**Praca inżynierska**

**Karol Etrych**

kierunek studiów: **informatyka stosowana**

Opiekun: dr inż. Janusz Malinowski

**Kraków, styczeń 2017**

Oświadczam, świadomy odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem osobiście i samodzielnie i nie korzystałem ze źródeł innych niż wymienione w pracy.

.................................................................

(czytelny podpis)

Spis treści

[1. Wstęp 4](#_Toc502739568)

[1.1. Założenia projektu. 4](#_Toc502739569)

[1.2. Mechanizm wnioskowania typów. Przykłady w innych językach C#, F#, Scala. 4](#_Toc502739570)

[1.3. generujący kod zgodny ze specyfikacją ECMA-335 4](#_Toc502739571)

[2. Parser 5](#_Toc502739572)

[1.1. AST 5](#_Toc502739573)

[1.2. Wybór sposobu implementacji parsera. 6](#_Toc502739574)

[1.3. Zasada działania parsera typu top-down. 6](#_Toc502739575)

[1.4. Problemy przy implementacji parsera. 6](#_Toc502739576)

[3. Struktura projektu 6](#_Toc502739577)

[1.1. Rezultaty pośrednie kompilacji (Railway oriented programming). 6](#_Toc502739578)

[1.2. Przetwarzanie AST. Katamorfizmy. 6](#_Toc502739579)

[4. Rozwiązywanie typów. 6](#_Toc502739580)

[1.1. Sprawdzanie czy typ jest zdefiniowany. 6](#_Toc502739581)

[1.2. Podmienianie specyfikatorów typu na w pełni kwalifikowane. 6](#_Toc502739582)

[5. Wnioskowanie typów. 6](#_Toc502739583)

[1.1. Algorytm działania. 6](#_Toc502739584)

[1.2. Algorytm znajdowania least-upper-bound. 6](#_Toc502739585)

[1.3. Dlaczego w AST pojawia się <'Expression> ? 6](#_Toc502739586)

[6. Sprawdzanie semantyki. 6](#_Toc502739587)

[1.1. Co jest sprawdzane. 6](#_Toc502739588)

[1.2. Reguły semantyki. 7](#_Toc502739589)

[7. Generowanie reprezentacji pośredniej. 7](#_Toc502739590)

[8. Generowanie Intermediate Language. 7](#_Toc502739591)

[9. Użyte narzędzia i podsumowanie. 7](#_Toc502739592)

[1.1. F#, VSCode+Ionide, FParsec, Argu, Expecto, .NET Core 7](#_Toc502739593)

[1.2. Dalszy rozwój projektu. 7](#_Toc502739594)

[10. Użycie kompilatora 7](#_Toc502739595)

[11. Bibliografia 7](#_Toc502739596)

# Wstęp

## Założenia projektu.

Planowane cechy języka to:

* Statyczne typowanie.
* Paradymaty: imperatywny, proceduralny, obiektowy
* Mechanizm wnioskowania typów.

Planowane cechy kompilatora:

* Wyjściowy kod zgodny z
* Możliwość używania bibliotek skompilowanych na platformę .NET.

## Mechanizm wnioskowania typów. Przykłady w innych językach C#, F#, Scala.

## generujący kod zgodny ze specyfikacją ECMA-335

1. Semantyka języka

Poniżej przedstawiono możliwości języka oraz zasady jego semantyki.

* 1. Moduły

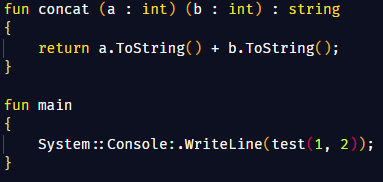
Wszelki kod umieszczany jest w modułach. Każdy plik z kodem źródłowym tworzy moduł. Przestrzeń nazw dla modułu może zostać określona z użyciem słowa kluczowego module:

**module** Project::Component::ExampleModule

Deklaracja modułu jest opcjonalna. Jeżeli nie zostanie podana, nazwa wynikowego modułu zostanie określona na bazie nazwy pliku i jego lokalizacji.

* 1. Funkcje

Funkcje definiowane są z użyciem słowa kluczowego *fun.* Parametry funkcji oraz ich typy podawane są w nawiasach. Typ zwracany podawany jest po parametrach. Ciało funkcji zawiera się w nawiasach klamrowych:



* 1. Klasy
  2. Typy
  3. Instrukcje
  4. Wyrażenia

# Architektura kompilatora

# Moduły

# Rezultaty pośrednie kompilacji (Railway oriented programming).

# Przetwarzanie AST. Katamorfizmy.

# Parser

## AST



## Wybór sposobu implementacji parsera.

## Zasada działania parsera typu top-down.

## Problemy przy implementacji parsera.

# Rozwiązywanie typów.

## Sprawdzanie czy typ jest zdefiniowany.

## Podmienianie specyfikatorów typu na w pełni kwalifikowane.

# Wnioskowanie typów.

## Algorytm działania.

Funkcje poniżej mogą bazować na typach funkcji zdefiniowanych w tym samym module wcześniej.

## Algorytm znajdowania least-upper-bound.

## Dlaczego w AST pojawia się <'Expression> ?

# Sprawdzanie semantyki.

## Co jest sprawdzane.

## Reguły semantyki.

1. Operandy operatorów binarnych muszą być tego samego typu.

2. Funkcje rekurencyjne muszą mieć podany zwracany typ.

# Generowanie reprezentacji pośredniej.

# Generowanie Intermediate Language.

# Użyte narzędzia i podsumowanie.

## F#, VSCode+Ionide, FParsec, Argu, Expecto, .NET Core

## Dalszy rozwój projektu.

W ramach dalszego rozwoju języka planowane jest dodanie nienulowalnych typów referencyjnych oraz wersja na .NET Standard (Linux).

# Użycie kompilatora

# Bibliografia