

**Uniwersytet Warszawski**  
Wydział Matematyki, Informatyki i Mechaniki

**Michał Borkowski**

Nr albumu: 370727  
**Marian Dziubiak**

Nr albumu: 370784

**Jakub Bujak**

Nr albumu: 370737  
**Marek Puzyna**

Nr albumu: 371359

# Kompilacja NianioLanga do efektywnych struktur języka C

Praca licencjacka  
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem  
**mgr. Radosława Bartosiaka**  
Instytut TODO

Maj 2018

## **Oświadczenie kierującego pracą**

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data

Podpis kierującego pracą

## **Oświadczenie autora (autorów) pracy**

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data

Podpisy autorów pracy

## **Streszczenie**

NianioLang jest językiem programowania ogólnego przeznaczenia, który można skompilować do wykonania na kilku platformach. Między innymi są to Java, JavaScript i C. Ze względu na chęć uproszczenia kompilacji NianioLanga utworzono środowisko uruchomieniowe przedstawiające odpowiednie abstrakcje, pozwalające na utworzenie dynamicznych struktur odpowiadających typom, jakie są dostępne do użycia w NianioLangu. Takie rozwiązanie nie jest niestety optymalne, szczególnie w przypadku niskopoziomowego języka jakim jest C. W tej pracy opisujemy wprowadzenie nowych typów i ich wsparcia w kompilatorze, które pozwolą na generowanie natywnego kodu w C, co znacznie zwiększy wydajność kompilowanych aplikacji.

## **Słowa kluczowe**

kompilacja, języki programowania, analiza semantyczna, NianioLang, system typów

## **Dziedzina pracy (kody wg programu Socrates-Erasmus)**

11.3 Informatyka

## **Klasyfikacja tematyczna**

D. Software

D.3. Programming languages

D.3.3. Language constructs and features



# Spis treści

<b>Wprowadzenie</b>	5
<b>1. Wstęp</b>	7
1.1. Czym jest NianioLang?	7
1.2. Typy w NianioLangu	7
1.3. Cele projektu	7
<b>2. Metodyka pracy</b>	9
2.1. Korzystanie z systemu kontroli wersji	9
2.2. Zgłaszanie zmian i code review	9
2.3. Techniki komunikacji w zespole	9
<b>3. Kompilator NianioLanga</b>	11
3.1. Budowanie drzewa AST	11
3.2. Analiza semantyczna	11
3.3. Architektura nlasma	11
3.4. Translacja drzewa AST do nlasma	11
3.5. Generowanie kodu C na podstawie nlasma	11
3.6. Implementacja typów NianioLanga w C	11
<b>4. Rozszerzenie systemu typów</b>	13
4.1. Rozdzielenie typu <code>ptd::sim</code>	13
4.2. Typy <code>own</code>	13
<b>5. Rozszerzenie nlasma</b>	15
5.1. Przekazywanie informacji o typach z drzewa AST	15
5.2. Statyczne sprawdzanie poprawności	15
<b>6. Nowe implementacje typów</b>	17
6.1. Typy proste	17
6.2. Tablice	17
6.3. Rekordy	17
6.4. Typy wariantowe	17
<b>7. Efekty optymalizacji i wnioski</b>	19
7.1. Porównanie czasu wykonania programów	19
<b>8. Wkład własny</b>	21
<b>Bibliografia</b>	23



# Wprowadzenie

*O tym że jest to praca licencjacka, o tym kto nam to zadanie zlecił, itp.*





# Rozdział 1

## Wstęp

1.1. Czym jest NianioLang?

1.2. Typy w NianioLangu

1.3. Cele projektu



## Rozdział 2

# Metodyka pracy

2.1. Korzystanie z systemu kontroli wersji

2.2. Zgłaszanie zmian i code review

2.3. Techniki komunikacji w zespole



## Rozdział 3

# Kompilator NianioLanga

### 3.1. Budowanie drzewa AST

### 3.2. Analiza semantyczna

### 3.3. Architektura nlasma

*Rejestry, wywołania funkcji, deklaracje typów, itp.*

### 3.4. Translacja drzewa AST do nlasma

### 3.5. Generowanie kodu C na podstawie nlasma

### 3.6. Implementacja typów NianioLanga w C



## Rozdział 4

# Rozszerzenie systemu typów

### 4.1. Rozdzielenie typu `ptd::sim`

### 4.2. Typy `own`

*Jaki jest cel tych typów, ich semantyka, jakie ograniczenia na ich użycie nakładamy (w stosunku do typów `ptd`).*





## Rozdział 5

# Rozszerzenie nlasma

5.1. Przekazywanie informacji o typach z drzewa AST

5.2. Statyczne sprawdzanie poprawności



## Rozdział 6

# Nowe implementacje typów

*W tej sekcji w każdym podrozdziale będziemy opisywać dlaczego dotychczasowe rozwiązanie było nieefektywne, jak można je było poprawić, które rozwiązanie wybraliśmy, dlaczego.*

### 6.1. Typy proste

### 6.2. Tablice

### 6.3. Rekordy

### 6.4. Typy wariantowe



## Rozdział 7

# Efekty optymalizacji i wnioski

### 7.1. Porównanie czasu wykonania programów



## Rozdział 8

# Wkład własny

*Co zrobiliśmy w rozbiu na osoby*





# Bibliografia

- [Bea65] Juliusz Beaman, *Morbidity of the Jolly function*, *Mathematica Absurdica*, 117 (1965) 338–9.