



УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У
НОВОМ САДУ



Марко Ердељи

Транспилација из *Python*-а у *Rust*: Студија случаја

ДИПЛОМСКИ РАД
- Основне академске студије -

Нови Сад, 2024

Садржај

1	Увод	1
2	Теоријска Позадина	2
2.1	Дефиниција и значај транспајлера	2
2.2	Структура и фазе транспилације	2
2.3	Технике коришћене у транспајлерима	4
2.4	Преглед Python-а и Rust-а: карактеристике језика и перформансе . . .	4
3	Литература	5

1 Увод

Транспилација (енгл. *transpilation*)¹ представља значајан корак у оптимизацији и унапређењу перформанси софтвера. У савременом софтверском инжењерству, употреба трансформација изворног кода у изворни (енгл. *source-to-source transformation*)² показала се као кључна за постигнуће ефикасније извршне верзије програма. Ова техника отвара пут ка дубљем разумевању структуре и перформанси софтвера, омогућавајући програмерима да директно утичу на трансформацију изворног кода према специфичним захтевима извршења [1].

Први транспајлери су развијени у касним 70-им и раним 80-им годинама прошлог века. Године 1978, Intel је предложио аутоматски преводилац кода за конверзију 8-битних програма у њихове еквивалентне 16-битне програме [2]. XLT86™ је предложен 1981. године као преводилац асемблерског језика са 8080 на 8086, са циљем да аутоматски трансформише ASM тип фајлове у A86 тип фајлове [3].

Овај рад фокусира се на транспилацију између два програмска језика високог нивоа (енгл. *high-level programming languages*)³, Python-a и Rust-a, истражујући процесе, изазове и резултате овог инжењерског задатка. Кроз детаљну анализу алата и техника које су коришћене, циљ је да се истражи како транспилација може да унапреди перформансе и одрживост софтверских решења у контексту машинског учења.

У уводном делу рада биће детаљно истражена техника транспилације, са посебним фокусом на процес транспилације Python-a у Rust. Циљ истраживања је дубље разумевање техника транспилације, као кључне за оптимизацију и унапређење перформанси софтвера. Биће анализирани аспекти Python-a и Rust-a, са фокусом на њихове предности и ограничења у контексту перформанси, безбедности и скалабилности.

Даље, истраживаће се постојећи транспајлери (енгл. *transpilers*)⁴, који се користе за Python и Rust транспилацију, са детаљном анализом алата и техника. Фокус ће бити на документацији процеса транспилације, укључујући кораке за постављање и коришћење транспајлера, као и приказивање примера кода пре и после транспилације. Посебна пажња биће посвећена студији случаја кроз примере мањих пројеката у области машинског учења, што ће омогућити дубље разумевање изазова, решења и потенцијалних унапређења.

На крају, у дискусији ће се сумирати главни налази истраживања, дискутовати предности и мане транспилације, као и дати предлози за будућа истраживања која би могла унапредити ефикасност и примену транспајлера у практичним софтверским решењима.

¹Процес превођења између програмских језика високог нивоа.

²Техника која укључује конверзију између различитих језика високог нивоа без међупроцеса са машинским кодом.

³Програмски језици високог нивоа су програмски језици који су дизајнирани да буду разумљиви људима и омогућавају апстракцију сложених операција.

⁴Алати за транспилацију кода из једног програмског језика у други.

2 Теоријска Позадина

2.1 Дефиниција и значај транспајлера

Хирцел и коаутори [4] дефинишу процес транспилације као ону у којој се софтвер пише на изворном језику, али се компајлира и извршава у различитом програмском језику. Чабер и коаутори [5] објашњавају да је транспилација метод генерисања кода у коме дође до превода из једног програмског језика високог нивоа у други програмски језик ниског нивоа. Разлози зашто је транспилација значајна:

- **Миграција (енгл. *Migration*)**⁵ : Омогућава пренос legacy кода на модерније програмске језике, чиме се унапређује одржавање и распрострањање апликација [6].
- **Компатибилност**: Генерисање кода који је компатибилан са старијим верзијама програмских језика, док развојници користе нове функционалности доступне у модернијим верзијама [7].
- **Преусмеравање вештина програмирања**: Омогућава адаптацију програмера или тимских преференција транспилацијом кода у језик који је погоднији за разумевање или вештине тима [8].
- **Побољшање перформанси**: Унапређење перформанси апликација транспилацијом делова кода у језик који има бољи компајлатор или је боље оптимизован за циљну платформу [6].

2.2 Структура и фазе транспилације

На високом нивоу, структура транспилера укључује неколико фаза које се секвенцијално извршавају. Структура транспилера је приказана на слици 1 [9].

- **Лексичка анализа (енгл. *lexical analysis*)**: Анализира ниску карактера која је представљена и дели је на токене који су легални чланови вокабулара језика у којем је програм написан.
- **Синтаксна анализа (парсирање) (енгл. *syntactic analysis/parsing*)**: Процесира секвенцу токена и производи међурепрезентацију, као што су стабло парсирања или секвенцијални међурепрезентацијски код.
- **Семантичка провера (енгл. *semantic checking*)**: Врши проверу програма за статичку-семантичку исправност, утврђујући да ли програм задовољава статичке-семантичке особине захтеване од изворног језика.

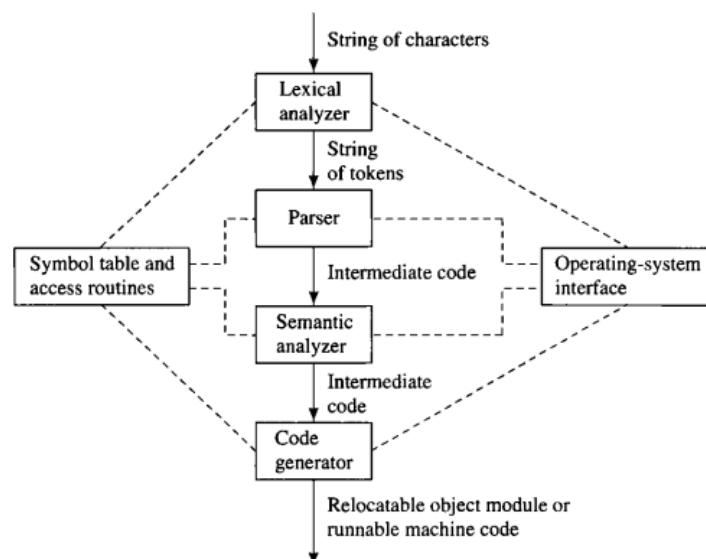
⁵Пренос старог кода на новије програмске језике.

- **Генерисање кода (енгл. *code generation*):** Трансформише међурепрезентацију у еквивалентни код у циљном језику или другом облику прилагодљивом за циљну платформу.

Транспилери могу бити имплементирани као једнопролазни или вишепролазни системи, слично као и класични компајлери. Једнопролазни транспилери пружају бржу компилацију али може бити теже постићи високу ефикасност генерисаног кода. Вишепролазни приступи омогућавају бољу оптимизацију и квалитетнији генерисани код, али уз трошак дужег времена компилације.

Као и код компајлера, након што се транспилација заврши, програми или њихови делови обично морају бити повезани (линковани) како би се међусобно повезали и са потребним библиотечким рутинама, те уčitани и премештени у меморију ради извршавања.

За многе високонивне језике, четири фазе могу бити комбиноване у један пролаз кроз изворни програм како би се произвео брз једнопролазни компајлер (енгл. *compiler*)⁶. Такав компајлер може бити сасвим одговарајућ за повремене кориснике или као алтернатива за инкременталну компилацију у окружењу развоја софтвера, где је циљ обезбедити брз циклус измена, компилације и дебаговања. Међутим, такав компајлер генерално није у могућности да произведе веома ефикасан код. Алтернативно, фазе лексичке и синтаксне анализе могу бити комбиноване у пролаз који производи табелу симбола и неки облик међурепрезентације, и семантичка провера и генерисање објектног кода из међурепрезентације могу се извршити као засебан други пролаз, или као два засебна пролаза (или семантичка провера може бити у великој мери део првог пролаза).



Слика 1: Структура традиционалног транспајлера

⁶Компајлер је програм који може да прочита програм написан на једном језику и преведе га у еквивалентни програм на другом језику. За разлику од транспајлера, који преводи програм са једног језика на други у истом нивоу апстракције [10].

2.3 Технике коришћене у транспајлерима

Транспајлери користе различите технике за постизање ефикасне и тачне трансформације кода. Неки од кључних приступа укључују:

- **Синтаксна анализа** (енгл. *syntactic analysis*): Процес препознавања и обраде синтаксних структура у изворном коду.
- **Семантичка анализа** (енгл. *semantic analysis*): Процес провере значења и исправности кода у односу на правила програмског језика.
- **Оптимизација кода** (енгл. *code optimization*): Технике за побољшање ефикасности и перформанси генерисаног кода.

2.4 Преглед Python-а и Rust-а: карактеристике језика и перформансе

Python и Rust су два програмска језика високог нивоа са значајним разликама у дизајну и перформансама. Python је познат по својој једноставности и лакоћи коришћења, што га чини популарним избором за брз развој и прототипирање. Rust, с друге стране, пружа високе перформансе и безбедност меморије, што га чини погодним за системе који захтевају поузданост и ефикасност.

- **Python:** Интерпретирани језик са динамичким типизацијама и великим бројем библиотека које подржавају различите домене примене, укључујући машинско учење, веб развој и науку о подацима.
- **Rust:** Компилирани језик са статичким типизацијама и снажним механизмима за управљање меморијом, као што су власништво и зајмови (енгл. *ownership and borrowing*), који омогућавају безбедно и брзо извршавање кода.

Овај рад истражује процес транспилације између Python-а и Rust-а, фокусирајући се на технике и алате који се користе за трансформацију кода, као и на изазове и предности које доноси овај процес.

3 Литература

- [1] D. B. Loveman, “Program Improvement by Source-to-Source Transformation,” J. ACM, vol. 24, no. 1, pp. 121–145, Jan. 1977, doi: 10.1145/321992.322000.
- [2] Intel. MCS-86 Assembly Language Converter Operating Instructions for ISIS-II Users. Technical Report. 1978. [Online]. Available: http://www.bitsavers.org/pdf/intel/ISIS_II/9800642A_MCS-86_Assembly_Language_Converter_Operating_Instructions_for_ISIS-II_Users_Mar79.pdf (приступљено 18. јула 2024).
- [3] Research, D. XLT86 8080 to 8086 Assembly Language Translator, User’s Guide. Technical Report. 1981. [Online]. Available: <http://www.s100computers.com/Software%20Folder/Assembler%20Collection/Digital%20Research%20XLT86%20Manual.pdf> (приступљено 18. јула 2024).
- [4] M. Hirzel and H. Klaeren, “Code coverage for any kind of test in any kind of transcompiled cross-platform applications,” in Proceedings of the 2nd International Workshop on User Interface Test Automation, in INTUITEST 2016. New York, NY, USA: Association for Computing Machinery, 2016, pp. 1–10. doi: 10.1145/2945404.2945405.
- [5] P. Chaber and M. Ławryńczuk, “Effectiveness of PID and DMC control algorithms automatic code generation for microcontrollers: Application to a thermal process,” 2016 3rd Conference on Control and Fault-Tolerant Systems (SysTol), Barcelona, Spain, 2016, pp. 618–623, doi: 10.1109/SYSTOL.2016.7739817.
- [6] M. Bysiek, A. Drozd, and S. Matsuoka, “Migrating Legacy Fortran to Python While Retaining Fortran-Level Performance through Transpilation and Type Hints,” 2016 6th Workshop on Python for High-Performance and Scientific Computing (PyHPC), pp. 9–18, 2016.
- [7] A. Pilsch, “Translating the Future: Transpilers and the New Temporalities of Programming in JavaScript,” 2018.
- [8] P. Misse-Chanabier, V. Aranega, G. Polito, and S. Ducasse, “Illicium A modular transpilation toolchain from Pharo to C,” in IWST19 - International Workshop on Smalltalk Technologies, Köln, Germany, Aug. 2019. [Online]. Available: <https://hal.science/hal-02297860> (приступљено 19. јула 2024).
- [9] S. Muchnick, Advanced compiler design implementation. Morgan kaufmann, 1997.
- [10] V. A. Alfred, S. L. Monica, and D. U. Jeffrey, Compilers principles, techniques & tools. pearson Education, 2007.