УДК 004.4

КОРОТКИЙ СИНТАКСИС ДЛЯ АНОНІМНИХ ФУНКЦІЙ В ДІАЛЕКТІ МОВИ ПРОГРАМУВАННЯ GO

Форкерт П.П. [1]; Іванченко М.Г. к.т.н., доц.[2]
[1, 2] Дніпровський національний університет імені Олеся Гончара
[1,2] Україна, Дніпро
[1] fxposter@gmail.com; [2] sidorova.m.g@gmail.com

Анотація. Стаття пропонує короткий синтаксис для анонімних функцій у діалекті GoNext, сумісному з Go, щоб зменшити багатослівність під час використання функцій вищого порядку. Описано розширення граматики мови та підхід транспіляції цих літералів у стандартні анонімні функції Go.
Ключові слова: *програмування; компіляція; синтаксис; транспіляція; Go.*

Вступна частина. Багато сучасних мов програмування надають можливість визначати анонімні функції та замикання [1]: C++, Java, JavaScript, C#, Python, Ruby, Rust тощо. Деякі мови з часом отримали додатковий синтаксис для цього, який є більш коротким, лаконічним та містить менше шаблонних елементів. Наприклад, Java отримала лямбда-вирази в Java 8 [2], які є синтаксичним цукром для створення внутрішніх класів, а JavaScript додав стрілкові функції [3]. Зазвичай цей синтаксис використовується в тих місцях, де розробникам потрібно описувати короткі однорядкові функції як аргументи для інших функцій.

Go наразі не має такого «синтаксичного цукру» для анонімних функцій, тому використання функцій вищого порядку стає досить багатослівним:

```
func filter(numbers []int, f func(int) bool) {
    // ...
}
func main() {
    numbers := []int{1, 2, 3}
    even := filter(numbers, func(n int) bool { return n % 2 == 0 })
    // ...
}
```

В репозиторії Go на GitHub є досить популярне обговорення, у якому запропоновано додавання коротких літералів функцій у цю мову програмування. Це обговорення було створено ще у 2017 році та наразі остаточного рішення щодо додавання цієї функціональності у Go немає [4].

Ця стаття зосереджена на проблемах та можливостях реалізації синтаксису коротких анонімних функцій у діалекті мови програмування Go під назвою GoNext, який має на меті бути надмножиною Go з додатковими функціями, водночас маючи можливість легкої і ефективної взаємодії з існуючими бібліотеками Go [5-6].

Основна частина. В обговоренні існуючої пропозиції [4] пропонувалося багато різних синтаксисів, але більшість із них зосереджені на одній конкретній частині визначення функції – виключенні типів аргументів із місць виклику. Один із запропонованих синтаксисів, який це робить:

```
(argument1, argument2, ...) => { body }
```

замість

```
func(argument1 type1, argument2 type2, ...) { body }
```

Отже, наведений вище приклад виглядатиме так:

```
func main() {
    numbers := []int{1, 2, 3}
```

```
    even := filter(numbers, (n) { return n % 2 == 0 })
    // ...
}
```

З точки зору парсингу, специфікацію Go [7] потрібно скоригувати, додавши `ShortFunctionLit` та налаштувавши визначення `Literal`:

```
ShortFunctionLit = "(" [ IdentifierList ] ")" "=>" FunctionBody .
Literal = BasicLit | CompositeLit | FunctionLit | ShortFunctionLit .
```

Наступний крок – визначити, як транспілювати ці короткі літерали у звичайний код Go. На відміну від додавання повнофункціональних перерахувань [8-9], неможливо просто конвертувати абстрактне синтаксичне дерево (AST) GoNext у Go AST, оскільки в короткому літералі функції відсутні типи, які мають бути виведені із зовнішнього контексту. У випадку з прикладом, наведеним вище, цей контекст є оголошенням функції `filter`.

Таким чином короткий літерал функції:

```
(n) => { return n % 2 == 0 }
```

транспілюється у:

```
func(n __1__) __2__ { return n % 2 == 0 }
```

Де `__1__` і `__2__` це місця для типів, які потрібно заповнити при генерації коду на Go. Цей літерал необхідно унифікувати в додатковому проході транспілятора з типом параметру функції `filter`: `func(int) bool` та перетворити на:

```
func(n int) bool { return n % 2 == 0 }
```

Це можливо зробити лише після виконання хоча б часткової перевірки типів, оскільки визначення функції `filter` може бути приблизно таким:

```
type filterFunc func(int) bool

func filter(numbers []int, f filterFunc) {
    // ...
}
```

і уніфікація має враховувати такі типи функцій, як `filterFunc`, щоб зрозуміти, як повністю транспілювати короткий синтаксис у той, що вже підтримується мовою Go.

До цього моменту обговорювалися лише непараметризовані функції. Основна причина цього полягає в тому, що повна підтримка синтаксису коротких літералів для параметризованих функцій потенційно може вимагати виведення типу значення, яке функція повертає, з тіла самої функції та її аргументів, без явної вказівки типу значення, що повертається. Найпростішим прикладом цього є узагальнена функція `map`:

```
func map[A any, B any](values []A, f func(A) B) []B {
    // ...
}
func main() {
    numbers := []int{1, 2, 3}
    squared := map(numbers, (n) => { return n * n })
    // ...
}
```

Якщо ми спробуємо виконати транспіляцію та виведення типів крок за кроком, це виглядатиме так:

```
(n) => { return n * n }
```

що транспілюється в

```
func(n __1__) __2__ { return n * n }
```

і після уніфікації з func(A) B де A це int (що можна вивести при перевірці типів функції) цей вираз перетворюється на

```
func(n int) __2__ { return n * n }
```

Це визначення функції, у якій не вказується тип повернення, замість цього компілятору потрібно автоматично вивести його. Але Go забороняє подібні функції, оскільки немає простого способу визначити тип повернення цієї функції в загальному випадку, наприклад, якщо вона має кілька повернень зі значеннями різних типів у своєму тілі.

Отже, з практичної точки зору, хоча технічно можливо визначити правила для виведення типу повернення в таких випадках, доцільно заборонити використання коротких літералів функцій у тих місцях, де очікуються параметризовані функції.

Висновки. У цій статті показано, як можна використовувати короткий синтаксис для анонімних функцій шляхом їх транспіляції в існуючий код Go. І хоча для універсальних функцій вищого порядку цей синтаксис важко транспілювати, його все ж можна використовувати для непараметризованого коду та дозволити розробникам насолоджуватися перевагами, які надає подібний короткий синтаксис.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] Wikimedia Foundation. (2025, October 27). *Anonymous function*. https://en.wikipedia.org/wiki/Anonymous_function

[2] Oracle. (n.d.). *Lambda Expressions*. https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html

[3] Mozilla Developer Network. (n.d.). *Arrow function expressions*. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions

[4] Github. (2017, August 17). *Go. proposal: spec: short function literals*. https://github.com/golang/go/issues/21498

[5] Forkert, P. P. & Sydorova, M. G. (2023). *Advantages of Golang as a foundation for new programming languages*, матеріали XXI міжнародної науково-практичної конференції «Математичне та програмне забезпечення інтелектуальних систем (МПЗІС-2023)», 7–8.

[6] Forkert, P. P., & Sydorova, M. G. (2023). *Challenges of using Golang as a foundation for new programming languages*, *матеріали VI Всеукраїнської науково-практичної інтернет-конференції студентів, аспірантів та молодих вчених «Сучасні інформаційні системи та технології» за тематикою: «Сучасні комп'ютерні системи та мережі в управлінні»*, 55–56.

[7] Go. (2025, August 12). The Go Programming Language Specification. Language version go1.25. https://go.dev/ref/spec

[8] Forkert, P. P. & Sydorova, M. G. (2023). Integrating full-featured enums into Go programming language. *Актуальні проблеми автоматизації та інформаційних технологій, (27)*, 3-16. doi:http://dx.doi.org/10.15421/432301

[9] Forkert, P. P., & Sydorova, M. G. (2024). *Improving enums in Go programming language dialect*, VI Міжнародна науково-практична конференція молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології (SOFT TECH-2024)», 148-150.

Forkert P.P., Ivanchenko M.G.

Short syntax for anonymous functions in Go programming language dialect

Abstract. *The article proposes a concise syntax for anonymous functions in the Go-compatible GoNext dialect to reduce verbosity when working with higher-order functions. It describes an extension to the language grammar and an approach for transpiling these literals into standard Go anonymous functions.*

Keywords: *programming, compilation, syntax, transpilation, Go.*