Překladač jazyka Zyba

Staticky typovaný jazyk kompilovaný do PHP

Zyba language compiler

Language with static typing transpiled into PHP

Středoškolská odborná činnost, rok 2022
Richard Blažek
Gymnázium Brno, třída Kapitána Jaroše 14

Prohlášení

Prohlašuji, že jsem svou závěrečnou maturitní práci vypracoval samostatně a použil jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

Prohlašuji, že tištěná verze a elektronická verze závěrečné maturitní práce jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Brně dne 15. ledna 2022 .....

Poděkování

Tímto bych chtěl poděkovat Matěji Žáčkovi za odborné vedení práce.

Anotace

Práce se zabývá navržením jazyka Zyba a implementací překladače tohoto jazyka do PHP, což by mělo umožnit používání tohoto jazyka na všech webhostinzích, které podporují PHP skripty. Rovněž bude možné vyvíjet část projektu v Zybě a část v PHP a používat funkce z jednoho jazyka ve druhém.

Klíčová slova

programovací jazyk; překladač; transpiling; webová aplikace; PHP; Zyba

Annotation

The thesis is concerned with the design of the Zyba language and implementing its compiler. The compiler generates PHP code, making it possible for the language to be used on all webhostings where PHP is supported. Also, it will be possible to combine Zyba and PHP when developing web applications and use functions from one of the languages in the other one.

Keywords

programming lanugage; compiler; transpiling; web application; PHP; Zyba

Obsah

1 Úvod

Cílem této práce je navrhnout programovací jazyk Zyba, který by měl umožnit psaní přehlednějšího kódu než PHP, a implementovat překladač ze Zyby do PHP, aby bylo možné programy v Zybě používat na všech na všech serverech s podporou PHP skriptů. Navrhl jsem Zybu jako staticky typovaný jazyk, protože překlad z jednoho dynamicky typovaného jazyka do druhého by umožnil provádět při překladu pouze syntatickou kontrolu. K vyhodnocení typů by mohlo dojít až za běhu programu a Zyba by tak představovala jen alternativní syntaxi pro PHP. Zybu jsem navrhl jako jazyk jednoduchý na naučení, ale s dostatečnou funkcionalitou pro psaní webových stránek.

Překladač jsem se rozhodl napsat v jazyce Haskell, protože umožňuje psát velmi stručné a přehledné programy. K čitelnosti programu přispívá jednak syntaxe jazyka, v němž se struktura programu vyjadřuje formátováním zdrojového kódu a ne oddělovači, jednak jeho striktní dodržování funkcionálního paradigmatu, které vyžaduje, aby funkce byly referenčně transparentní (tzn. bez vedlejších efektů). Navíc jeho typový systém obsahuje algebraické datové typy a umožňuje zápis rekurzivních typů, což se u překladače hodí například na zápis syntaktického stromu.

2 Návrh jazyka

2.1 Syntaxe

Syntaxe Zyby je stejně jako u řady dalších jazyků (např. C++, Java, C#, JavaScript) odvozená z jazyka C, aby byla blízká ostatním programátorům, v řadě detailů je však odlišná. Všechny operátory jsou binární, zleva asociativní a mají stejnou prioritu. Práce s poli a slovníky (tj. jejich vytváření a přístup k prvkům a jejich změna) není záležitostí speciální syntaxe, ale provádí se zabudovanými metodami až na úrovni sémantiky. Volání funkce je vyjádřeno hranatými závorkami. Díky těmto změnám je syntaxe jednoznačná i bez použití středníků pro ukončení příkazů, takže středníky jsou v Zybě používány na uvozování jednořádkových komentářů.

Soubor v Zybě se skládá z deklarací. Každá deklarace buď přiřadí určité globální konstantě hodnotu určitého výrazu, nebo importuje konstanty deklarované v jiném souboru.

⟨*file*⟩ ::= ⟨*declaration*⟩*

⟨*declaration*⟩ ::= ⟨*name*⟩ ⟨*expression*⟩ | ⟨*import*⟩

Importovat lze jak ze Zyby tak i z PHP. Import ze Zyby vyžaduje cestu k souboru, který má být importován, a název jmenného prostoru, do kterého budou importované hodnoty spadat. Import z PHP to vyžaduje rovněž, ale před názvem jmenného prostoru musí být napsáno slovo `php` a za cestou k souboru musí následovat záznam se jmény a typy importovaných hodnot. Záznam tvoří složené závorky a v nich několik dvojic názvů (tj. jména importovaných hodnot) a výrazů (tj. jejich typy).

⟨*import*⟩ ::= ⟨*import-zyba*⟩ | ⟨*import-php*⟩

⟨*import-zyba*⟩ ::= `"import"` ⟨*name*⟩ ⟨*literal-string*⟩

⟨*import-php*⟩ ::= `"import php"` ⟨*name*⟩ ⟨*literal-string*⟩ ⟨*record*⟩

⟨*record*⟩ ::= `"{"` (⟨*name*⟩ ⟨*expression*⟩)* `"}"`

Výraz tvoří jeden či více podvýrazů oddělených binárními operátory. Podvýraz tvoří jednotka, kterou můžou následovat volání funkce a přístupy k prvkům záznamu nebo jmenného prostoru. Volání funkce tvoří několik výrazů v hranatých závorkách; přístup k prvku tvoří tečka následovaná jménem. Zabudované metody, které budou podrobněji popsány později, se volají způsobem, který kombinuje syntaxi přístupu a syntaxi volání funkce, ale z hlediska gramatiky se nejedná o zvláštní případ.

⟨*expression*⟩ ::= ⟨*call*⟩ (⟨*operator*⟩ ⟨*call*⟩)*

⟨*subexpression*⟩ ::= ⟨*unit*⟩ (⟨*call*⟩ | ⟨*access*⟩)*

⟨*call*⟩ ::= `"["` ⟨*expression*⟩* `"]"`

⟨*access*⟩ ::= `"."` ⟨*name*⟩

Jednotek je několik druhů: výraz v závorkách; literál celého čísla, reálného čísla, logické hodnoty či textu; záznam nebo lambda funkce.

⟨*unit*⟩ ::= `"("` ⟨*expression*⟩ `")"`

⟨*unit*⟩ ::= ⟨*literal-int*⟩ | ⟨*literal-real*⟩ | ⟨*literal-bool*⟩ | ⟨*literal-text*⟩

⟨*unit*⟩ ::= ⟨*record*⟩

⟨*unit*⟩ ::= ⟨*lambda*⟩

Lambda funkce začíná slovem `fun`. Následují argumenty funkce, výraz specifikující typ navrácené hodnoty a blok kódu, který tvoří několik příkazů ve složených závorkách. Argumenty funkce se zapisují v hranatých závorkách.

⟨*lambda*⟩ ::= `"fun"` ⟨*arguments*⟩ ⟨*expression*⟩ ⟨*block*⟩

⟨*arguments*⟩ ::= `"["` ⟨*argument-group*⟩* `"]"`

⟨*argument-group*⟩ ::= ⟨*name*⟩+ `":"` ⟨*expression*⟩

⟨*block*⟩ ::= `"{"` ⟨*statement*⟩* `"}"`

⟨*statement*⟩ ::= ⟨*expression*⟩ | ⟨*assignment*⟩ | ⟨*if*⟩ | ⟨*while*⟩

⟨*assignment*⟩ ::= ⟨*name*⟩ `"="` ⟨*expression*⟩

⟨*if*⟩ ::= `"if"` ⟨*expression*⟩ ⟨*block*⟩ (`"else if"` ⟨*block*⟩)* (`"else"` ⟨*block*⟩)?

⟨*while*⟩ ::= `"while"` ⟨*expression*⟩ ⟨*block*⟩

3 Závěr