Streszczenie pracy

Celem niniejszej pracy dyplomowej... W skrócie: Stworzyć kompilator który zamieni JavaScript na IL Assembler i będzie można go uruchomić na .NET. Następnie porównanie działania skryptów uruchamianych na maszynach Node.js oraz .NET. Można jeszcze porównać program napisany w C# z programem napisanym w js.

Abstract

The aim of this diploma thesis was...

Spis treści

1 Wstęp

W branży programistycznej istnieje bardzo dużo języków programowania oraz różnych środowisk uruchomieniowych dla nich przeznaczonych. Podczas tworzenia oprogramowania niezbędny jest dla programisty kompilator. Zamienia on kod programu napisanego w konkretnym języku programowania, na zrozumiały dla środowiska uruchomieniowego ciąg instrukcji. Warto podkreślić, że języki programowania, które są proste do zrozumienia i opanowania dla programistów oraz pozwalają na pisanie programów, które można uruchomić na różnych urządzeniach i umożliwiają tworzenie bardzo zróżnicowanych rodzajów oprogramowania, są o wiele częściej używane niż inne. Powoduje to, że powstaje dużo różnych bibliotek i frameworków ułatwiających i automatyzujących tworzenie oprogramowania.

Jednym z popularnych języków wśród programistów jest język JavaScript, który może być uruchomiany w przeglądarkach internetowych lub w maszynie wirtualnej takiej jak Node.js. Innym z popularnych środowisk uruchomieniowych jest platforma .NET, dla której istnieje wiele kompilatorów różnych języków. Wykorzystanie istniejących bibliotek, frameworków czy modułów napisanych w języku JavaScript w niezmienionej formie na platformie .NET, umożliwi programistom na tworzenie bardziej uniwersalnego kodu.

Celem niniejszej pracy jest **zaprojektowanie** oraz **implementacja kompilatora** języka **JavaScript** na kod **IL Assembler** uruchamiany na **platformie .NET**. Następnie w celu sprawdzenia poprawności działania kompilatora, zostaną przeprowadzone testy przy pomocy prostych implementacji kodu JavaScript. Zostaną również zaprojektowane oraz zaimplementowane dwa bardziej skomplikowane testy, do **porównania maszyn wirtualnych Node.js oraz .NET**. Zostanie także porównany kod assemblera generowanego przez kompilator .Net Core języka C# z kodem generowanym przez implementowany kompilator.

Praca podzielona jest na cztery części. Pierwsza część opisuje pojęcia i technologie wykorzystywane do realizacji tego projektu oraz technologie pokrewne, które w pewnym stopniu realizują cel pracy lub realizują podobne założenia. Druga część poświęcona jest zaprojektowaniu tworzonego kompilatora. Kolejna część opisuje sposób implementacji tego kompilatora na podstawie zdefiniowanych założeń. Ostatnia część opisuje przeprowadzone testy realizowane w ramach pracy oraz przedstawia wyniki ich działania.

2 Pojęcia i technologie

2.1 Zakres projektu

Projekt będzie realizował zaprojektowanie i implementację kompilatora języka JavaScript na platformę uruchomieniową .NET core. W tym rozdziale zostaną opisane pojęcia oraz technologie, które będą wykorzystywane w realizacji tego projektu. Na początku opisane będą takie pojęcia jak *kompilator* oraz *maszyna wirtualna*. Następie zostaną omówione technologie wiodące w projekcie takie jak *JavaScript*, *Node.js*, *.NET Core* oraz *IL Assembler*.

2.1.1 Kompilator

W dzisiejszych czasach niezbędnym narzędziem programisty jest kompilator. Jest to narzędzie, którego zadaniem jest tłumaczenie programu napisanego przez programistę, na program który będzie można uruchomić na konkretnym środowisku uruchomieniowym.

Mówiąc ściślej kompilator to program napisany w języku implementacyjnym, który odczytuje język źródłowy i tłumaczy go na język wynikowy. Proces zamiany kodu źródłowego na wynikowy nazywany jest kompilacją. Kodem wynikowym procesu kompilacji może być od razu kod maszynowy, który interpretowany jest bezpośrednio przez procesor lub maszynę wirtualną, albo do kodu pośredniego, który też może zostać skompilowany przez inny kompilator.

Kompilatory mogą być napisane w dowolnym języku programowania. Istnieje kilka specjalnie zaprojektowanych do tego zadania języków, takie jak *Pascal* czy *Algol 68*. Nie mniej jednak, wybór języka do implementacji kompilatora przez twórcę, powinien się opierać na założeniu, że powinien on zminimalizować wysiłek implementacyjny i zmaksymalizować jakość kompilatora.

Język źródłowy który przetwarzany jest przez kompilator prawie zawsze jest oparty na wcześniej zdefiniowanej gramatyce. Dzięki temu program kompilatora potrafi odróżnić od siebie kolejne instrukcje i zamienić je na równoważne ciągi instrukcji w języku docelowym.[1]

2.1.2 Maszyna wirtualna

Wirtualizacja stała się ważnym narzędziem w projektowaniu systemów komputerowych, a maszyny wirtualne używane są w wielu obszarach informatycznych, od systemów operacyjnych po architektury procesorów języków programowania.

Dla programistów oraz użytkowników wirtualizacja likwiduje tradycyjne interfejsy oraz ograniczenia zasobów związanych z różnymi urządzeniami. Maszyny wirtualne zwiększają interoperacyjność oprogramowania oraz wszechstronność platformy, dlatego też często się je wykorzystuje.

Maszyna wirtualna to nic innego jak program uruchamiany na prawdziwej maszynie, który potrafi obsługiwać pożądaną architekturę. W ten sposób można obejść rzeczywistą kompatybilność maszyny i ograniczenia zasobów sprzętowych. Pozwala to, między innymi na równoczesne tworzenie oprogramowania

[1] W. M. Mckeeman. "Compiler Construction". W: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 1974, s. 1–36. ISBN: 9783540069584. DOI: 10.1007/978-3-662-21549-4_1. URL: http://link.springer.com/10.1007/978-3-662-21549-4_1, s. 1–4.

dla wielu platform, bez konieczności stosowania bezpośrednio interfejsów rzeczywistej maszyny, a jedynie wykorzystanie tych udostępnianych przez maszynę wirtualną.[2]

2.1.3 JavaScript

Opis

2.1.4 Node.js

Opis

2.1.5 .NET Core

Opis

2.1.6 IL Assembler

Opis

2.2 Technologie pokrewne

W tej sekcji będą przedstawione technologie pokrewne. Najpierw technologie konkurencyjne, następnie istniejące rozwiązania a na końcu rozwiązania podobne/nawiązujące w jakiś sposób do tematu.

2.2.1 ECMAScript

W sumie jest to standard języka na podstawie którego definiowana jest składnia JavaScript. Czy warto o tym wspominać?

2.2.2 ActionScript

Obiektowy język oparty na ECMAScript używany w Adobe Flash/

2.2.3 JScript

Jest to implementacja JavaScript przez Microsoft która jest uruchomiana w środowisku .NET. Istnieją pewne różnice w porównaniu do JavaScript.

2.2.4 TypeScript

Język programowania stworzony przez Microsoft. Uruchamiany na Node.js lub Deno. Może być przekompilowany do js es5 przez Babel.

2.2.5 ?CoffeeScript

Język programowania kompilowany do JavaScript. Nie wiem czy warto wspominać.

Inne języki kompilowane do JavaScript: Roy, Kaffeine, Clojure, Opal

[2] J.E. Smith i Ravi Nair. "The architecture of virtual machines". W: *Computer* 38.5 (2005), s. 32–38. ISSN: 0018-9162. DOI: 10.1109/MC.2005.173. URL: http://ieeexplore.ieee.org/document/1430629/.

2.2.6 Babel

Kompiluje przykładowo TypeScript do JavaScript lub JavaScript ES6 do ES5.

2.2.7 Deno

Maszyna wirtualna dla języka JavaScript oraz TypeScript.

2.2.8 asm.js

Jeśli dobrze zrozumiałem jest to okrojony JavaScript który tak samo może być uruchamiany w przeglądarkach czy Node.js/Deno. Wykorzystywany przy kompilacji kodu c++ do uruchamiania na tych maszynach.

2.2.9 Emscripten

Kompilator kodu LLVM do JavaScript.

2.2.10 WebAssembly

Język niskopoziomowy, który działa z szybkością zbliżoną do rozwiązań natywnych i pozwala na kompilację kodu napisanego w C/C++ do kodu binarnego działającego w przeglądarce internetowej. (Pomija JavaScript).

2.2.11 C#

Czy opisywać rodzinę .NET?

Na maszynę .NET istnieją implementacje języków takich jak Python, Java, C++ i inne.

2.2.12 .NET Framework

Platforma programistyczna.

2.2.13 Mono

Implementacja open source platformy .NET Framework.

2.2.14 DotGNU

Alternatywa dla Mono.

2.2.15 ?Roslyn

.NET Compiler Platform

3 Projekt kompilatora

Opis

3.1 Środowisko i narzędzia

Opis sprzętu na którym będzie wszystko uruchomiane. Do implementacji będą wykorzystane:

- C#
- Visual Studio Code
- WSL (Ubuntu 20.04 LTS)
- JavaScript
- Node.js

3.2 Analiza języka JavaScript i określenie zakresu implementacji

Opis składni JavaScript. Implementacja JavaScript w standardzie ES5. Musi być Turing-complete. Dodatkowo implementacja funkcji.

3.3 Parser

Używamy ANTLR z gotową gramatyką

Rozważane możliwości i wykonano przegląd narzędzi: Opcje: 1. Napiszę własną implementację, która może być mało czytelna i mieć dużo błędów. (Baza: http://informatyka.wroc.pl/node/391) 2. Użyję ANTLR i napiszę własną gramatykę. 3. Użyję ANTLR i użyję gotowe gramatyki.

po 2 zdania: Gotowe narzędzia:

- LEX & YYAC
- ANTLR
- Coco/R
- ?gppg & gplex
- Owl (https://github.com/ianh/owl)

i więcej... https://en.wikipedia.org/wiki/Comparison_of_parser_generators

3.4 Struktura projektu

Diagramy i opisy. Jak będzie wyglądał ten rozdział zależy jak wyjdzie implementacja.

4 Implementacja aplikacji kompilatora

4.1 Parser

Sposób implementacji lub przygotowania i użycia gotowych narzędzi.

4.2 Analiza leksykalna

Tekst

4.3 Gramatyka

Tekst

4.4 Funkcjonalności

Opis sposobu przetwarzania instrukcji

4.5 Generowanie assemblera

Opisać jak wygenerować kod .NET

5 Testy

Opis zakresu testów i jak będą przebiegać. Każdy z poniższych testów będzie sprawdzany pod kątem poprawności wykonywania działań, czasu wykonywania w porównaniu do wykonania kodu na Node.js (dla bardziej złożonych testów z czasem będzie więcej), zajętość pamięci (również tylko przy tych których to ma sens) oraz porównaniu kodu z asemblerowego wygenerowanego za pośrednictwem języka C#.

5.1 Proste operacje matematyczne

Test dodawania, odejmowania, mnożenia, dzielenia, przypisywania.

5.2 Kolejność wykonywania działań

Test na bardziej złożonych wyrażeniach. Sprawdzenie poprawności działania nawiasów oraz kolejności wykonywania działań.

5.3 Wyrażenia warunkowe

Test wyrażeń warunkowych. Kolejność wykonywania operacji and i or.

5.4 Tablice

Test obsługi tablic jedno i wielowymiarowych.

5.5 Obiekty

Test obsługi obiektów.

5.6 Klasy

Jeśli będzie implementacja. Test działania obiektów klas.

5.7 Funkcje

Test działania funkcji. 1. Funkcja "void"bez parametrów. 2. Funkcja "voidź parametrami. 3. Funkcja zwracająca różne typy (proste, tablice, obiekty) bez parametrów. 4. Funkcje zwracająca różne typy z parametrami. 5. inne

5.8 Algorytm 1

5.8.1 Opracowanie pseudokodu algorytmu 1

5.8.2 Implementacja algorytmu 1

5.8.3 Testy algorytmu 1

5.9 Algorytm 2

5.9.1 Opracowanie pseudokodu algorytmu 2

5.9.2 Implementacja algorytmu 2

5.9.3 Testy algorytmu 2

6 Podsumowanie