

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 16. ledna 2022

Poděkování

Tímto bych chtěl poděkovat Matěji Žáčkovi za odborné vedení práce.

Anotace

Práce se zabývá návrhem 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	5
2	Návrh jazyka	5
2.1	Syntaxe	5
2.2	Typový systém	8
2.2.1	Datové typy	8
2.3	Jmenné prostory	8
2.4	Zabudované metody	8
3	Konstrukce překladače	8
4	Závěr	8

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 syntaktickou 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, ale liší se v řadě detailů. 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, aniž by musel uživatel používat středníky. Středníky a čárky jsou považovány za bílé znaky, takže je možné je používat, kde to programátor uzná za vhodné pro přehlednost kódu. Komentáře jsou jednořádkové a začínají křížkem (#).

Následuje popis prvků syntaxe a její formální popis v rozšířené Backus-Naurově formě (EBNF).

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

$$\langle file \rangle ::= \langle declaration \rangle^*$$
$$\langle declaration \rangle ::= \langle name \rangle \langle expression \rangle \mid \langle import \rangle$$

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).

$$\begin{aligned}\langle import \rangle &::= \langle import\text{-}zyba \rangle \mid \langle import\text{-}php \rangle \\ \langle import\text{-}zyba \rangle &::= \text{"import"} \langle name \rangle \langle literal\text{-}text \rangle \\ \langle import\text{-}php \rangle &::= \text{"import php"} \langle name \rangle \langle literal\text{-}text \rangle \langle record \rangle \\ \langle record \rangle &::= \text{"{"} (\langle name \rangle \langle expression \rangle)^* \text{"}"}\end{aligned}$$

Výraz tvoří jeden či více podvýrazů oddělených binárními operátory. Podvýraz tvoří jednotka, kterou můžou následovat volání funkce a přístupy k prvkům záznamu nebo jmenového 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.

$$\begin{aligned}\langle expression \rangle &::= \langle call \rangle (\langle operator \rangle \langle call \rangle)^* \\ \langle subexpression \rangle &::= \langle unit \rangle (\langle call \rangle \mid \langle access \rangle)^* \\ \langle call \rangle &::= \text{"["} \langle expression \rangle^* \text{"}" } \\ \langle access \rangle &::= \text{"."} \langle name \rangle\end{aligned}$$

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

$$\begin{aligned}\langle unit \rangle &::= \text{"("} \langle expression \rangle \text{")" } \\ \langle unit \rangle &::= \langle literal\text{-}int \rangle \mid \langle literal\text{-}real \rangle \mid \langle literal\text{-}bool \rangle \mid \langle literal\text{-}text \rangle \\ \langle unit \rangle &::= \langle record \rangle \\ \langle unit \rangle &::= \langle lambda \rangle\end{aligned}$$

Lambda funkce začíná slovem **fun**. Následují argumenty funkce v hranatých závorkách, výraz specifikující typ navrácené hodnoty a blok kódu, který tvoří několik příkazů ve složených závorkách. Argumenty funkce, pokud nějaké jsou, se zapisují po skupinách, z nichž každá se skládá z názvů jednoho či více argumentů následovaných dvojtečkou a výrazem specifikujícím typ těchto argumentů.

$$\begin{aligned}\langle lambda \rangle &::= \text{"fun"} \text{"["} \langle arguments \rangle \text{"}" } \langle expression \rangle \langle block \rangle \\ \langle arguments \rangle &::= (\langle name \rangle^+ \text{":"} \langle expression \rangle)^* \\ \langle block \rangle &::= \text{"{"} \langle statement \rangle^* \text{"}" }\end{aligned}$$

Příkazem může být výraz, jenž má být vyhodnocen, přiřazení, příkaz `if` nebo cyklus `while`. Přiřazení se zapíše jako jméno proměnné, do níž má být přiřazeno, znak rovná se a výraz, jehož hodnota bude přiřazena. Příkaz `if` začíná slovem `if` následovaným podmínkou a blokem kódu; za blokem můžou následovat další části, jež začínají slovy `else if` a rovněž pokračují podmínkou a blokem; na konci podmínky může být slovo `else` následované blokem, který se provede, pokud žádná z předchozích podmínek nebyla splněna. Cyklus `while` začíná slovem `while` následovaným podmínkou a blokem kódu. Podmínka je výraz, který vrací logickou hodnotu.

$$\langle \textit{statement} \rangle ::= \langle \textit{expression} \rangle \mid \langle \textit{assignment} \rangle \mid \langle \textit{if} \rangle \mid \langle \textit{while} \rangle$$

$$\langle \textit{assignment} \rangle ::= \langle \textit{name} \rangle "=" \langle \textit{expression} \rangle$$

$$\langle \textit{if} \rangle ::= "if" \langle \textit{condition} \rangle \langle \textit{block} \rangle ("else\ if" \langle \textit{condition} \rangle \langle \textit{block} \rangle)^* ("else" \langle \textit{block} \rangle)?$$

$$\langle \textit{while} \rangle ::= "while" \langle \textit{condition} \rangle \langle \textit{block} \rangle$$

$$\langle \textit{condition} \rangle ::= \langle \textit{expression} \rangle$$

Nejmenší jednotkou syntaxe jsou tokeny. Token může být jedno z následujících:

- **Jméno** je posloupnost znaků, jež obsahuje pouze číslice, malá a velká písmena latinky a podtržítka a jejíž první znak není číslice.
- **Literál celého čísla** je buď posloupnost dekadických číslic tvořící číslo, nebo posloupnost dekadických číslic, udávající základ číselné soustavy, a po ní znak `r` a posloupnost číslic z oné číselné soustavy, písmena `A` až `Z` bez ohledu na velikost se považují za číslice 10 až 35. Lze používat i soustavy o základu vyšším než 36, ale není v nich možné zapsat všechna čísla protože pro ně nejsou další číslice.
- **Literál reálného čísla** začíná jako literál celého čísla, ale pokračuje desetinnou tečkou po které můžou následovat další číslice. Rovněž lze používat libovolnou číselnou soustavu.
- **Literál logické hodnoty** je `1b` pro pravdu a `0b` pro nepravdu. Je to totiž kratší než tradiční `true` a `false` a navíc je zápis obou hodnot stejně dlouhý.
- **Literál textu** je posloupnost znaků v uvozovkách. Patří do něj všechny znaky mezi nimi, tedy i konce řádků a speciální znaky, ale je-li potřeba zapsat znak uvozovky, musí se zdvojit, aby nedošlo k ukončení literálu.
- **Operátor** je libovolná posloupnost znaků „`+ - * / % & | ~ ^ < > = !`“. Operátory jsou zabudované metody, takže jestli dotýčný operátor skutečně něco znamená se posoudí až při vyhodnocování zabudovaných metod.
- **Oddělovač** je jeden znak; některý ze znaků „`() [] {} . : “`.

2.2 Typový systém

Zyba je staticky typovaný jazyk se silnou typovou kontrolou; typy všech hodnot tedy musí být známy v čase překlady a nemůžou se implicitně konvertovat. Pro konverzi hodnoty z jednoho typu na druhý lze použít zabudované metody, pokud je programátor nechce převádět ručně. Určitou zvláštností Zyby je to, že v místě, kde je očekáván typ, se dá použít libovolná hodnota onoho typu (např. hodnota `3+4` se dá použít jako typ `int`). Jména `int`, `bool` a další tak ve skutečnosti znamenají pouze výchozí hodnoty těchto typů, nikoliv typy samotné (např. `int` je totéž jako `0`). Samozřejmě není vhodné takovým způsobem psát kód, protože by to bylo matoucí, ale ta možnost tu je. Má to ale i některé výhody, protože to činí jazyk jednodušším a navíc to umožňuje vytvářet typové aliasy úplně stejně jako se vytváří proměnné a konstanty. Ostatně tento přístup má určitou paralelu v chování některých objektově orientovaných jazyků, které umožňují volat statické metody jak na třídě, tak i na objektu; i v nich se tedy dá někdy použít hodnota (objekt) na místo typu (třídy).

2.2.1 Datové typy

Název	Popis	Výchozí hodnota
<code>int</code>	Celé číslo	Nula (<code>0</code>)
<code>real</code>	Reálné číslo	Nula jako reálné číslo (<code>0.0</code>)
<code>bool</code>	Logická hodnota	Nepravda (<code>0b</code>)
<code>text</code>	Textový řetězec	Prázdný text (<code>""</code>)
<code>T.list</code>	Pole hodnot typu <code>T</code>	Prázdné pole
<code>K.dict[V]</code>	Asociativní pole s klíči typu <code>K</code> a hodnotami typu <code>V</code>	Prázdné asociativní pole
<code>R.fun[T₀ ... T_m]</code>	Funkce přebírající argumenty typů <code>T₀</code> až <code>T_m</code> a vracující <code>R</code>	Funkce přebírající libovolné argumenty typů <code>T₀</code> až <code>T_m</code> vracující vždy výchozí hodnotu typu <code>R</code>
<code>{N₀ T₀ ... N_m T_m}</code>	Záznam s prvky, jejichž názvy jsou <code>N₀</code> až <code>N_m</code> a typy jsou <code>T₀</code> až <code>T_m</code>	Záznam se všemi prvky nabývajícími svých výchozích hodnot

2.3 Jmenné prostory

2.4 Zabudované metody

3 Konstrukce překladače

4 Závěr