Synopsis de Projet

Compilateur Rust

Sany BAKIR, Bastian LEBRUN
Marcus RABOURDIN, Louis-Alexandre CHARNAY

Janvier 2025

1 Contexte

Le S4 représente pour nous une opportunité unique d'élargir nos connaissances en informatique en explorant un nouveau langage : Rust. Jusqu'à présent, nos projets académiques nous ont permis de travailler avec des langages comme Csharp et C, mais ces projets étaient entièrement guidés et imposés. Le fait que ce projet soit libre constitue un défi inédit pour nous, tant sur le plan organisationnel que technique.

Ce projet de création de compilateur, est une excellente occasion d'apprendre et de nous confronter des concepts tout à fait nouveaux pour nous tous. En effet, à ce jour, nous n'avons aucune expérience préalable en conception de compilateurs, ce qui rend ce projet d'autant plus enrichissant et stimulant.

2 Intérêt d'utiliser Rust

Bien que Rust soit imposé dans le cadre de ce projet, il est important de noter qu'il s'agit d'un langage de programmation moderne offrant de nombreux avantages. Contrairement au C, où la gestion de mémoire repose sur le développeur, ce qui peut conduire à des problèmes de sécurité comme des fuites de mémoire, Rust se distingue par son système d'ownership. Ce système permet une gestion de mémoire sécurisée et efficace, sans avoir besoin de garbage collector.

De plus, Rust est également reconnu pour sa performance comparable à celle du C ou C++, ce qui fait un choix idéal pour des projets complexes et exigeants comme la création d'un compilateur.

3 Intérêt algorithmique du projet

Comme dit précédemment la création d'un compilateur Rust en Rust est à la fois un défi technique et à la fois un défi algorithmique. Ce projet mobilise une variété de concept fondamentaux en algorithmique. Premièrement, nous avons la tokenisation, qui consiste à transformer le code source en une suite de tokens. Pour cela, nous avons besoin de développer un analyseur lexical, basé sur des algorithmes vus au cours du début du S3 tels que les automates finis déterministes (DFA) ou non déterministes.

Ces automates permettent de reconnaître les différents éléments du langage, comme les mots-clés, les opérateurs, ou encore les identifiants, en respectant les règles lexicales définies.
Ensuite, l'analyse syntaxique, ou appelé "parsing", via les tokens récupérer, s'appuie sur des algorithmes de parsing, destinés à vérifier si une séquence de tokens respecte les règles du langage Rust. Cette étape implique de manipuler des structures de données comme les arbres syntaxiques abstrait (AST), construit à l'aide de différents algorithmes de parsing.

La dernière étape est la génération du code machine. À ce stade, le rôle compilateur est de convertir le code source, écrit en Rust, en instructions assembleurs. Cette phase est essentielle car elle garantir que le code généré soit correct mais aussi optimisé pour une exécution rapide.

En résumé, ce projet met le point sur plusieurs aspects algorithmiques important dans la programmation :

- L'utilisation d'automate DFA et NFA pour la reconnaissance lexicale et syntaxique.
- L'utilisation de structures de données complexes comme les arbres syntaxiques.
- L'utilisation d'algorithmes de parcours comme le BFS et le DFS pour l'analyse et génération de code.