Атрибутные грамматики

Лекция №11

Цель

Обрабатывать все правила семантики до начала выполнения кода.

Цель

Обрабатывать все правила семантики до начала выполнения кода.

Дополнительная цель: исправить неточности прошлых лекций

1. Таблица символов и таблица переменных
2. Обработка областей видимости

Интерпретатор языка Tiger

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

Ветка	Что добавляет
01_grammar_checker	Валидатор синтаксиса на ANTLR4
02_lexical_analysis	Лексический анализ Tiger
03_expressions	Разбор выражений
04_variables	Переменные и присваивания
05_if_else	Ветвления
06_functions	Пользовательские функции и процедуры

Атрибутные грамматики

Формализация правил семантики

Правила грамматики:

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

Формализация правил семантики

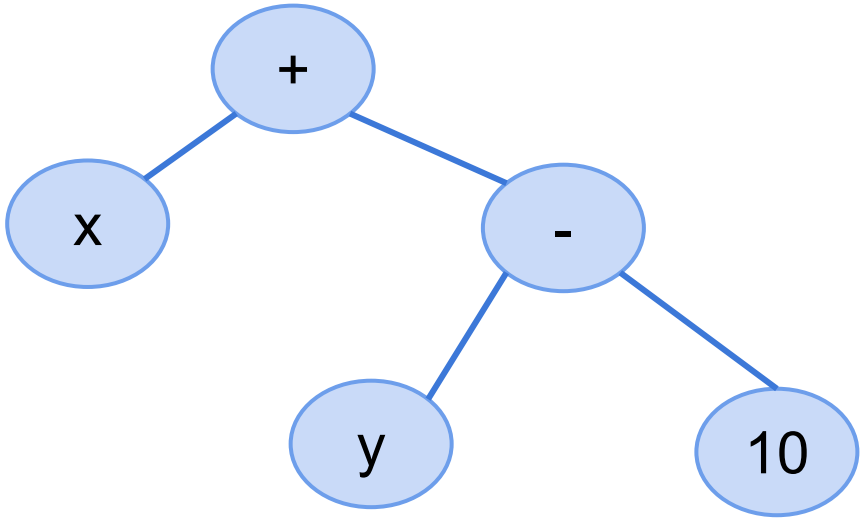
Правила грамматики:

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

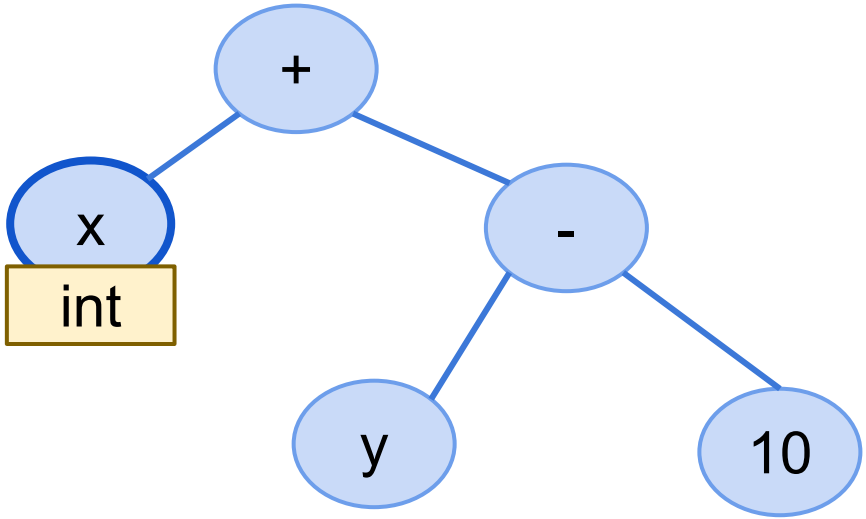
Правило семантики:

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

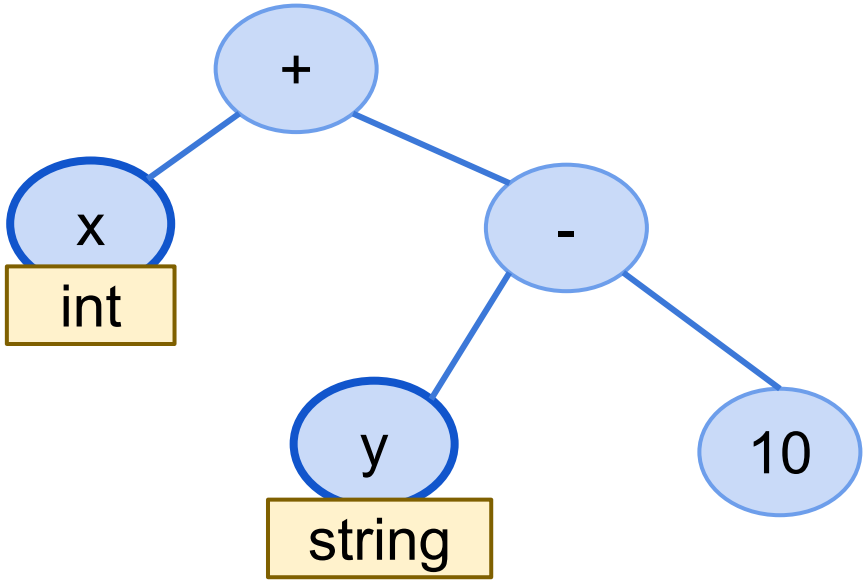
Вычисление атрибута “type”



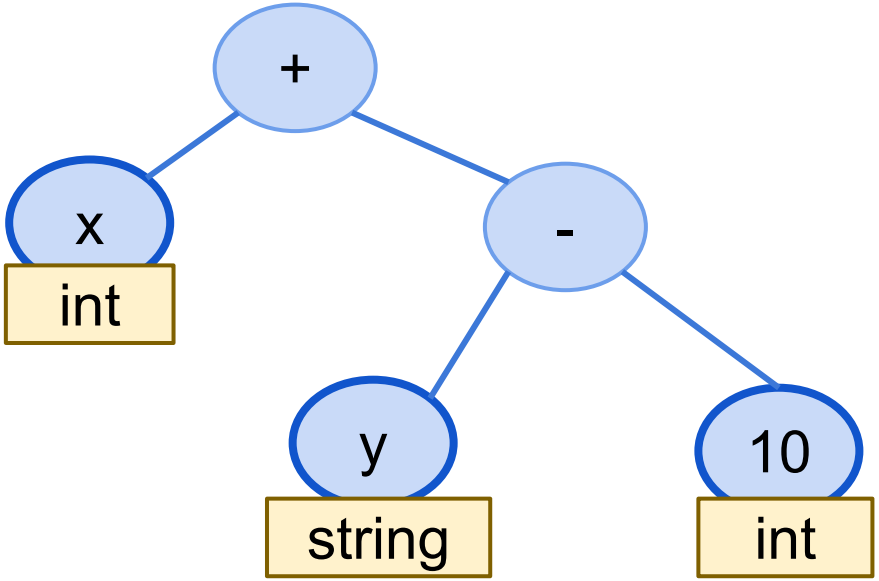
Вычисление атрибута “type”



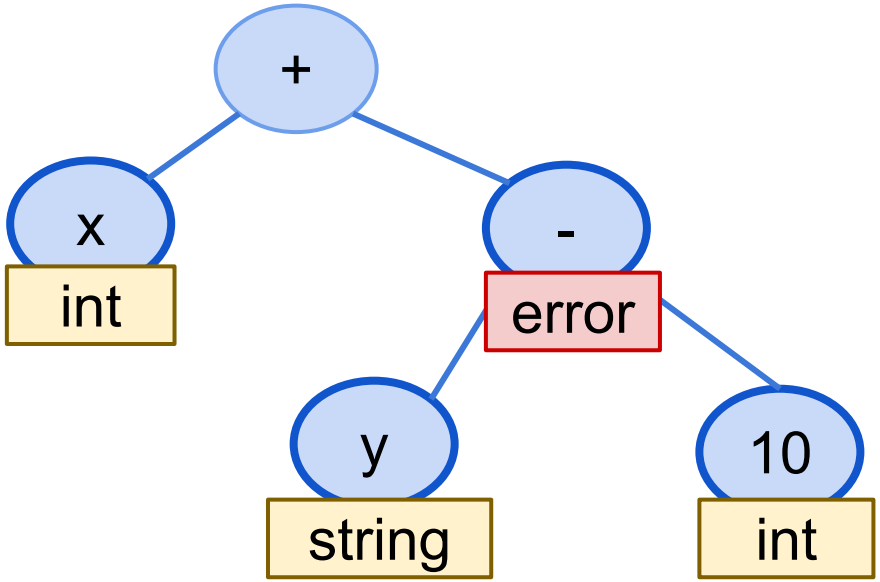
Вычисление атрибута “type”



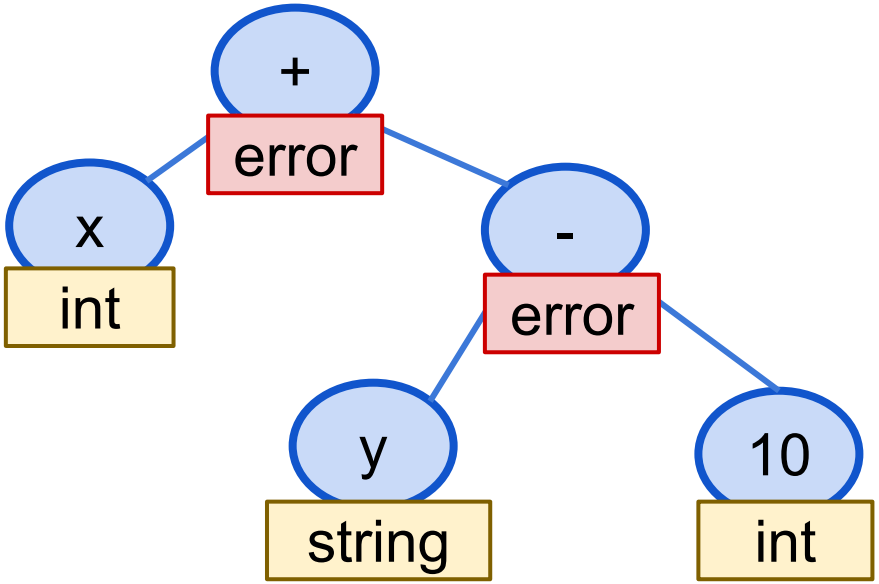
Вычисление атрибута “type”



Вычисление атрибута “type”



Вычисление атрибута “type”

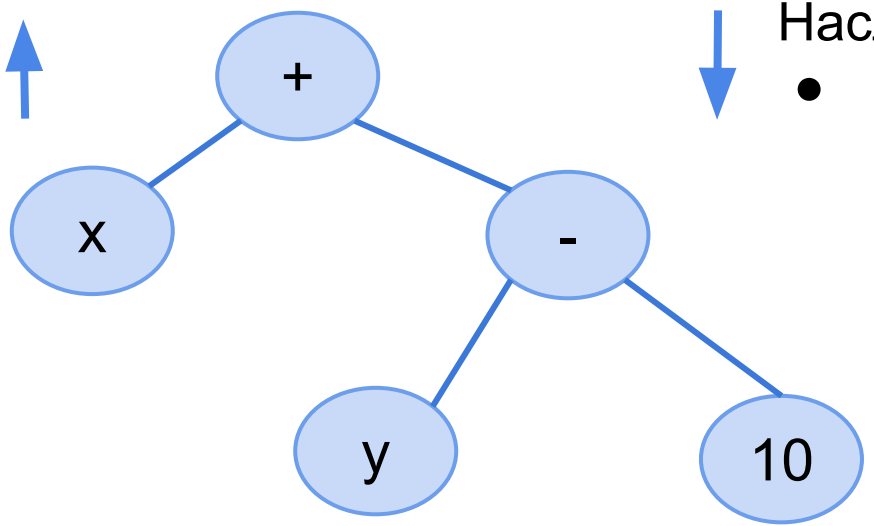


Виды атрибутов

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

Синтезируемые:

- type
- value

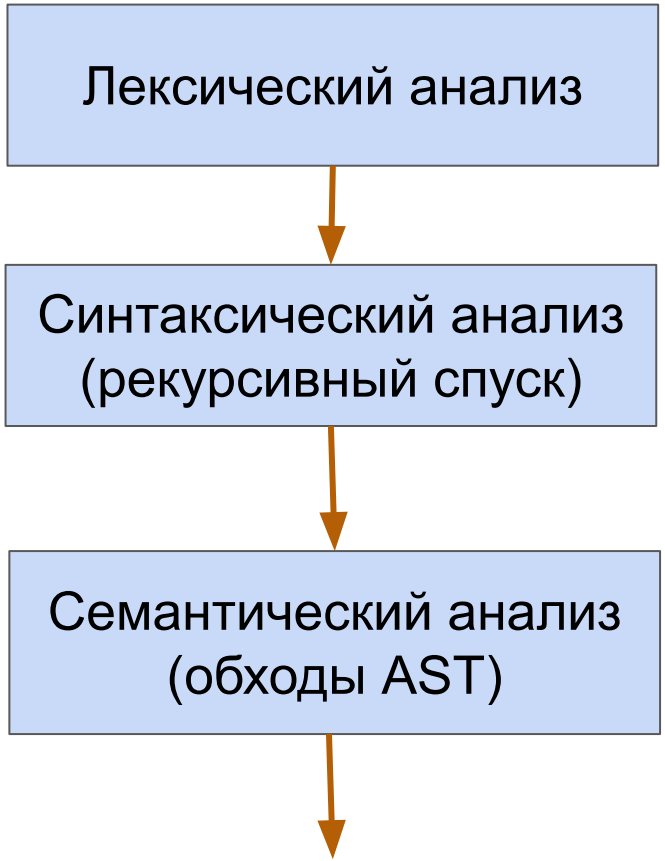


Наследуемые:

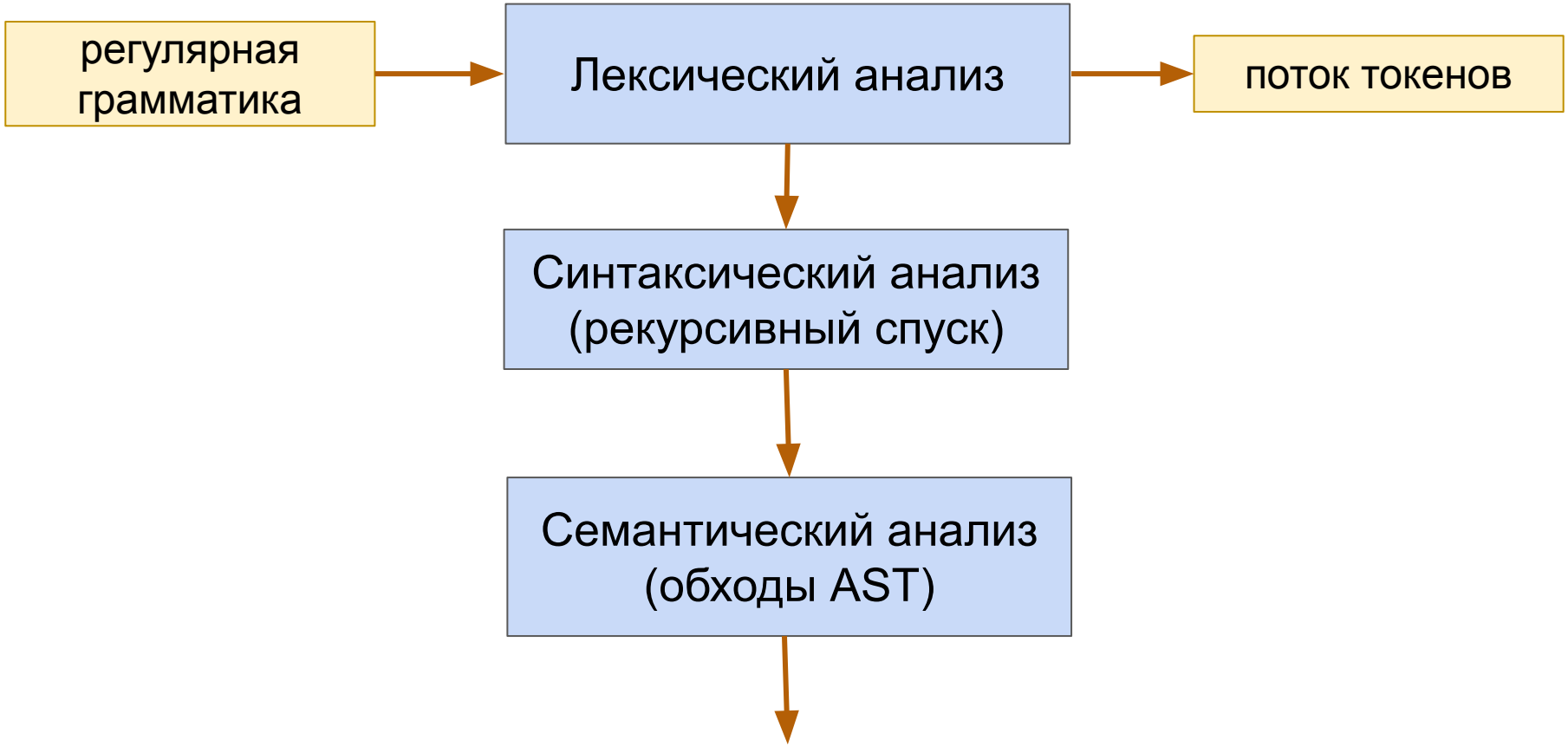
- scope

Атрибуты в AST

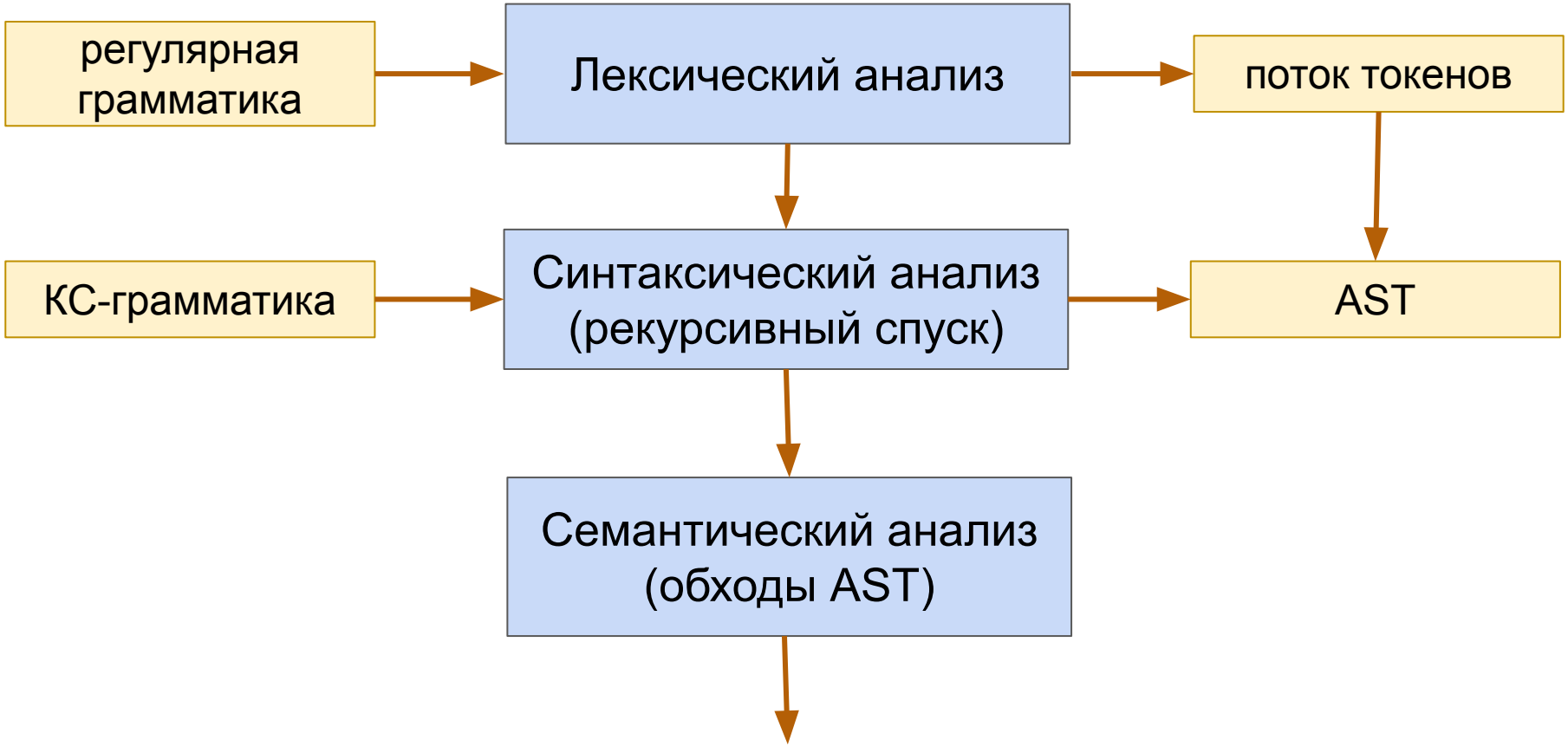
Атрибуты в AST



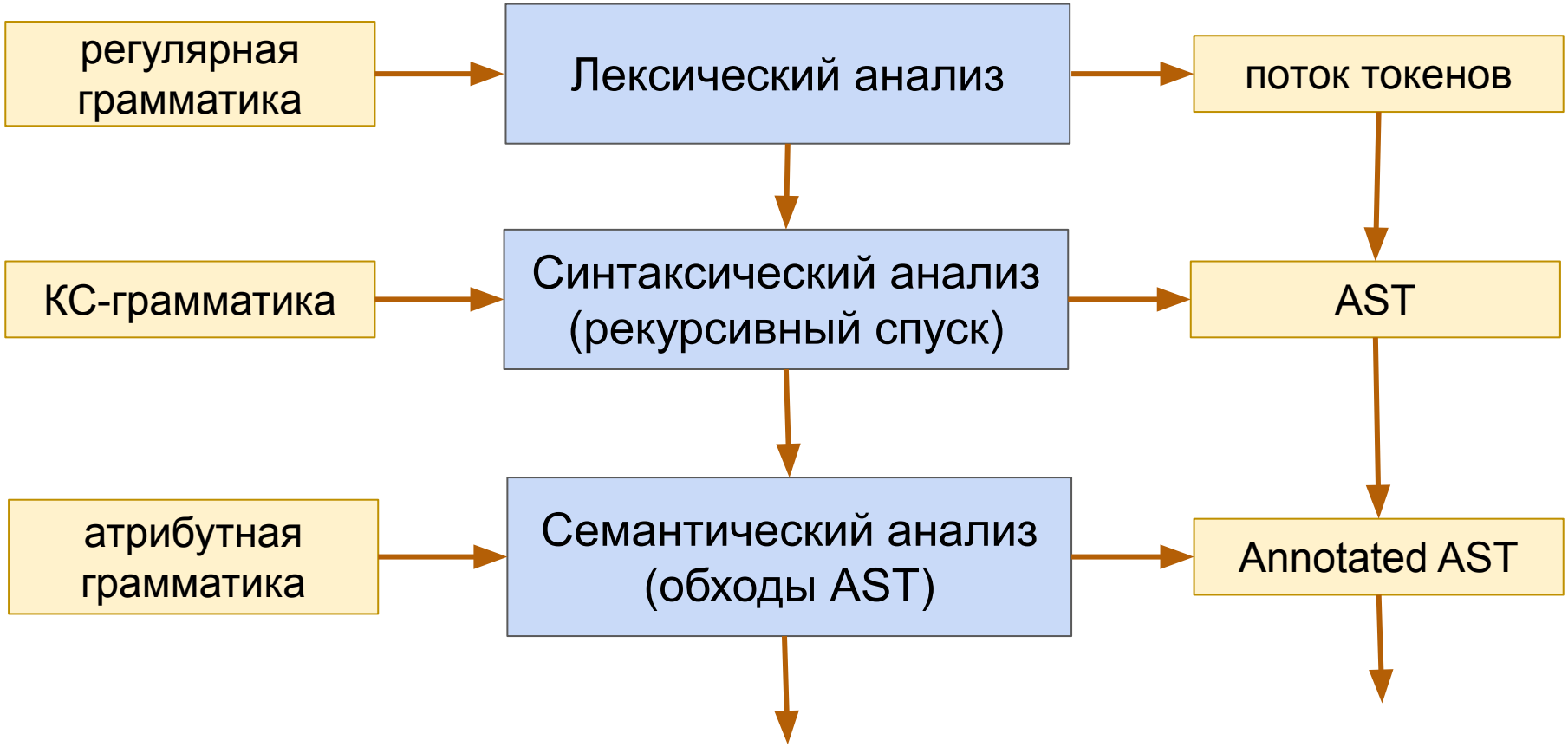
Атрибуты в AST



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) { ... }
}
```

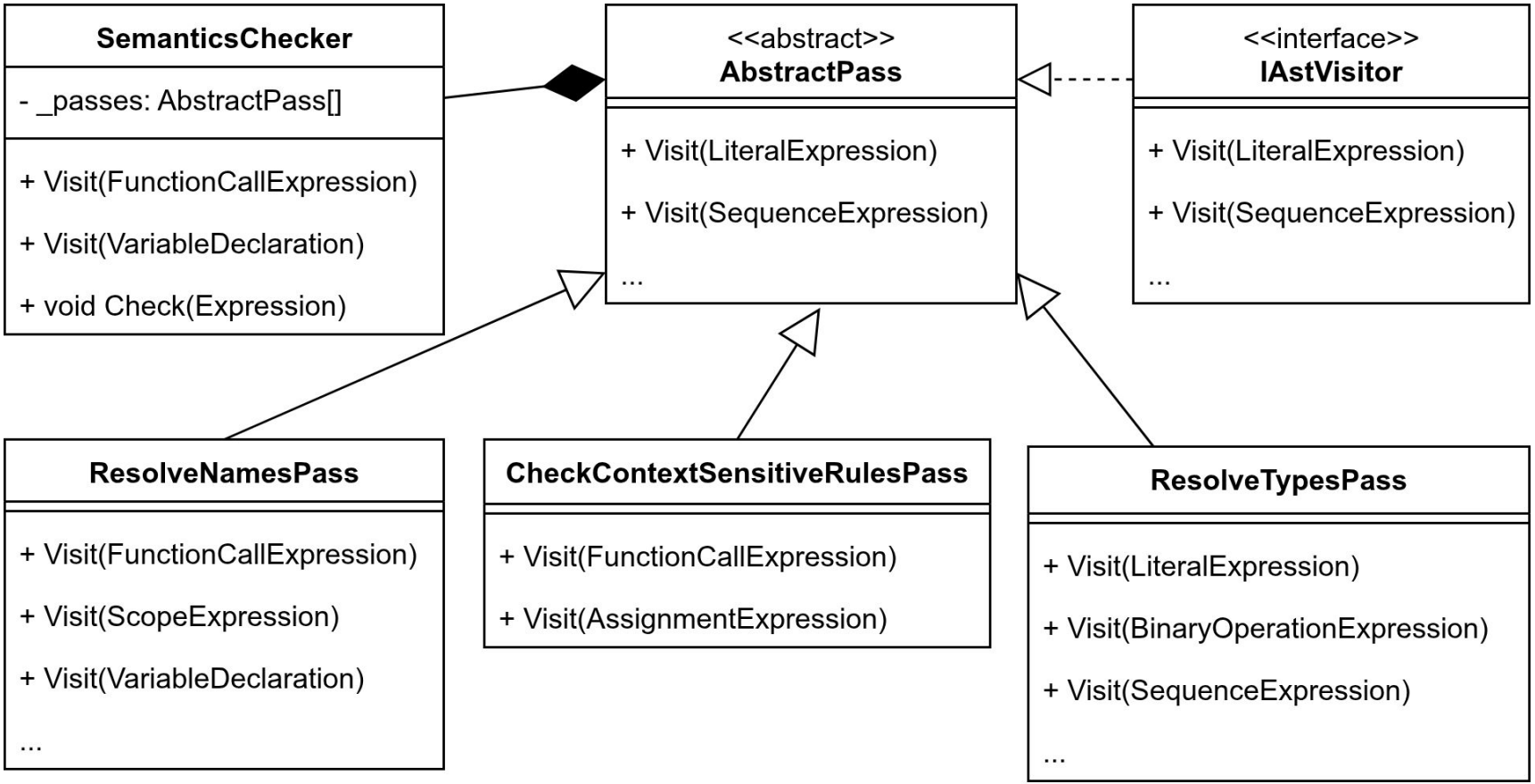
Класс ReferenceEqualityComparer<T>

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);
    }
}
```

Проверка семантики

Обработка в несколько проходов

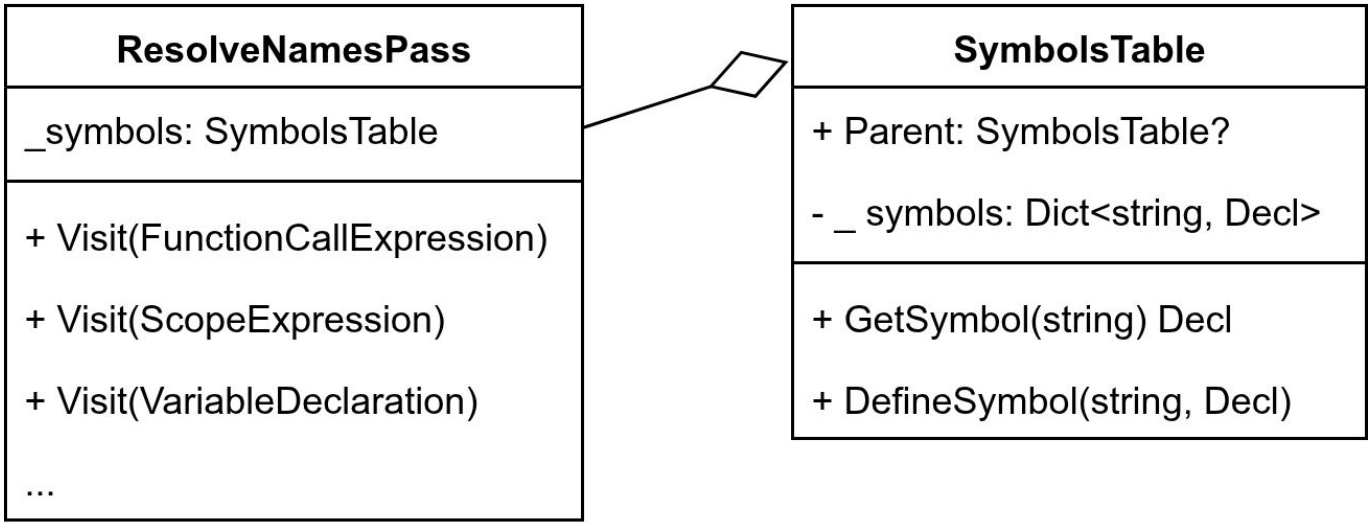


Обработка в несколько проходов

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

Таблица символов

Класс таблицы символов



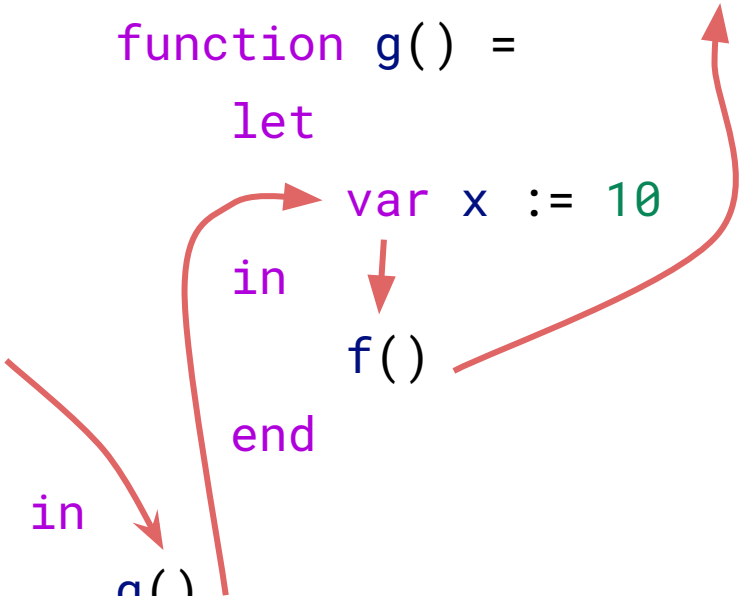
Области видимости

Динамическая область видимости

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

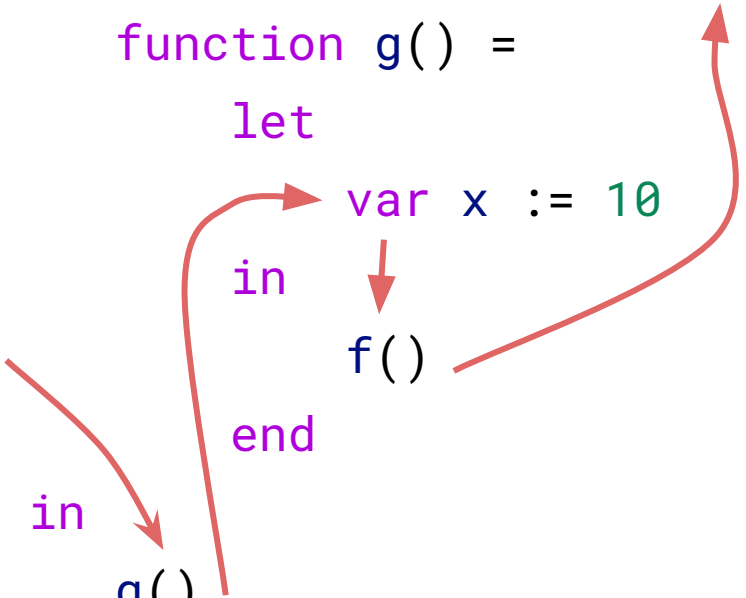
Динамическая область видимости

```
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

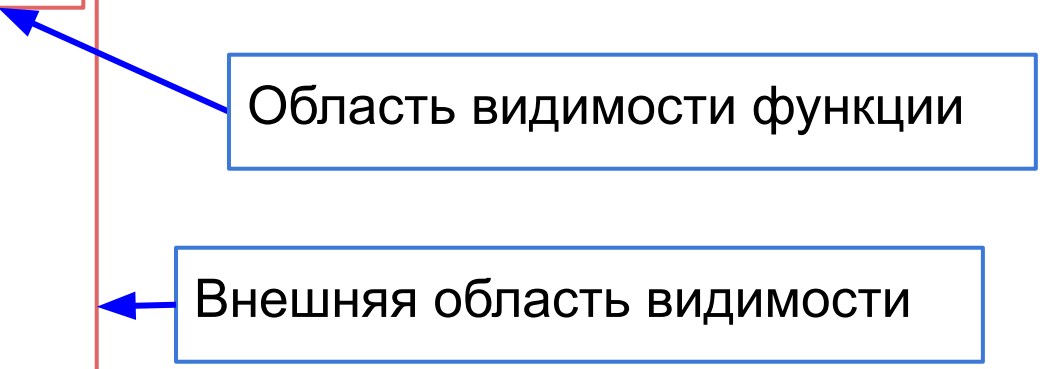
Набор переменных зависит от порядка вызова функций

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
```

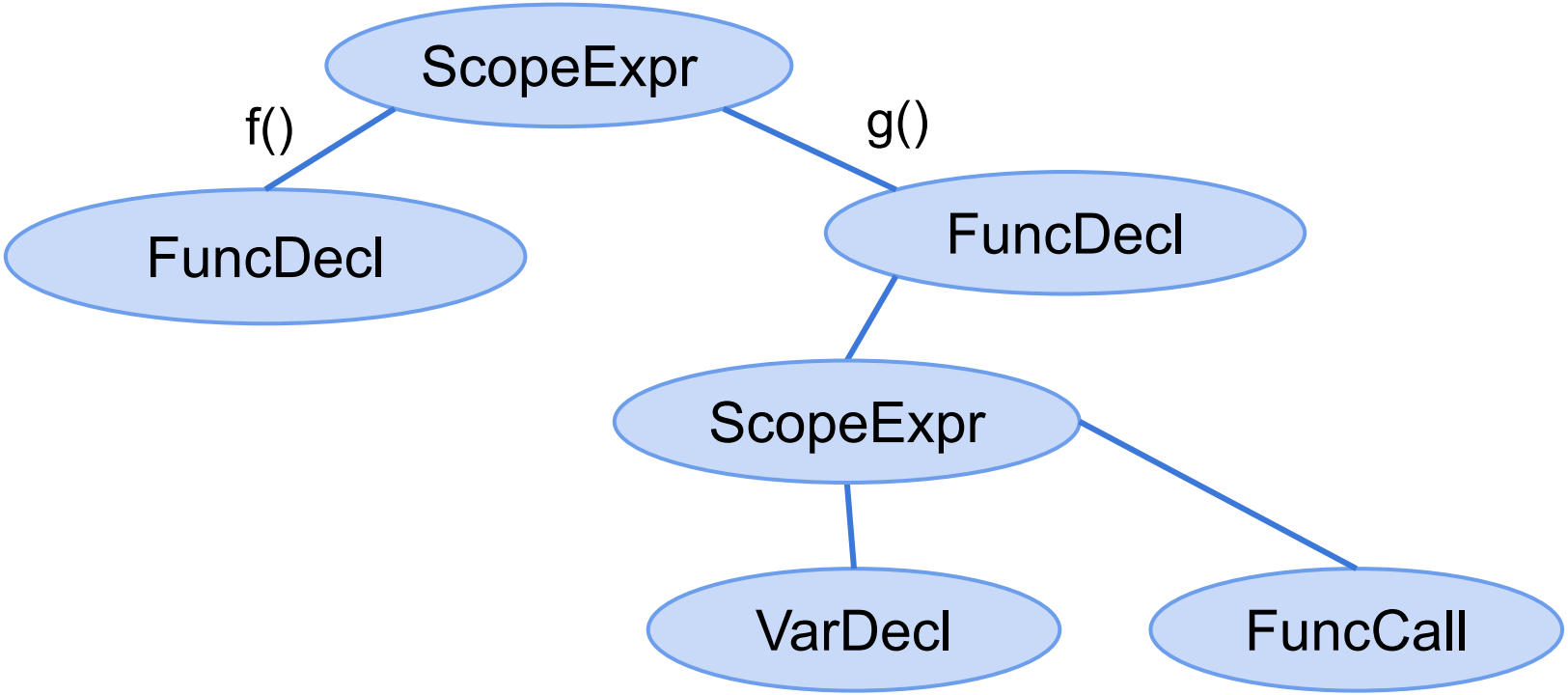
No variable with name "x"

Область видимости функции

Внешняя область видимости

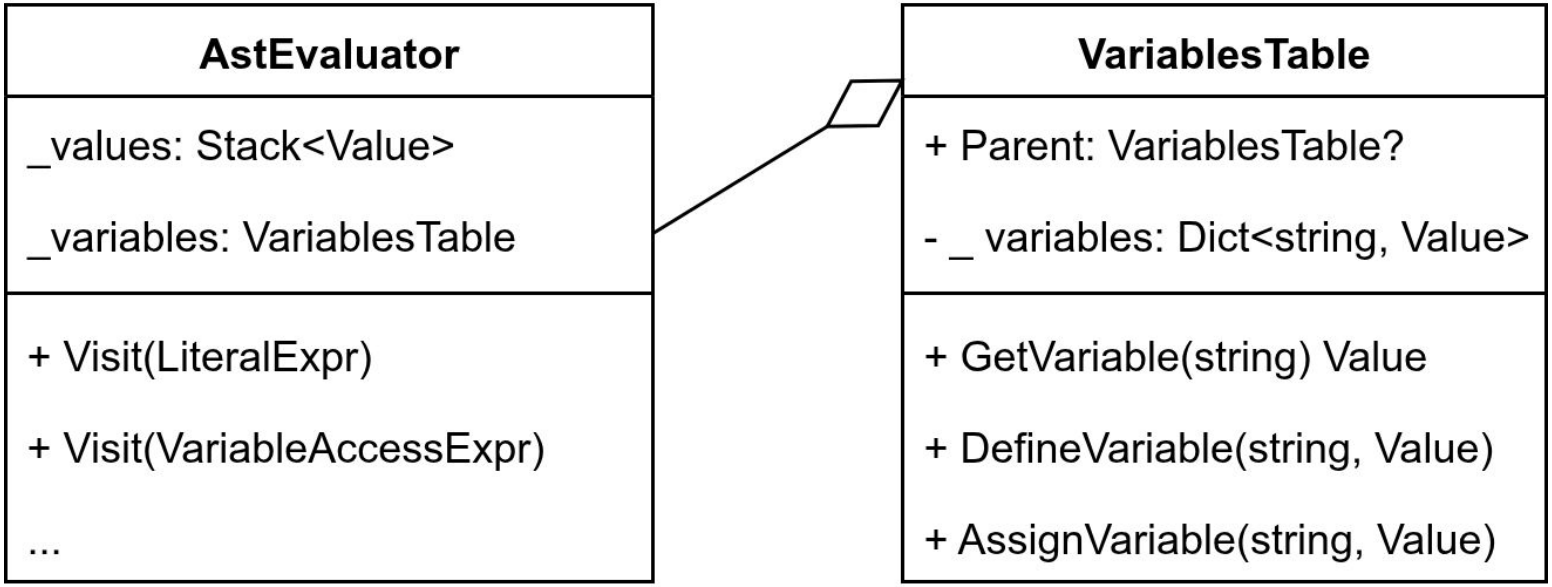
Набор переменных зависит от расположения функции в коде

Вычисление атрибута “scope”



Интерпретация по AST

Таблица переменных

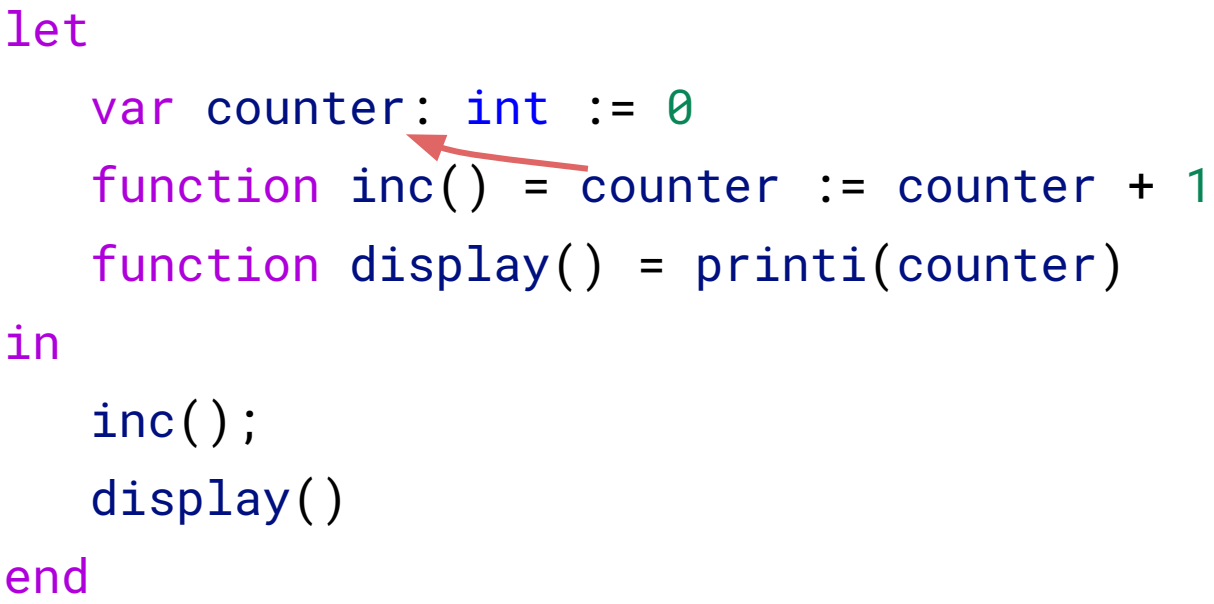


Замыкание (Closure)

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

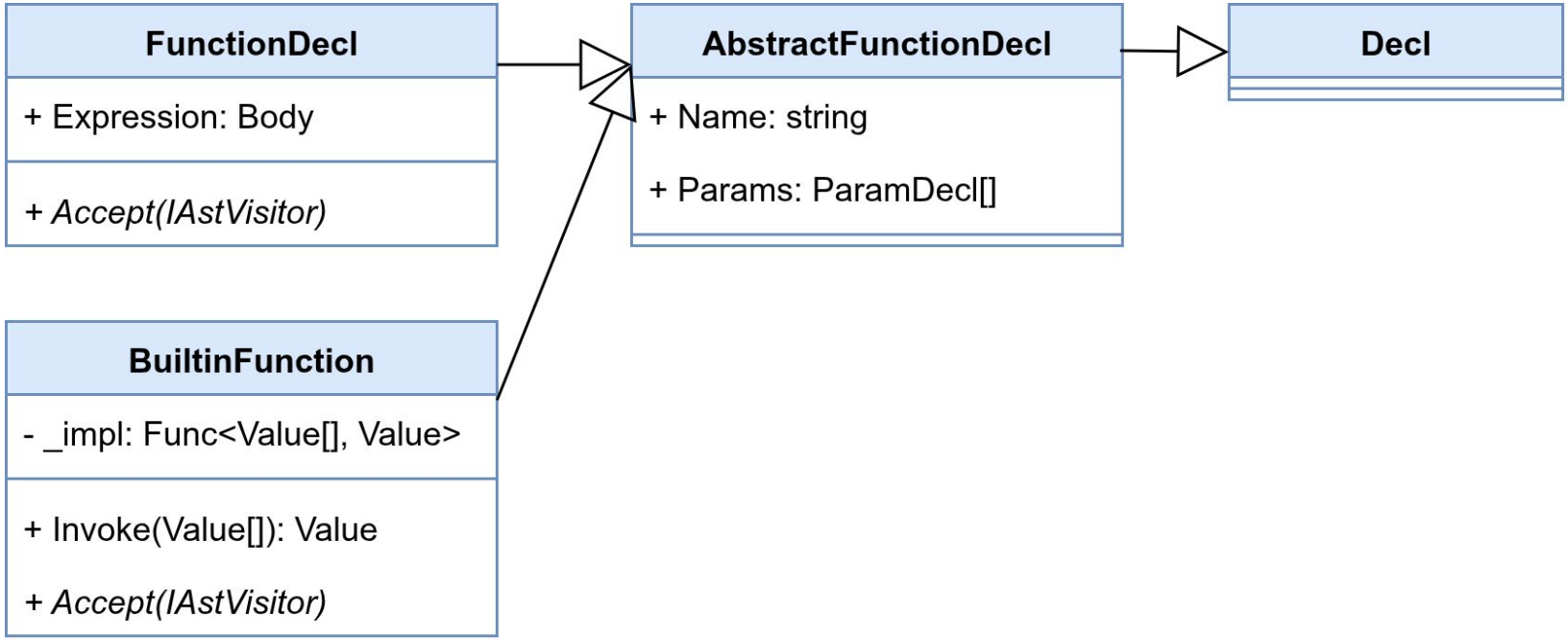
Шаг 2. Создание VariablesTable, дочернего от захваченного

Замена таблицы переменных при вызове

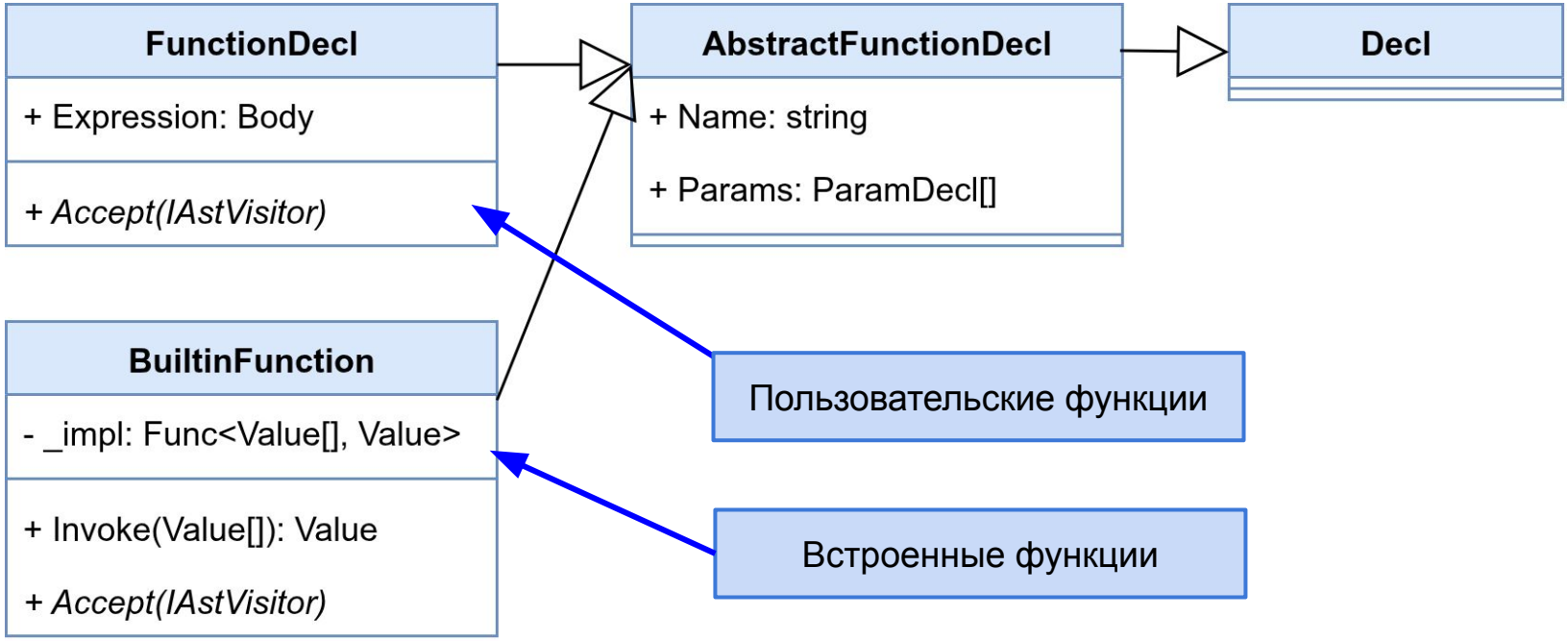
```
private void InvokeFunction(FuncCallExpr e, FuncDecl fn) {
    VariablesTable captured = _context.GetCapturedVariables(fn);
    VariablesTable oldVariables = _variables;
    _variables = new VariablesTable(parent: captured);
    // ...вычисляем аргументы, записываем в таблицу переменных.
    // ...вызываем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

Поддержка рекурсии

AST для функций



AST для функций

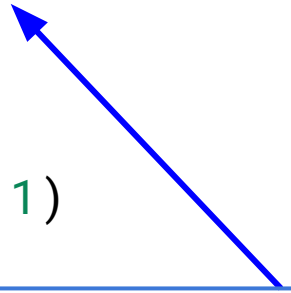


Рекурсия

```
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:

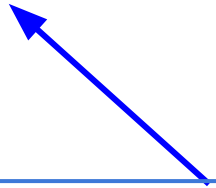
1. Добавить функцию в SymbolsTable
2. Обойти параметры и тело функции

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

Подытожим

Аналитикам на подумать

Для функций:

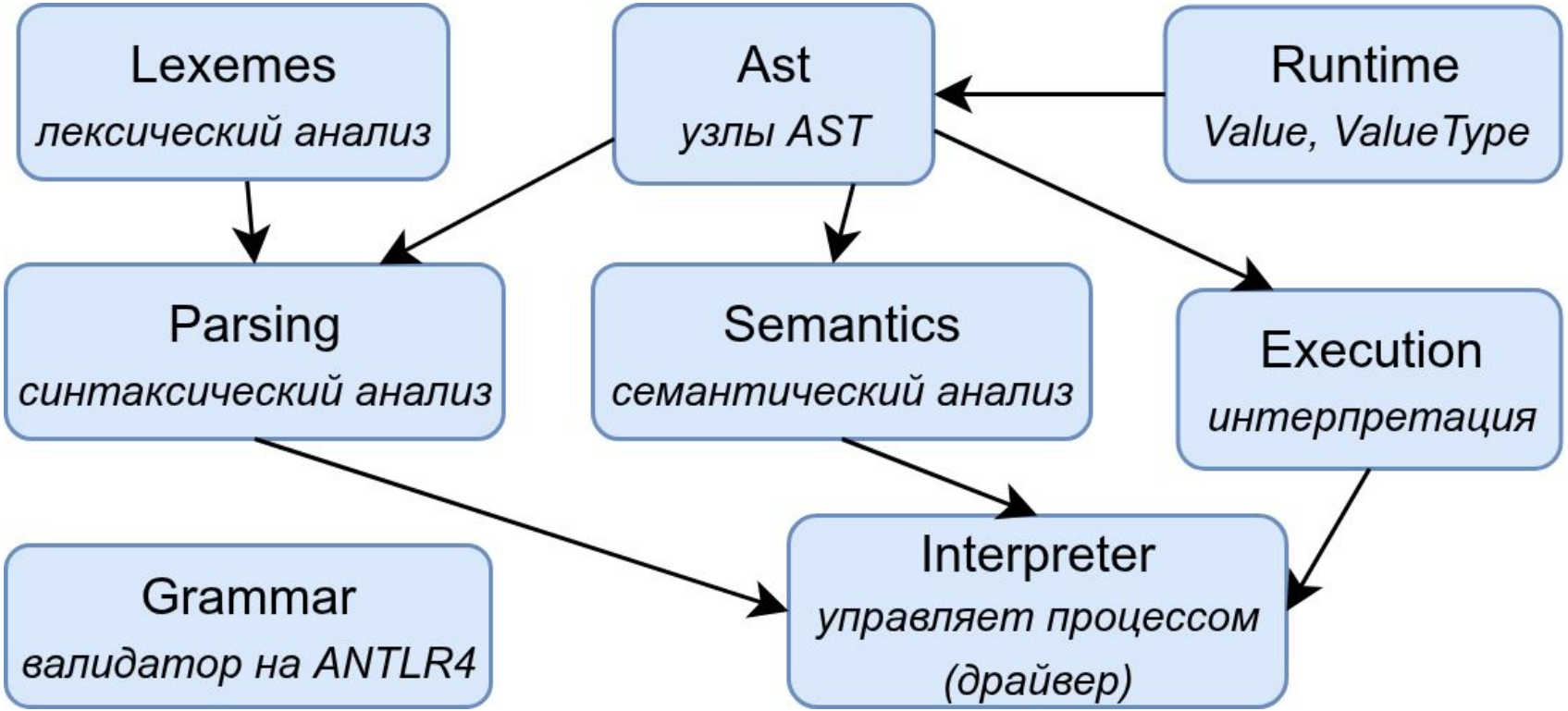
1. Области видимости — лексические или динамические?
2. Есть ли рекурсия?
3. Есть ли взаимная рекурсия?
4. Есть ли замыкания (вложенные функции)?

Интерпретатор языка Tiger

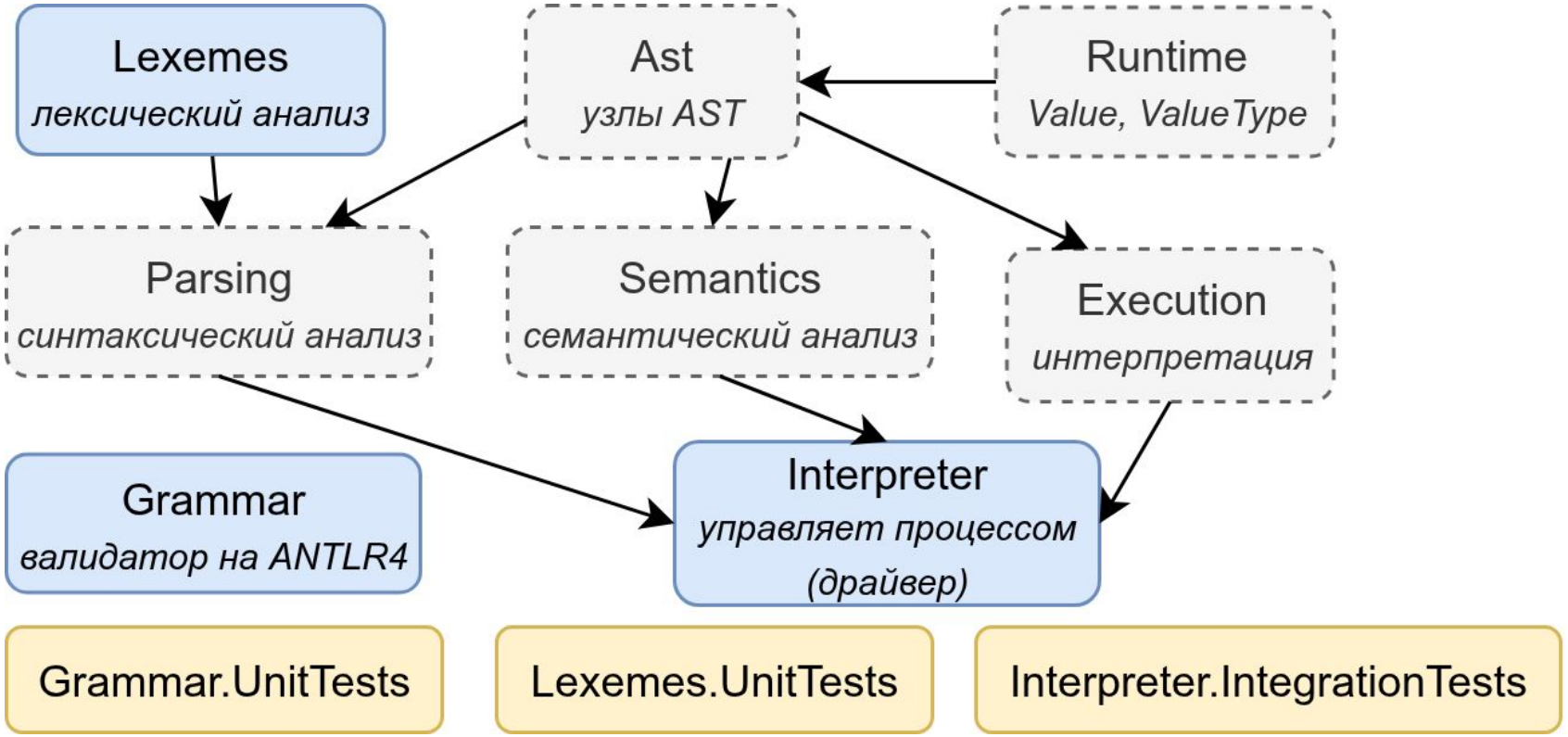
https://sourcecraft.dev/sshambir-public/pstiger

Ветка	Что добавляет
01_grammar_checker	Валидатор синтаксиса на ANTLR4
02_lexical_analysis	Лексический анализ Tiger
03_expressions	Разбор выражений
04_variables	Переменные и присваивания
05_if_else	Ветвления
06_functions	Пользовательские функции и процедуры

Модули проекта PsTiger



Тесты проекта PsTiger



Спасибо за внимание!
Вопросы?