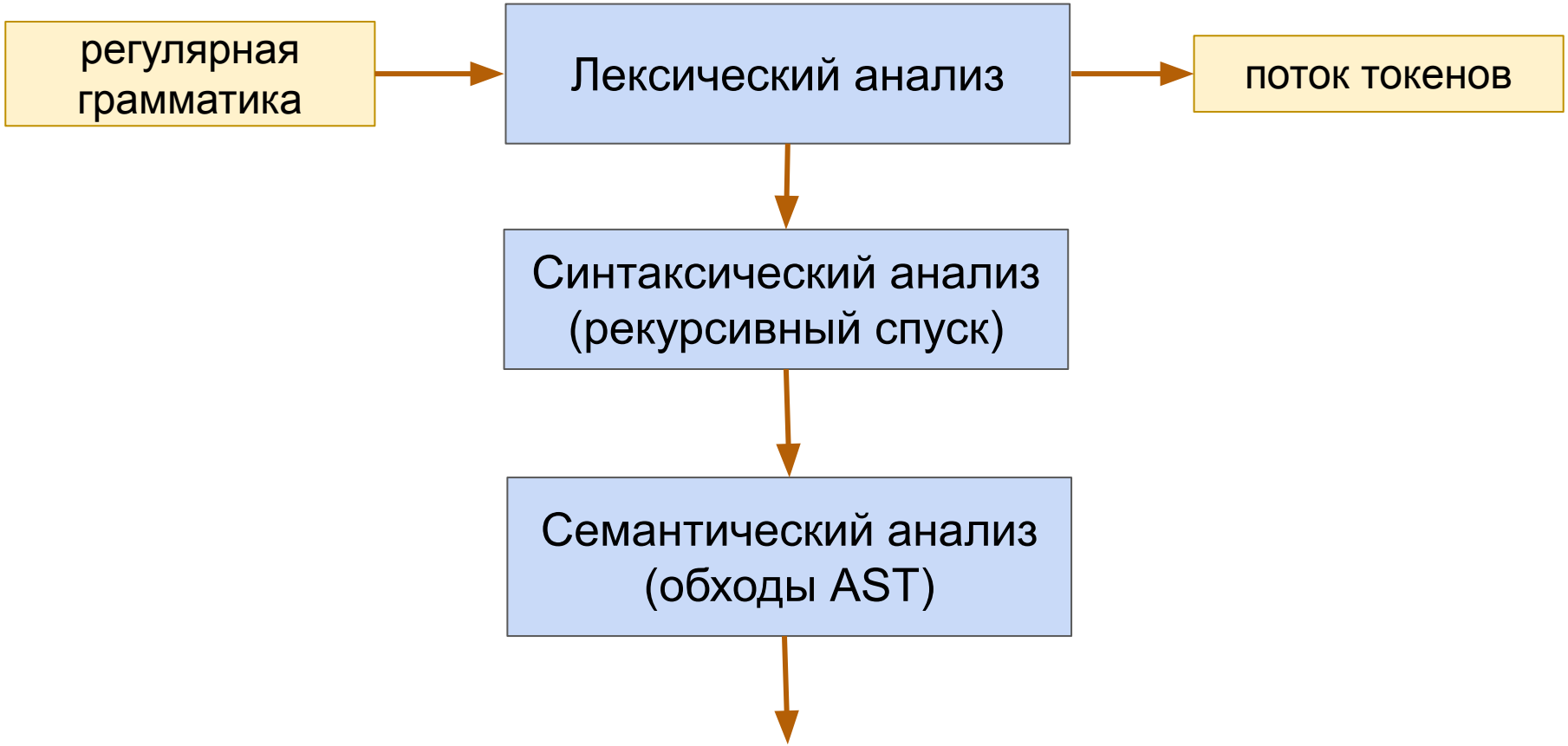
- scope

Атрибуты в AST

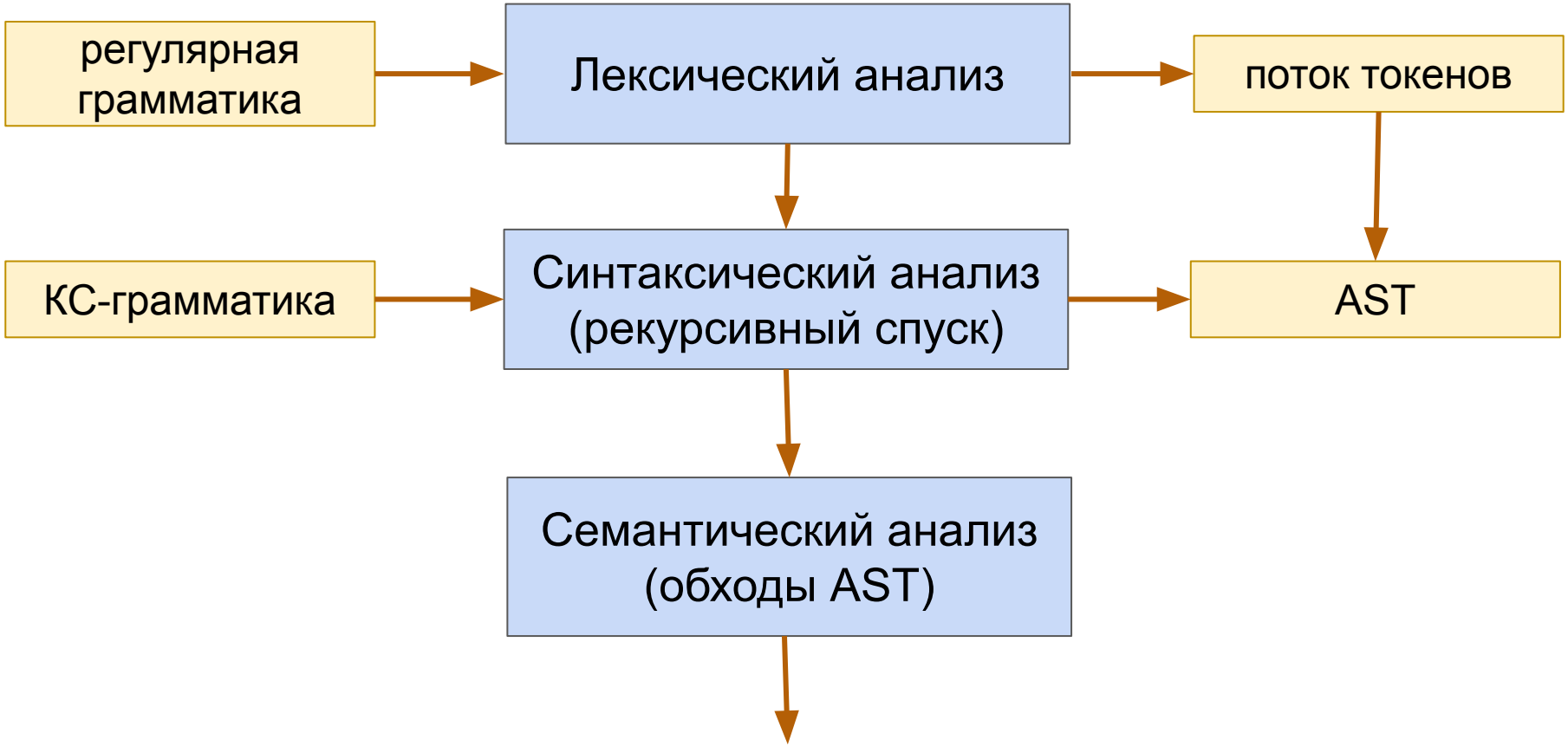
Атрибуты в AST



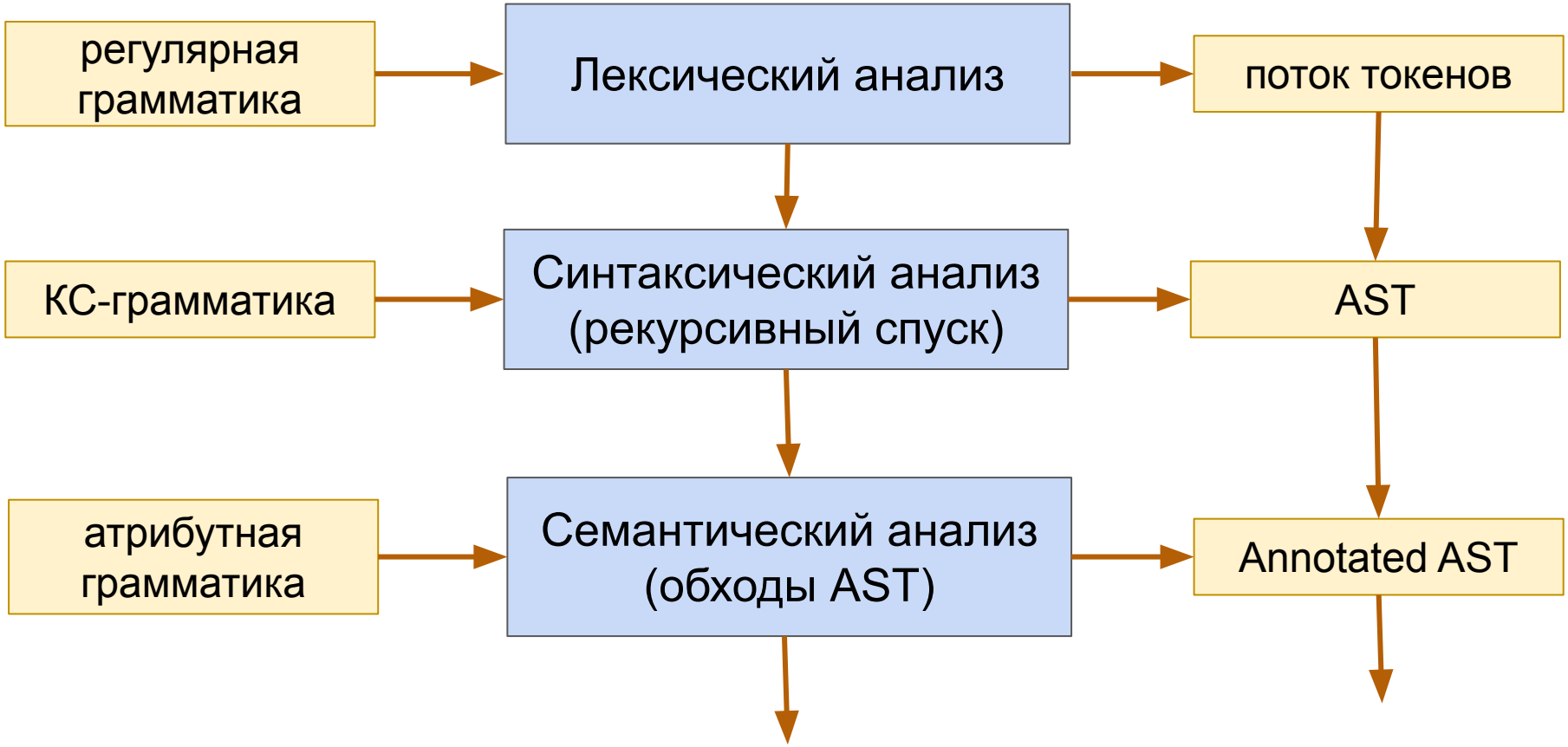
Атрибуты в AST



Атрибуты в AST



Атрибуты в AST



Способы хранения атрибута

1. Поля в узлах в дерева
    - `class Expression { Type type; }`
2. Отдельная таблица
    - `Dictionary<Expression, Type> types;`
3. Локальные переменные в посетителе
    - `class AstEvaluator { Stack<Value> _values; }`

Класс AstAttribute<T>

1. Set() можно вызвать один раз
2. после Set() можно вызывать Get()

```
public struct AstAttribute<T> {
    private T _value;
    private bool _initialized;
    public T Get() { ... }
    public void Set(T value) { ... }
}
```

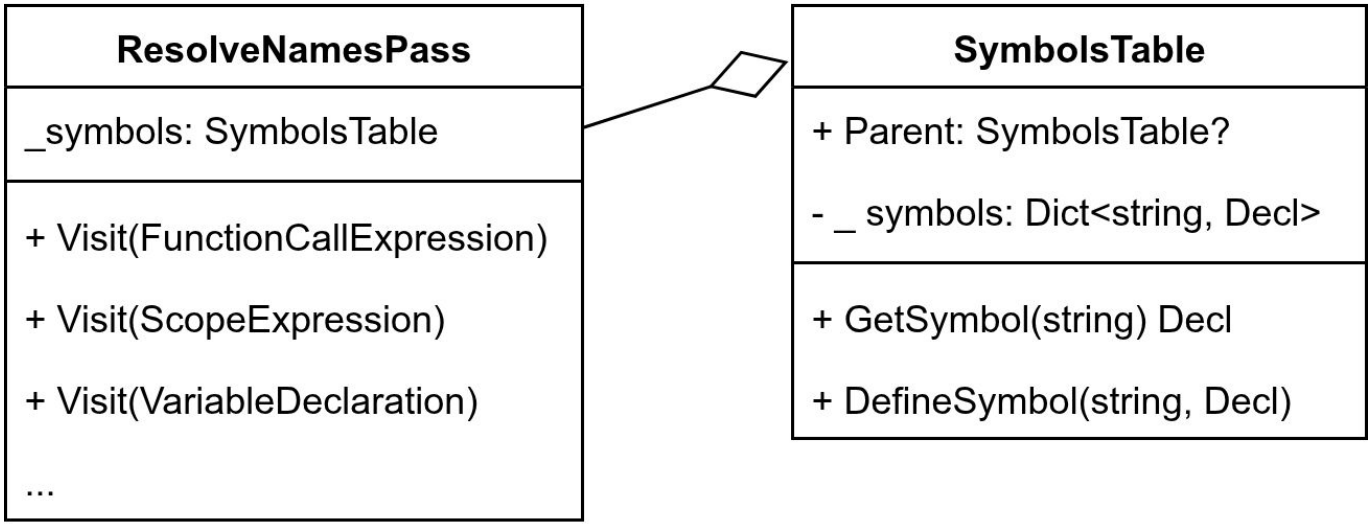
Класс ReferenceEqualityComparer<T>

Позволяет использовать тип T как ключ в словаре.

```
public class ReferenceEqualityComparer<T> : IEqualityComparer<T> {
    public bool Equals(T? x, T? y) {
        return ReferenceEquals(x, y);
    }
    public int GetHashCode(T obj) {
        return RuntimeHelpers.GetHashCode(obj);
    }
}
```

Таблица символов

Класс таблицы символов



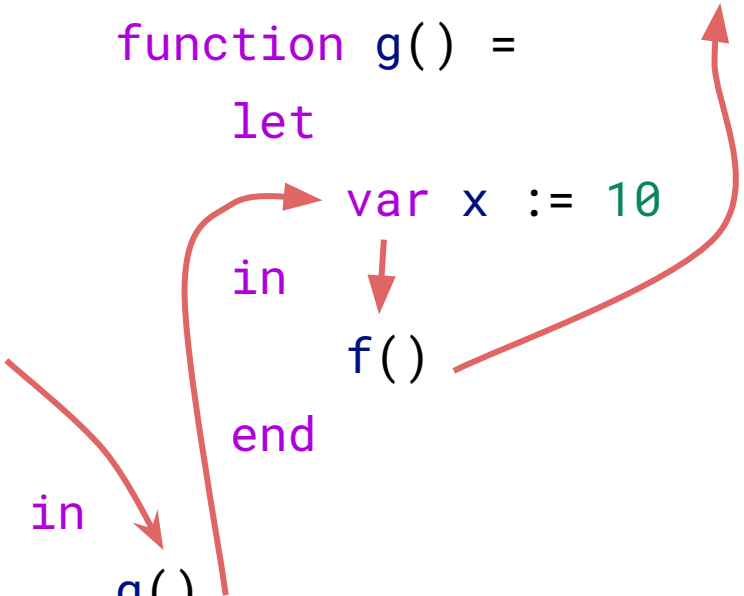
Области видимости

Динамическая область видимости

```
let
    function f() = print(x)
    function g() =
        let
            var x := 10
        in
            f()
        end
in
    g()
end
```

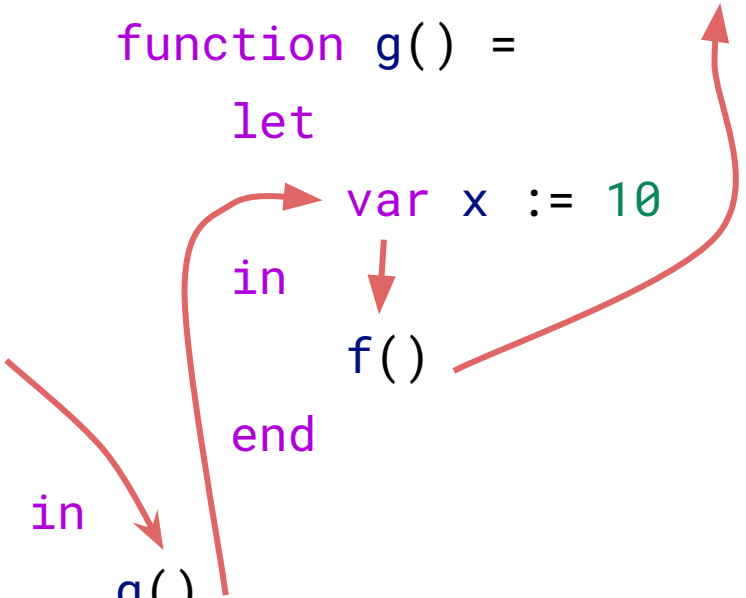
Динамическая область видимости

```
let
    function f() = print(x)
    function g() =
        let
            var x := 10
        in
            f()
        end
in
    g()
end
```



Динамическая область видимости

```
let
    function f() = print(x)
    function g() =
        let
            var x := 10
        in
            f()
        end
in
    g()
end
```



x = 10

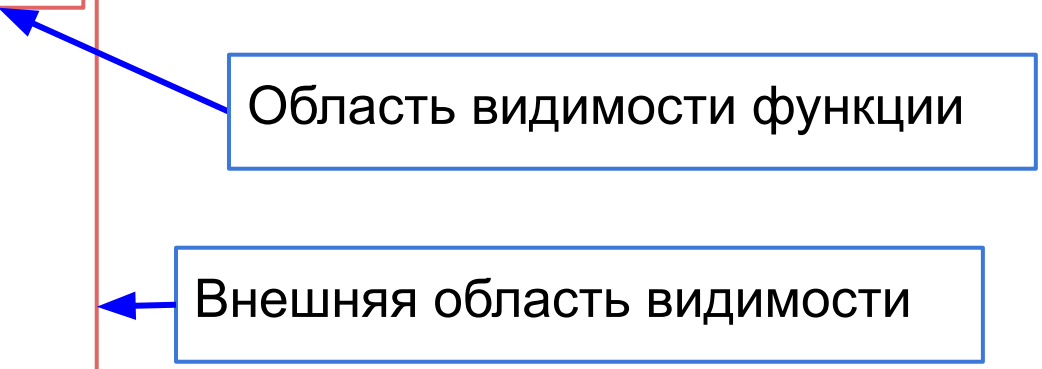
Набор переменных зависит от порядка вызова функций

Лексическая область видимости

```
let
    function f() = print(x)
    function g() =
        let
            var x := 10
        in
            f()
        end
in
    g()
end
```

Лексическая область видимости

```
let
    function f() = print(x)
    function g() =
        let
            var x := 10
        in
            f()
        end
in
    g()
end
```



Лексическая область видимости

```
let
    function f() = print(x)
    function g() =
        let
            var x := 10
        in
            f()
        end
in
    g()
end
```

No variable with name "x"

Область видимости функции

Внешняя область видимости

Лексическая область видимости

```
let
    function f() = print(x)
    function g() =
        let
            var x := 10
        in
            f()
        end
in
    g()
end
```

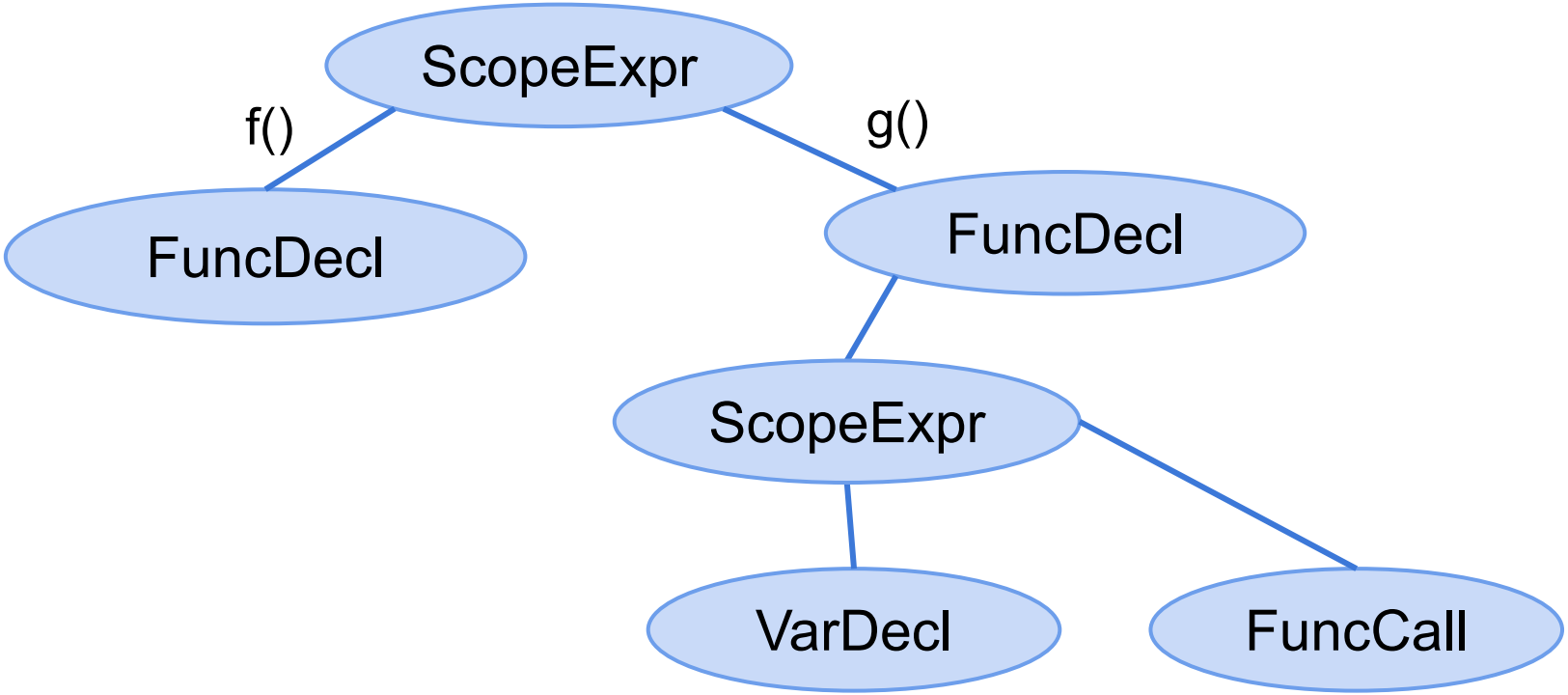
No variable with name "x"

Область видимости функции

Внешняя область видимости

Набор переменных зависит от расположения функции в коде

Вычисление атрибута “scope”



Типы данных в интерпретаторе Tiger

Язык Tiger

- Учебный язык программирования
- Разработан Andrew Appel для книг и курсов в Princeton University

```
let
    var a : int := 0
in
    for i := 0 to 100 do (
        a := a + 1;
        ()
    )
end
```

Типы данных в Tiger

Типы данных:

1. Строки — string
2. Целые числа — int
3. Структуры и массивы

Типы данных в Tiger

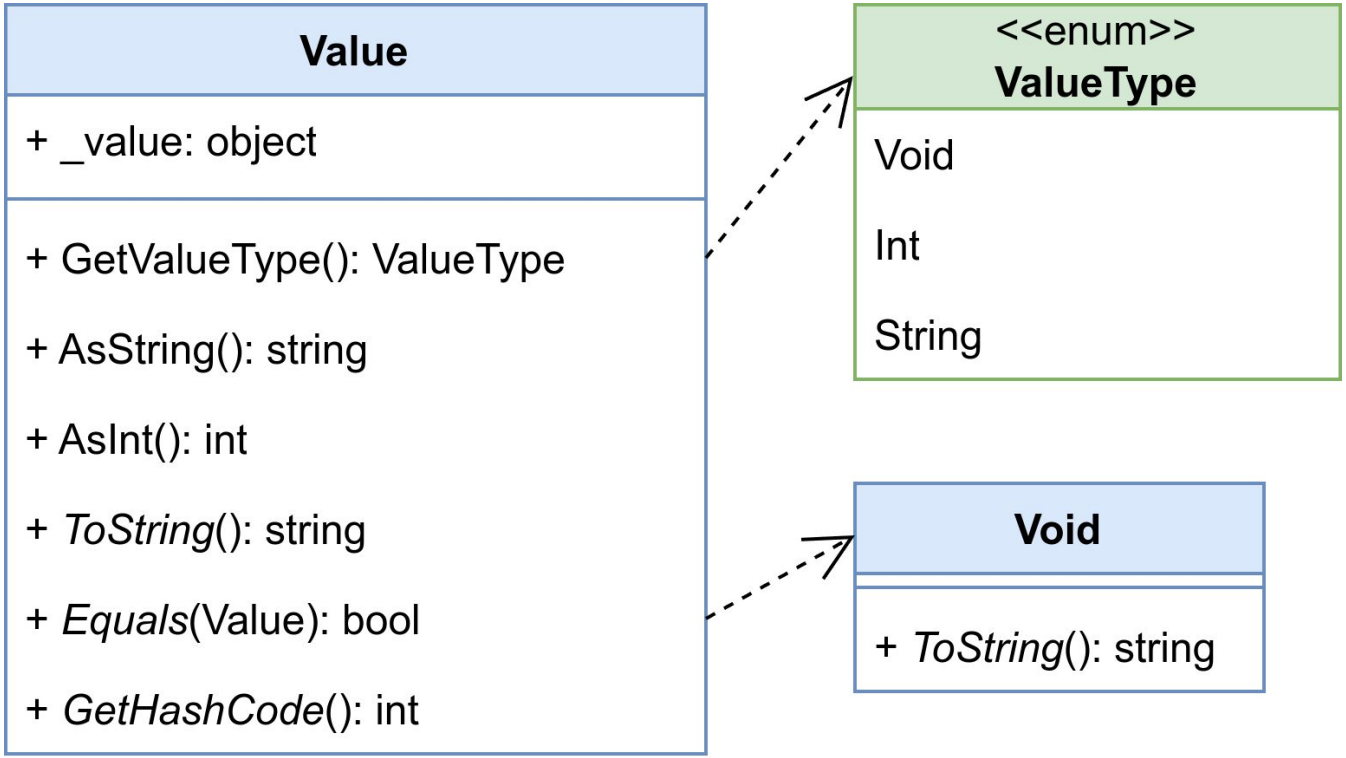
Типы данных:

1. Строки — string
2. Целые числа — int
3. Структуры и массивы

Особенности:

1. Выражения и функции могут ничего не возвращать
2. Нет приведений типов — как явных, так и неявных

Класс Value


Value
+ _value: object
+ GetValueType(): ValueType
+ AsString(): string
+ AsInt(): int
+ ToString(): string
+ Equals(Value): bool
+ GetHashCode(): int
<<enum>>
ValueType
Void
Int
String
Void
+ ToString(): string

Сопоставление по шаблону в C#

```
public string AsString() {
    return _value switch {
        string s => s,
        _ => throw new InvalidOperationException($"Value
{_value} is not a string"),
    };
}
```

Сопоставление по шаблону в C#

```
public ValueType GetValueType() {
    return _value switch  {
        string => ValueType.String,
        int => ValueType.Int,
        Void => ValueType.Void,
        _ => throw new InvalidOperationException($"Unexpected
value {_value} of type {_value.GetType()}"),
    };
}
```

Сложность вычислений при динамической типизации

В языке Tiger:

1. Операторы `+, -, *, /, &, |` работают с числами
2. Операторы `<, >, <=, >=` работают с числами и строками
3. Операторы `=, <>` работают с числами, строками и ссылочными типами

Проблемы:

1. Бинарные операции — двойная диспетчеризация по типам
2. Много проверок на типы

Решение №1: разделить проверки и вычисления

Сделаем две реализации IAstVisitor:

1. `AstTypeChecker` — проверяет соответствие типов
    - не вычисляет результаты
    - относится к семантическим проверкам
2. `AstEvaluator` — выполняет интерпретацию по AST
    - предполагает, что типы уже проверены

Решение №2: ситуативный рефакторинг

```
public void Visit(BinaryOperationExpression e) {
    e.Left.Accept(this);
    e.Right.Accept(this);
    Value right = _values.Pop(), left = _values.Pop();
    _values.Push(EvaluationUtil.ApplyBinaryOperation(
        e.Operation, left, right
    ));
}
```

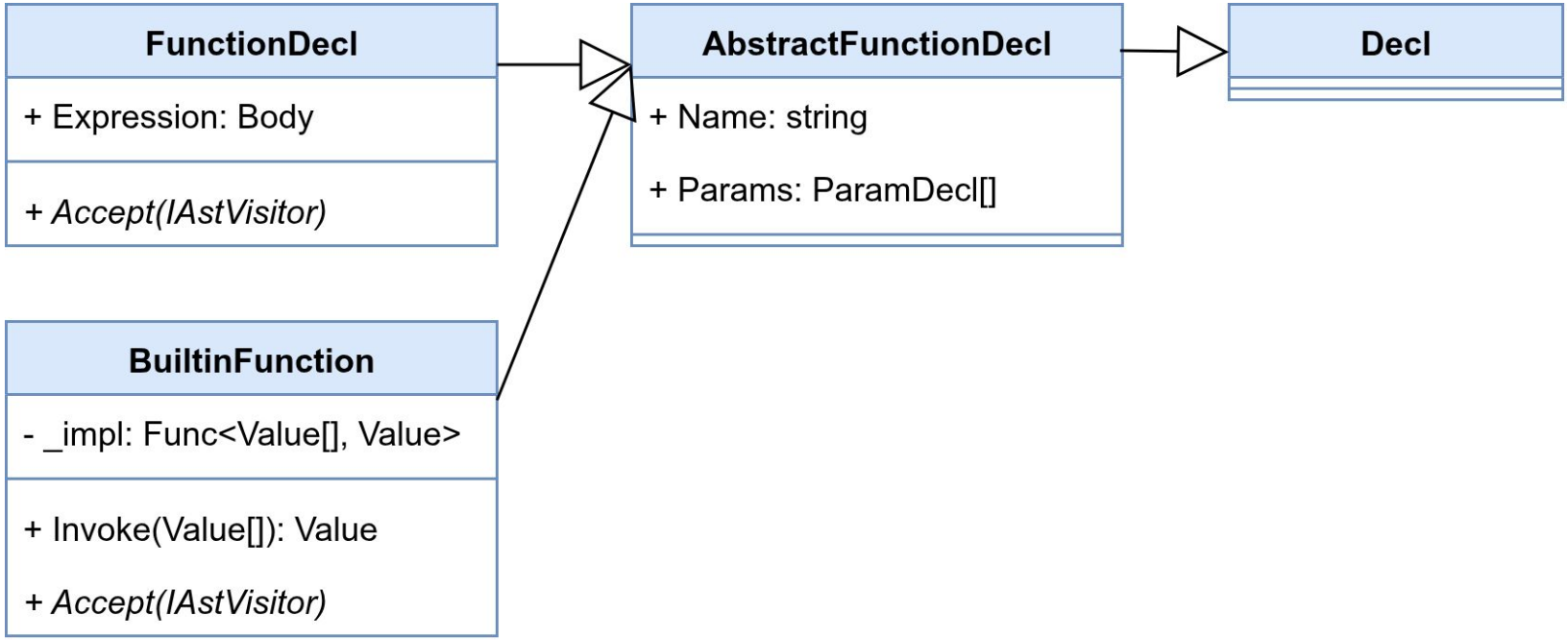
Решение №2: ситуативный рефакторинг

```
public void Visit(BinaryOperationExpression e) {
    e.Left.Accept(this);
    e.Right.Accept(this);
    Value right = _values.Pop(), left = _values.Pop();
    _values.Push(EvaluationUtil.ApplyBinaryOperation(
        e.Operation, left, right
    ));
}
```

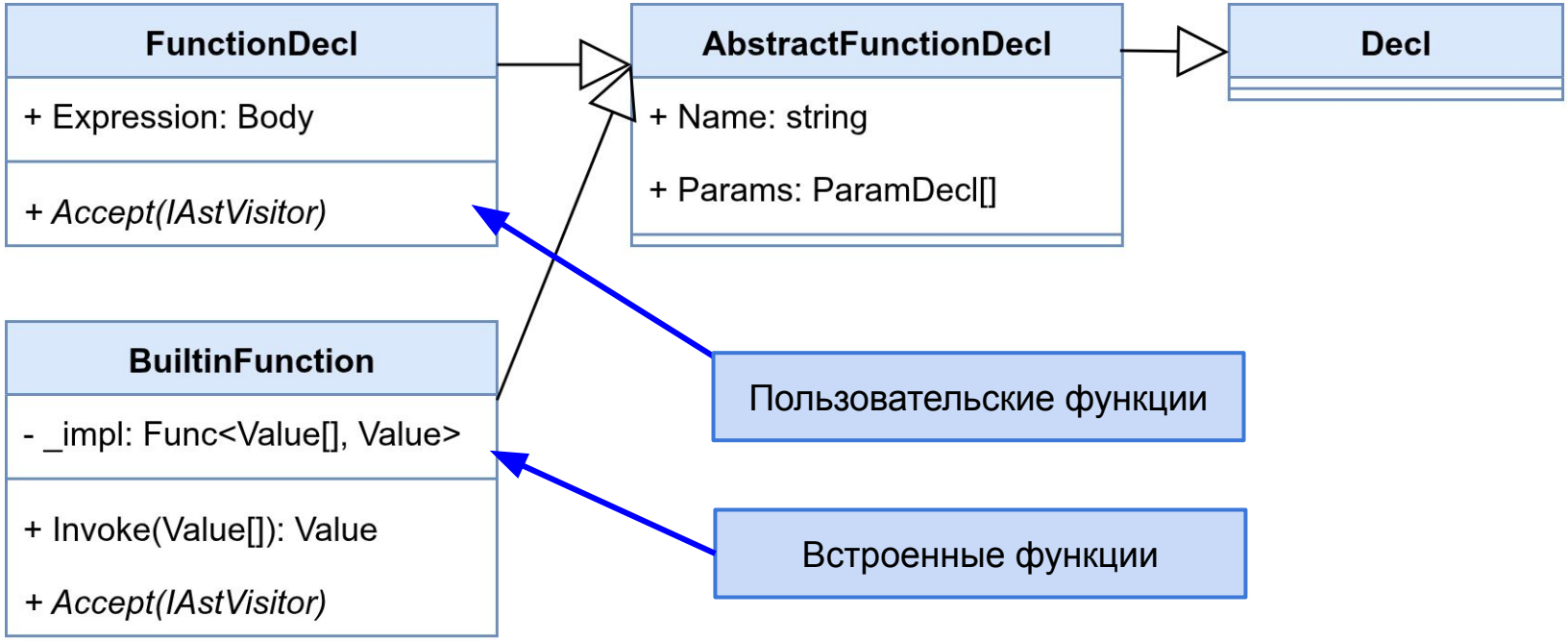
switch-case внутри вызываемого метода

Поддержка рекурсии

AST для функций



AST для функций

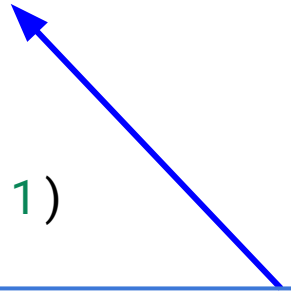


Рекурсия

```
let
    function factorial(x: int): int =
        if x <= 1
        then 1
        else x * factorial(x - 1)
in
    printi(factorial(5))
end
```

Рекурсия

```
let
    function factorial(x: int): int =
        if x <= 1
        then 1
        else x * factorial(x - 1)
in
    printi(factorial(5))
end
```



Специализированный обход AST:

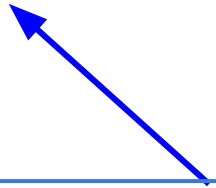
1. Добавить функцию в SymbolsTable
2. Обойти параметры и тело функции

Взаимная рекурсия

```
let
    function f(n: int) = g(n + 1)
    function g(n: int) = (
        printi(n);
        if n < 10 then
            f(n + 1)
    )
in
    f(1)
end
```

Взаимная рекурсия

```
let
    function f(n: int) = g(n + 1)
    function g(n: int) = (
        printi(n);
        if n < 10 then
            f(n + 1)
    )
in
    f(1)
end
```



Специализированный обход AST:

1. Добавить непрерывный ряд функций в SymbolsTable
2. Обойти параметры и тело каждой функции

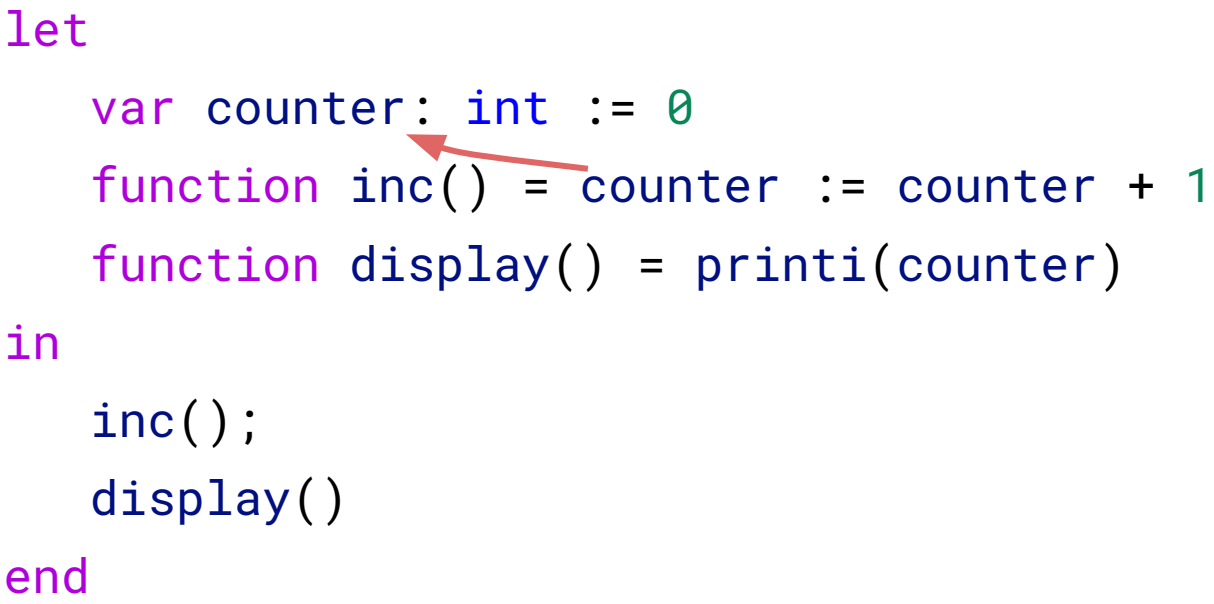
Поддержка замыканий

Замыкание (Closure)

Замыкание — функция, которая «захватывает» переменные внешней лексической области видимости.

Замыкание (Closure)

Замыкание — функция, которая «захватывает» переменные внешней лексической области видимости.



Замыкание (Closure)

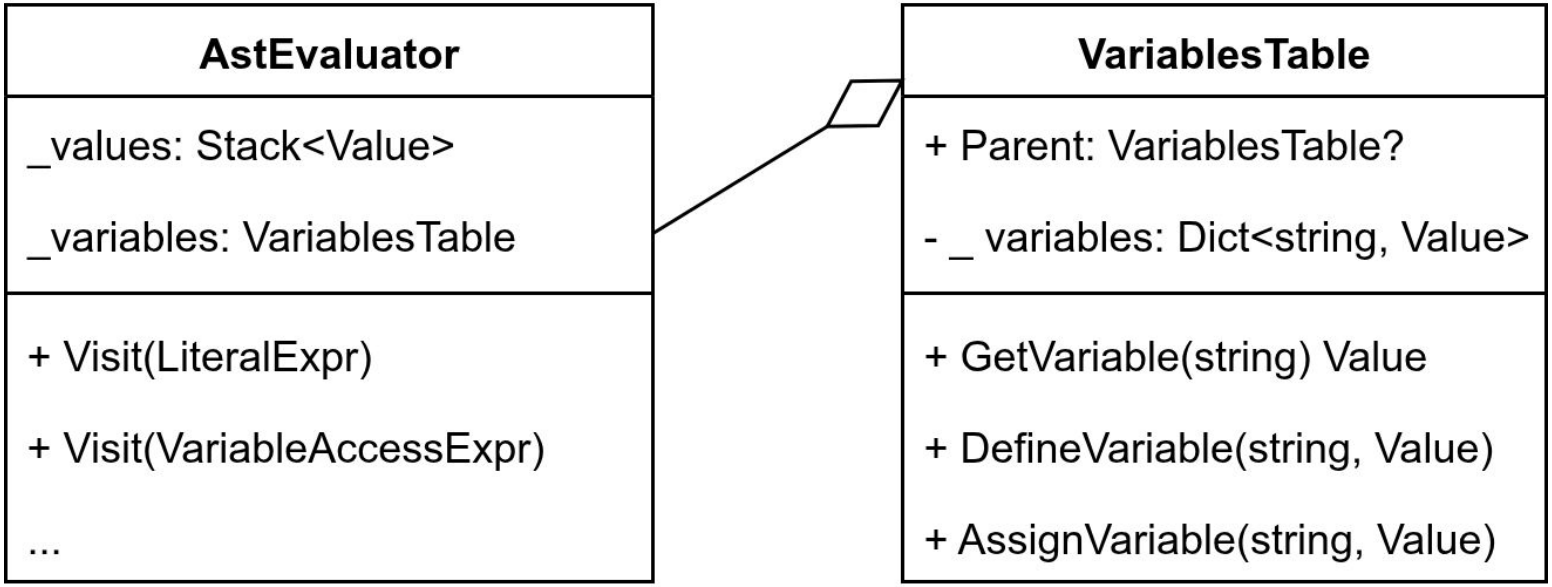
Замыкание — функция, которая «захватывает» переменные внешней лексической области видимости.

```
let
    var counter: int := 0
    function inc() = counter := counter + 1
    function display() = printi(counter)
in
    inc();
    display()
end
```

Шаг 1. Захват VariablesTable

Шаг 2. Создание VariablesTable, дочернего от захваченного

Таблица переменных



Замена таблицы переменных при вызове

```
private void InvokeFunction(FuncCallExpr e, FuncDecl fn) {
    VariablesTable captured = _context.GetCapturedVariables(fn);
    VariablesTable oldVariables = _variables;
    _variables = new VariablesTable(parent: captured);
    // ...вычисляем аргументы, записываем в таблицу переменных.
    // ...вызываем функцию и сохраняем результат в стеке.
    // ...восстанавливаем старую таблицу переменных
}
```

Подытожим

Вопросы к спецификации языка

Для функций:

1. Области видимости — лексические или динамические?
2. Есть ли рекурсия?
3. Есть ли взаимная рекурсия?
4. Есть ли замыкания (вложенные функции)?

Интерпретатор языка Tiger

https://sourcecraft.dev/sshambir-public/pstiger

Ветка	Что добавляет
05_if_else	Ветвления
06_functions	Пользовательские функции и процедуры
07_loops	Циклы for и while, выражение break

Спасибо за внимание!
Вопросы?