ОСОБЕННОСТИ ПРОЕКТИРОВАНИЯ ДИАЛЕКТА РЕФАЛА, НАПРАВЛЕННОГО НА ПРАКТИЧЕСКОЕ ИСПОЛЬЗОВАНИЕ

АБРАМОВ НИКИТА

«ПРАКТИЧНОСТЬ» ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Важные черты направленного на практическое использование ЯП:

- простой и выразительный синтаксис, лёгкая читаемость;
- семантическая подсветка синтаксиса и автодополнение кода (**важнее, чем кажется на первый взгляд!**)
- *принцип наименьшего удивления* для синтаксиса;
- модульность, возможность лёгкого переиспользования кода;
- поддержка низкоуровневых вызовов и FFI (*например, для написания TCP-сервера, используя эффективные существующие реализации*).

ДИАЛЕКТ EXPREFAL

ExpRefal (Experimental Refal, *временное название*) – статически типизированный диалект Рефала, основной фокус которого – использование как практического языка программирования, подходящего для выполнения повседневных задач и автоматизации.

ExpRefal находится в стадии активного проектирования.

Данный доклад посвящён рассмотрению некоторых особенностей диалекта *ExpRefal*, в частности, его системы типизации.

REFAL НА ПРАКТИКЕ

Сильные стороны Рефала для практического программирования:

- богатые средства для *сопоставления с образцом*;
- *простота изучения*;
- лаконичность и выразительность.

Потенциальные области применения Рефала на практике (без учёта педагогических применений):

- (основная) использование в качестве *скриптового языка, в том числе как компонента прикладных программ*;
- работа с алгоритмами, использующими рекурсивные или последовательные структуры, такие как строки и деревья;
- использование в компиляторах и интерпретаторах, для *генерации кода и высокоуровневых оптимизаций*.

КЛАССИЧЕСКИЙ РЕФАЛ И ПРИНЦИП НАИМЕНЬШЕГО УДИВЛЕНИЯ

Большинство существующих диалектов Рефала поддерживают обратную совместимость с более ранними версиями.

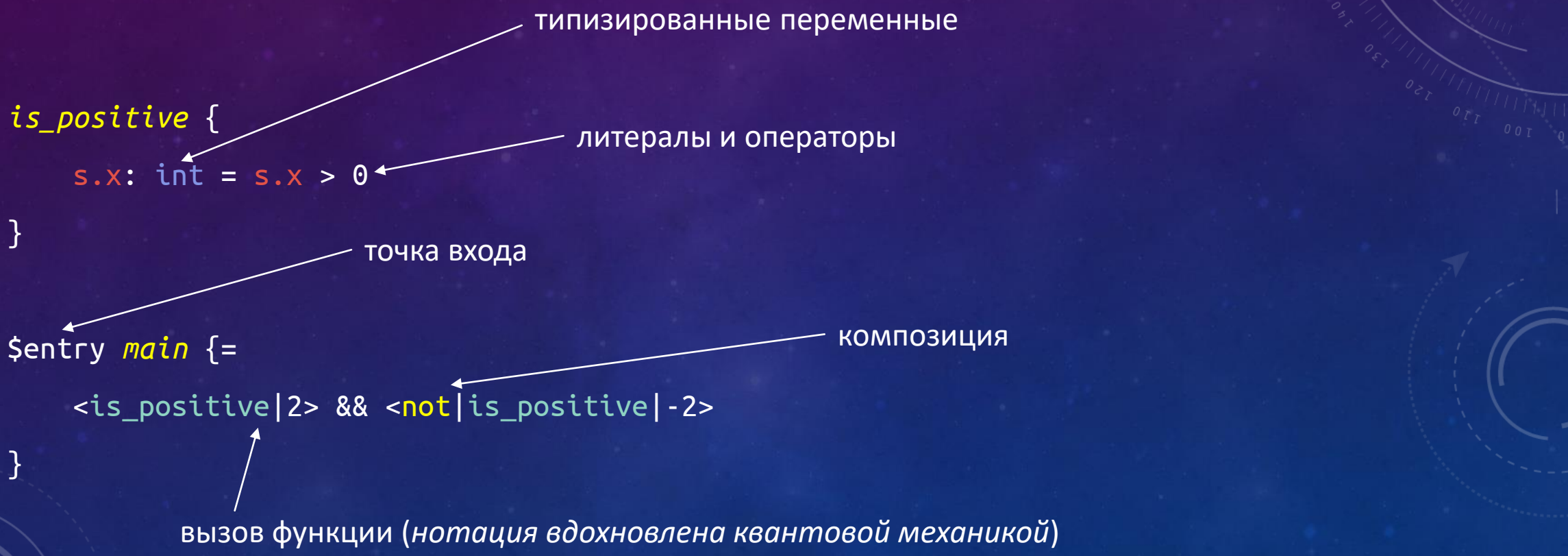
Несмотря на преимущества данного подхода, изучающему Рефал программисту, привыкшему к современным языкам программирования, таким как Rust, Kotlin, Haskell, приходится столкнуться с такими конструкциями (взяты из Refal-5λ), как

- *макроцифры: в современных ЯП детали реализации арифметики скрыты от программиста;*
- *функция Card (она пришла из ранних диалектов): по названию трудно догадаться об её назначении, если не знать историю вычислительной техники!*

*В диалекте ExpRefal было принято решение **не поддерживать обратную совместимость** с предыдущими диалектами Рефала в целях соответствия принципу наименьшего удивления.*

EXPREFAL: ОСНОВНОЙ СИНТАКСИС

Программа – последовательность функций, определяемых *сопоставлением с образцами*.




```
is_positive {
    s.x: int = s.x > 0
}

$entry main {=
    <is_positive|2> && <not|is_positive|-2>
}
```

ТИПИЗАЦИЯ В EXPREFAL

Статическая система типизации с полиморфизмом.

Поддерживаемые типы:

- *атомарные*: `int` (комбинированный алгоритм обычной/длинной арифметики), `char`, `bool`, `float`, `double`;
- *специальные*: `empty`, `()` (единичный тип), `dyn` (универсальный тип);
- *составные*: функциональный тип `T -> V`, списки `list<T>`, словари `map<K, V>`, кортежи `(...)`, массивы `array<T>`, опциональный тип `?`, сумма `|`.

Потоковый тип `[T]` – абстрактное представление типов-последовательностей: `list`, `map`, `array`.

ЛЯМБДА-ФУНКЦИИ И ПЕРЕМЕННЫЕ

Лямбда-функции определяются с помощью фигурных скобок {…}.

Семантика классов переменных не совпадает с классическим Рефалом:

- *s-переменные*: сопоставляются с одним значением определённого типа;
- *e-переменные*: сопоставляются с потоковым типом, элементы которого имеют определённый тип;
- *t-переменные*: пока отсутствуют, но могут быть введены в процессе проектирования.

указание типа возвращаемого значения для рекурсивной функции

```
map: [b] {
    ([], s.f: a -> b) = []
    ([s.x: a, e.x: a], s.f: a -> b) = [<s.f|s.x>, <map|(e.x, s.f)>]
}
$entry main {=
    <map|([1,2,3], { s.x: int = s.x * 2 })>
}
```

АЛГОРИТМЫ РЕАЛИЗАЦИИ ТИПИЗАЦИИ В EXPREFAL

АВТОМАТИЧЕСКАЯ ТИПИЗАЦИЯ ФУНКЦИЙ

В ExpRefal для всех функций (в том числе лямбда-функций) используется автоматическая типизация для вывода типа функции. Для рекурсивных функций необходимо указывать значение типа результата.

Тип функции

f {

<образец1> = <выражение1>

…

<образец_n> = <выражение_n>

}

определяется как $(\mathrm{O}_1 \mid \ldots \mid \mathrm{O}_n) \rightarrow (\mathrm{B}_1 \mid \ldots \mid \mathrm{B}_n)$.

УПРОЩЕНИЕ ТИПОВ

Типы для образцов и выражений строятся с помощью *унификации и структурной индукции*.

При этом также возможно реализовать *алгоритм упрощения типов*, выполняющий следующие преобразования типов:

- `a | a => a`
- `a | b =>` b, если a является подтипом b.

Применение данных простых правил превращает автоматические функциональные типы вида

$$\text{list}\langle\text{double}\rangle|\text{list}\langle\text{double}\rangle \to \text{double}|\text{float}|\text{float}$$

в сокращённые типы вида

$$\text{list}\langle\text{double}\rangle \to \text{double}$$

ПРИВЕДЕНИЕ ТИПОВ

Если *ожидаемый и фактический* типы передаваемого значения не совпадают, то вызывается *алгоритм автоматического приведения типов*, который пытается применить следующие преобразования подтипов в более общие типы:

- `float => double`
- `любой тип => dyn`
- `a => a | b`
- `(a | b) -> c => a -> c`
- `a -> b => a -> (b | c)`
- `list<a> => [a]` (и аналогично для других потоковых типов)

ПРИМЕРЫ ПОТЕНЦИАЛЬНОГО ПРАКТИЧЕСКОГО ИСПОЛЬЗОВАНИЯ EXPREFAL

КАК СКРИПТОВЫЙ ЯЗЫК

```
break_caesar_one {
    (s.c: array<char>, s.k: int) = <map|(s.c, {
        s.cc: char = <from_code|(<code|s.cc> - <code|'a'> - s.k) % 26 + <code|'a'>>
    })>
}

break_caesar {
    s.enc: array<char> = <most_likely|map(*, {
        s.k: int = <break_caesar_one|(s.enc, s.k)>
    })|range|(0,26)>
}
```

частичное применение функции

DATA SCIENCE И MACHINE LEARNING

```
linear_regression {
    (s.x: matrix<double>, s.y: matrix<double>) = <mul|(
        <inv|mul|(<t|s.x>, s.x)>, <mul|(s.x, s.y)>
    )>
    # ridge-регрессия, если задан параметр λ
    (s.x: matrix<double>, s.y: matrix<double>, s.l: double) = <mul|(
        <inv|add|(
            <mul|(<t|s.x>, s.x)>, <mul|(s.l, <ident|size|s.x>)>
        )>, <mul|(s.x, s.y)>
    )>
}
```

ПЕРСПЕКТИВЫ И ПЛАНЫ

Дальнейшая доработка проекта ExpRefal:

- система модулей;
- FFI для вызова функций из других ЯП;
- **реализация интерпретатора ExpRefal.**