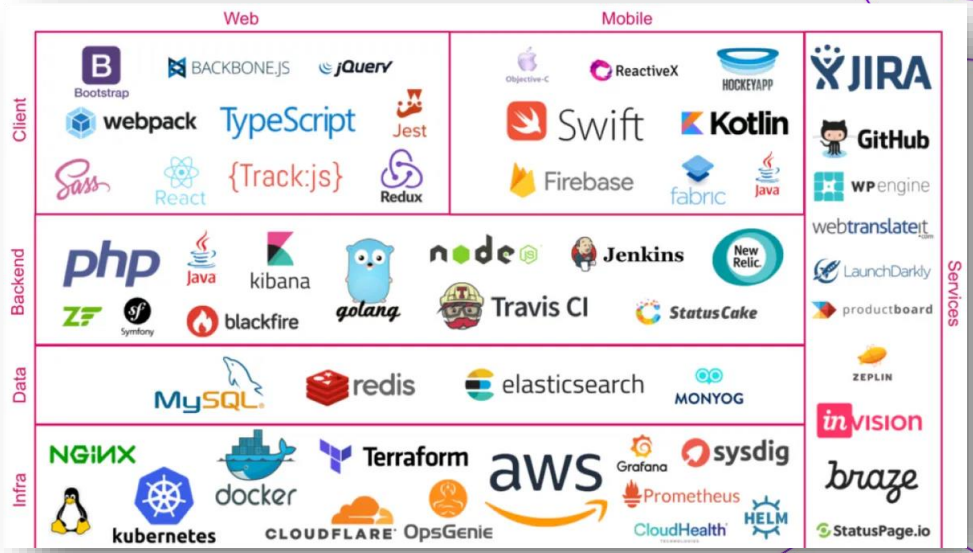
ИТМО

Выработка методов к анализу мультиязыковых текстов программ

Подготовил: Орловский М.Ю
Системное и прикладное программное обеспечение
Научный руководитель: Логинов И.П.

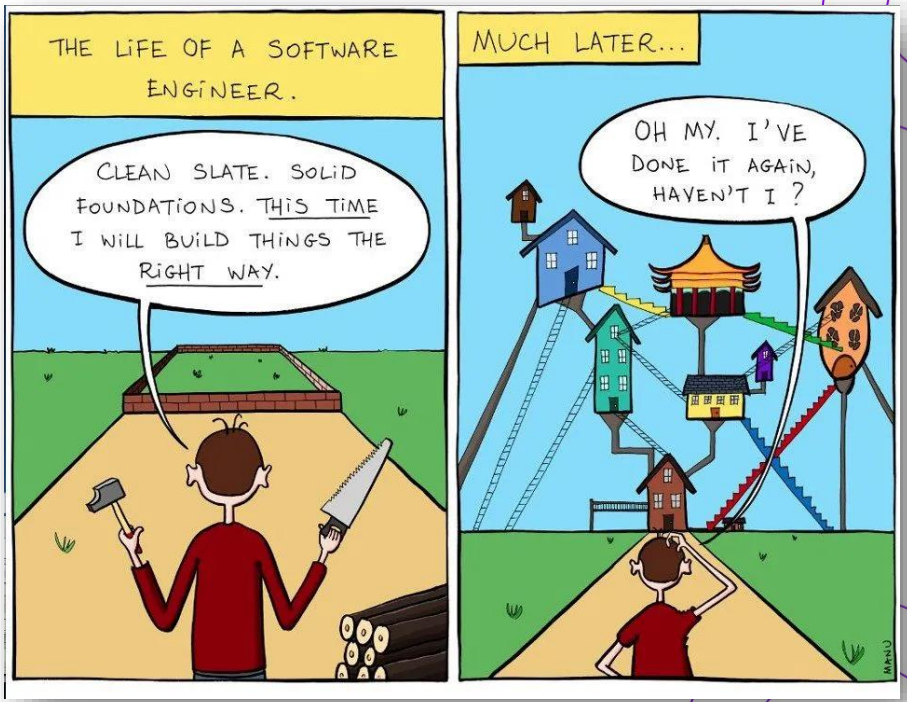
Современная разработка



Особенности разработки ПО в 2023:

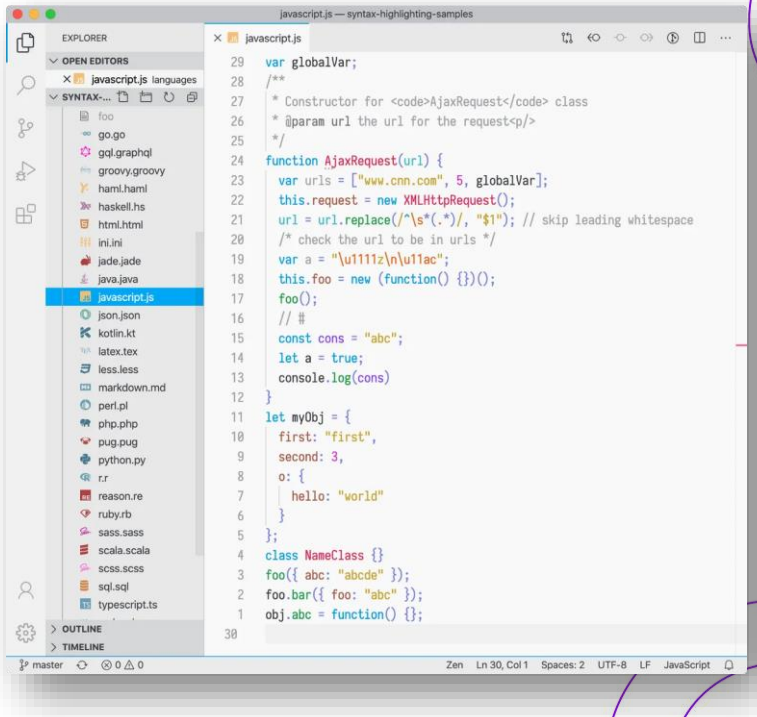
- Разнообразие инструментальных средств;
- Множество предметных областей;
- Разнообразие подходов к проектированию.

Зависимости



Многие зависимости программных компонентов неявны, и нам приходится с этим работать. Хорошо, что дебаггер всегда помогает… верно?

Интеграция с IDE



Очень редко IDE имеет поддержку навигации по мультиязыковому коду, не говоря уже о полноценном рефакторинге или сборе статистики. Обычно, мультиязыковые IDE узкоспециализированы и не предназначены для универсального использования

Постановка задачи

Цель работы:

Разработка универсального метода статического анализа мультиязыкового кода

Задачи:

- Рассмотрение предметной области мультиязыкового анализа;
- Проведение анализа современных инструментальных средств на наличие мультиязыкового анализа;
- Исследовать влияние конкретных прикладных областей на возможность обобщения анализа;
- Формализовать предлагаемый метод и выбрать оптимальные структуры данных;
- Разработать прототип анализатора, протестировать на определенных сценариях, проанализировать результаты.

О мультиязыковом анализе

На практическом уровне, мультиязыковой анализ можно разделить на два этапа:

1. Внутриязыковой (стандартный) анализ;
2. Межъязыковой анализ.

При этом, второй этап использует часть наработок из первого этапа для обеспечения большей полноты и непротиворечивости

О мультиязыковом анализе

При анализе полезно проводить разделение вовлеченных языков на две группы:

- Языки GPL (General purpose language) – являются основными инструментами для решения задач в избранной предметной области

Примеры: C++, Java, C#, Python, JavaScript и другие

- Языки DSL (Domain specific language) – являются вспомогательными языками, используемыми в определенных предметных областях

Примеры: Shell, SQL, Makefile, CMake, XML/JSON/YAML/TOML и многие другие

Анализ существующих инструментов

ИТМО

В данный момент, в индустрии имеется довольно мало инструментов, поддерживающих полноценную мультиязыковую разработку

Многие инструменты являются специализированными и проприетарными, такие как продукты компаний Microsoft, JetBrains или Apple.

Открытых решений, при этом ещё и активно поддерживаемых практически не существует

Анализ существующих инструментов

Основная функциональность существующих инструментов:

1. Обнаружение зависимостей;
2. Переименование символов по их зависимостям;
3. Контекстная информация о синтаксическом элементе;
4. Обнаружение несоответствий при межъязыковом взаимодействии;
5. Поиск определения или объявления символа;
6. Поиск всех использований определенного символа;
7. Автодополнение, доступное в межъязыковом контексте.

Влияние прикладных областей

Согласно одному из исследований*, наиболее часто сочетание различных языков представлено в следующих предметных областях:

1. Веб-разработка;
2. Клиент-серверные решения;
3. Встроенные системы.

*Mayer, P., Kirsch, M. & Le, M.A. On multi-language software development, cross-language links and accompanying tools: a survey of professional software developers. J Softw Eng Res Dev 5, 1 (2017). https://doi.org/10.1186/s40411-017-0035-z

Влияние прикладных областей

Основные проблемы в зависимости от разных областей

Веб-разработка	Очень большое количество различных языков с различным синтаксисом
Энтерпрайз, клиент-сервер	Широкая сеть распределенных проектных зависимостей, помноженная на версионность
Встроенные системы	Не следующие стандартам компиляторы, широкое использование кодогенерации

Выработка и формализация метода

Для поддержки внутриязыкового анализа решено использовать следующие структуры данных (по необходимости):

- AST (Abstract syntax tree);
- CFG (Context flow graph);
- DFG (Data flow graph).

Также, предполагается использование онтологии.

Онтология – формальная система, описывающая сущности и взаимосвязи конкретной предметной области.

Так как предметная область – все языки, онтология является универсальной

Выработка и формализация метода

Для формализации метода используется следующая терминология:

- Модель зависимостей системы – совокупность фрагментов кода, связанных между собой семантически;
- Фрагмент кода – логическая единица, представляющая фрагмент программного кода и его зависимости
- Фрагмент кода описывается сигнатурой, окружением и внутренними зависимостями;
- Окружение – зависимость от чего-либо;
- Сигнатура – предоставление чего-либо для формирования зависимости.

Выработка и формализация метода

ИТМО

Для обеспечения связывания фрагментов вводится язык, представляющий расширение типизированного лямбда-исчисления первого порядка

```
A, B ::= N | A -> B | A x B | A + B | Any | None
a, b ::= a | \(a : A)b | b(a)
```

Где N – любой номинальный тип

Выработка и формализация метода

Этапы анализа:

1. Фрагментный (внутриязыковой) анализ – генерация фрагментов с определенными окружениями, сигнатурами и внутренними связями;
2. Системный (межъязыковой) анализ – связывание фрагментов посредством связывания окружений с сигнатурами с использованием введенной онтологии и языка.

Состав онтологии:

1. Возможные межъязыковые связи и их семантика;
2. Заранее определенные типы.

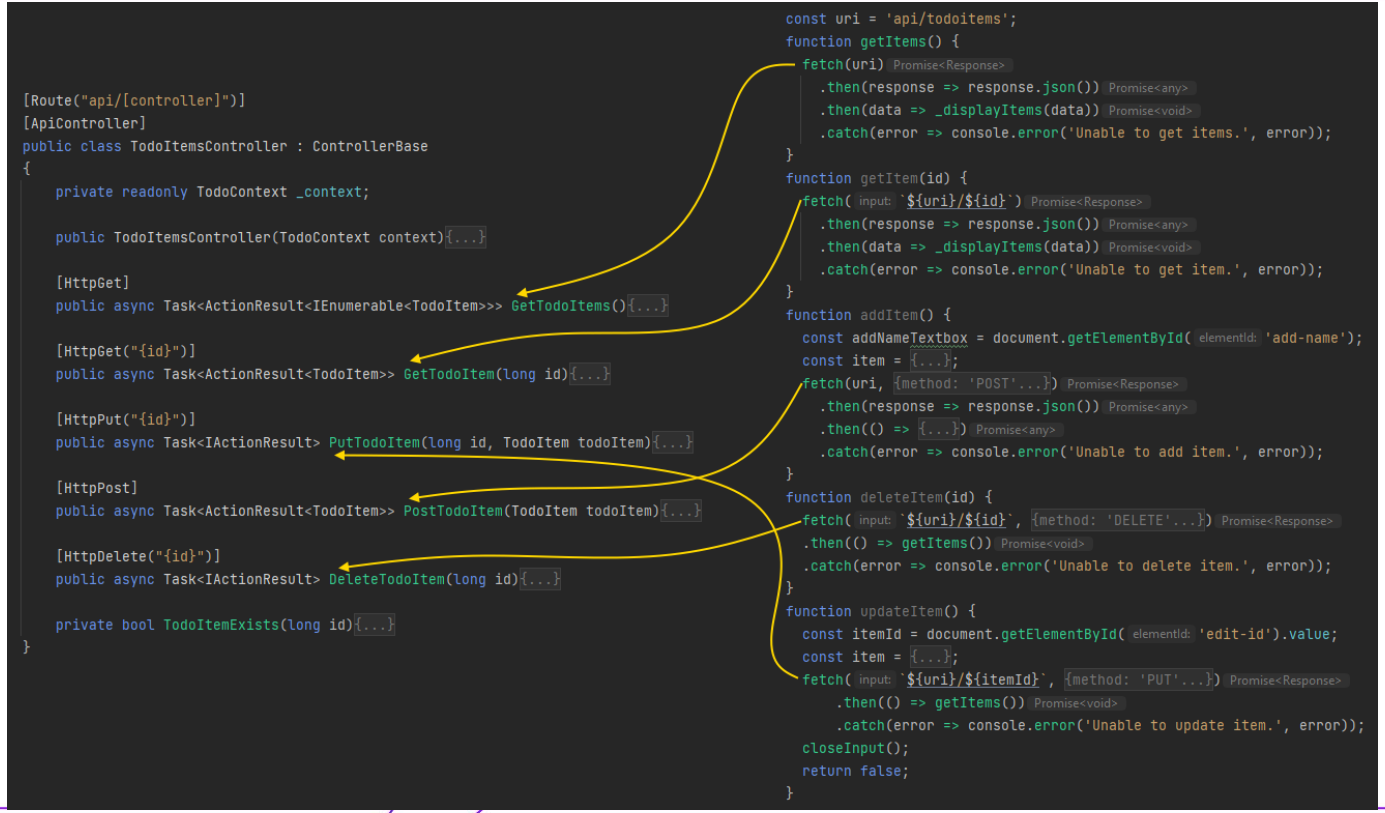
Краткое описание сценариев

Для тестирования выбраны три сценария использования в зависимости от предметной области:

1. C# и JavaScript (Веб, клиент-сервер);
2. Python, Sh и C (ML, научные вычисления);
3. Sh и C (встроенные системы).

Сценарий 1

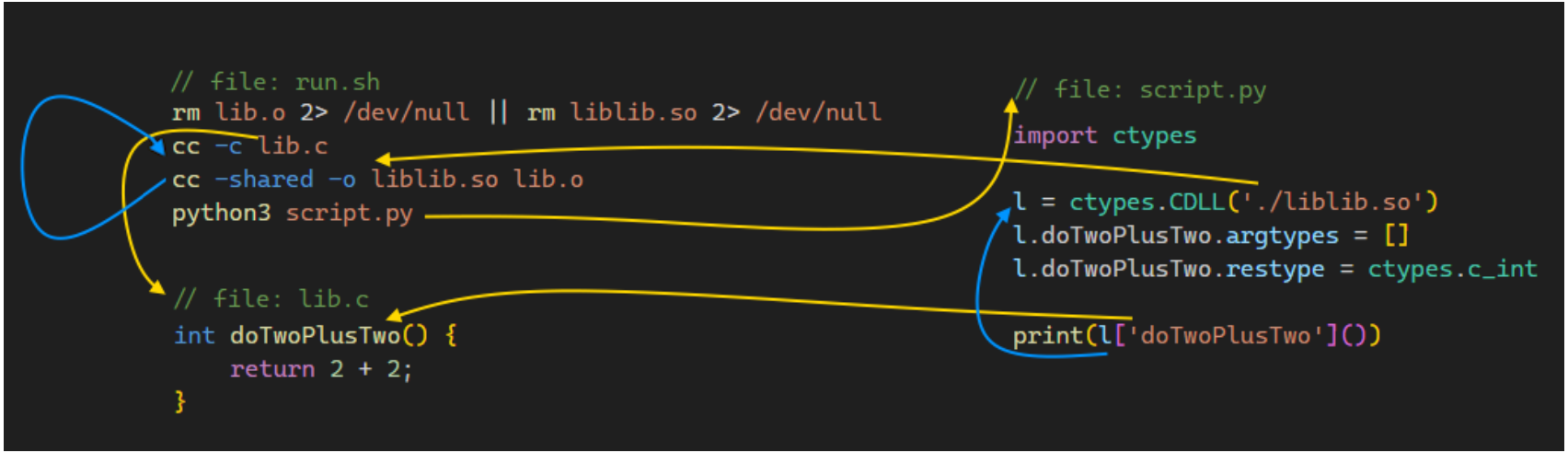
ИТМО



```
[Route("api/[controller]")]
[ApiController]
public class TodoItemsController : ControllerBase
{
    private readonly TodoContext _context;

    public TodoItemsController(TodoContext context){...}

    [HttpGet]
    public async Task<ActionResult<IEnumerable<TodoItem>>> GetTodoItems(){...}

    [HttpGet("{id}")]
    public async Task<ActionResult<TodoItem>> GetTodoItem(long id){...}

    [HttpPut("{id}")]
    public async Task<IActionResult> PutTodoItem(long id, TodoItem todoItem){...}

    [HttpPost]
    public async Task<ActionResult<TodoItem>> PostTodoItem(TodoItem todoItem){...}

    [HttpDelete("{id}")]
    public async Task<IActionResult> DeleteTodoItem(long id){...}

    private bool TodoItemExists(long id){...}
}
```

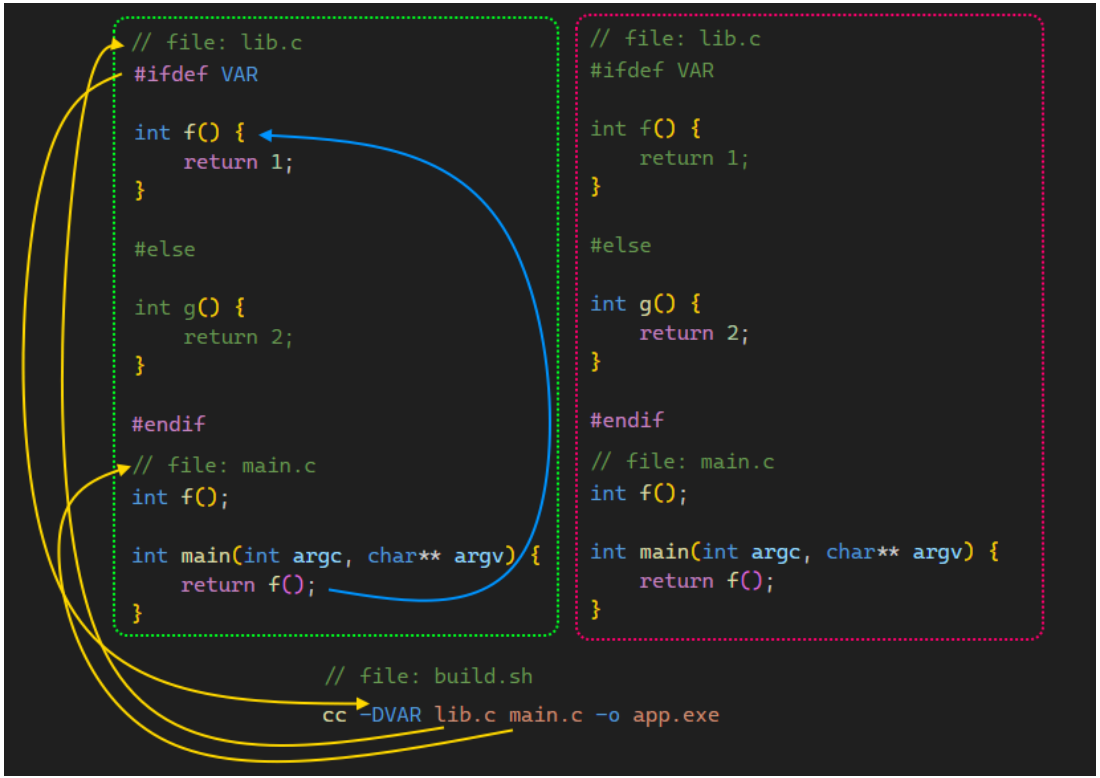
```
const uri = 'api/todoitems';
function getItems() {
    fetch(uri) Promise<Response>
        .then(response => response.json()) Promise<any>
        .then(data => _displayItems(data)) Promise<void>
        .catch(error => console.error('Unable to get items.', error));
}
function getItem(id) {
    fetch( input: `${uri}/${id}`) Promise<Response>
        .then(response => response.json()) Promise<any>
        .then(data => _displayItems(data)) Promise<void>
        .catch(error => console.error('Unable to get item.', error));
}
function addItem() {
    const addNameTextbox = document.getElementById( elementId: 'add-name');
    const item = {...};
    fetch(uri, {method: 'POST'...}) Promise<Response>
        .then(response => response.json()) Promise<any>
        .then(() => {...}) Promise<any>
        .catch(error => console.error('Unable to add item.', error));
}
function deleteItem(id) {
    fetch( input: `${uri}/${id}`, {method: 'DELETE'...}) Promise<Response>
    .then(() => getItems()) Promise<void>
    .catch(error => console.error('Unable to delete item.', error));
}
function updateItem() {
    const itemId = document.getElementById( elementId: 'edit-id').value;
    const item = {...};
    fetch( input: `${uri}/${itemId}`, {method: 'PUT'...}) Promise<Response>
        .then(() => getItems()) Promise<void>
        .catch(error => console.error('Unable to update item.', error));
    closeInput();
    return false;
}
```

Сценарий 2


// file: run.sh
rm lib.o 2> /dev/null || rm liblib.so 2> /dev/null
cc -c lib.c
cc -shared -o liblib.so lib.o
python3 script.py
// file: lib.c
int doTwoPlusTwo() {
return 2 + 2;
}
// file: script.py
import ctypes
l = ctypes.CDLL('./liblib.so')
l.doTwoPlusTwo.argtypes = []
l.doTwoPlusTwo.restype = ctypes.c_int
print(l['doTwoPlusTwo']())

Сценарий 3

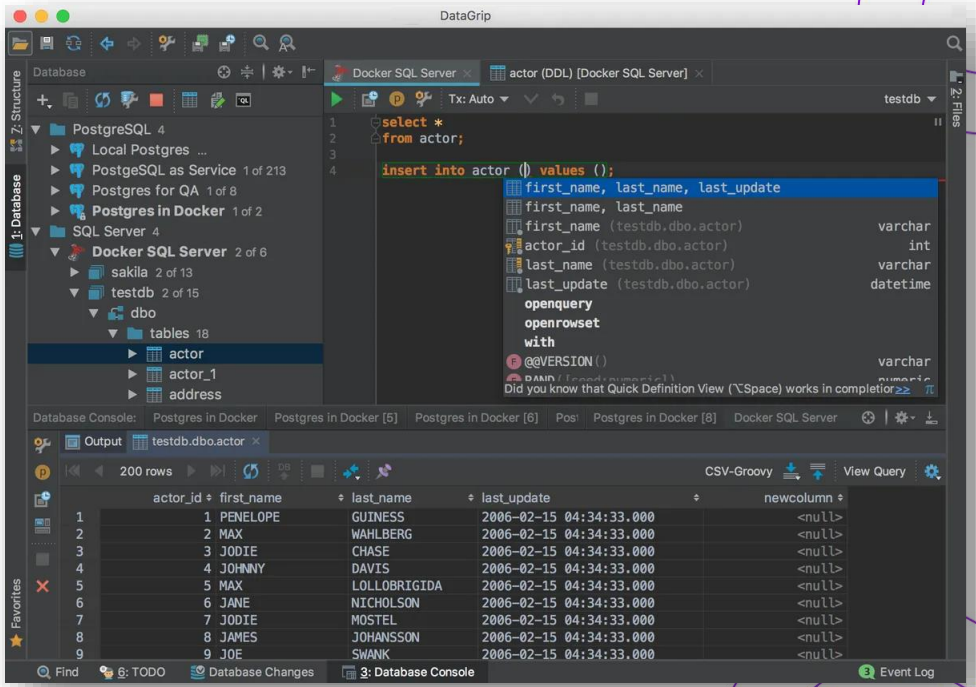
ИТМО



Ограничения и особенности метода

- Независим относительно технологий;
- Анализ асинхронный;
- Используется заранее выработанная, формальная онтология;
- Семантика извлекается косвенно при фрагментном анализе;
- Фрагментный анализ может являться основным источником семантики;
- Простая система типов;
- Сложности с поддержкой условной компиляции и кодогенерации (что на самом деле является отдельной нетривиальной задачей).

Дальнейшие исследования



Возможные направления:

- Более мощная система типов, отвечающая потребностям анализа;
- Разработка и внедрение метода в существующий внутриязыковой анализатор;
- Исследование более сложных систем модулей для их адаптации.

Спасибо
за внимание!

ITMO more than a UNIVERSITY

Github.com/uberdever
t.me: @uberdever