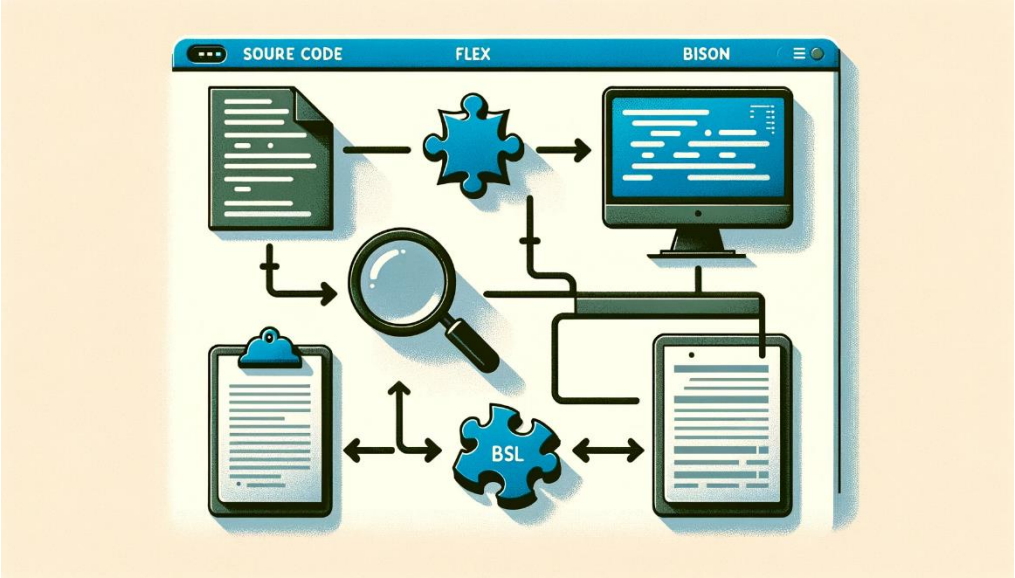
Conception et Implémentation d'un Analyseur pour tpc : Une Approche avec Flex et Bison



AOUCHER ANAIS
HNAIEN ACHRAF

Professeur : Eric LAPORTE

Module : Analyse syntaxique

—2023-2024—

1. Résumé

Ce rapport traite de la création d'un analyseur syntaxique pour le langage tpc, un langage que nous avons imaginé et qui s'inspire du C. L'ambition du projet était de développer un outil capable d'interpréter le code source tpc pour le convertir en un arbre syntaxique abstrait. Pour cela, nous avons utilisé Flex pour l'analyse des tokens et Bison pour la partie syntaxique.

Au début du rapport, nous introduisons le langage tpc en détaillant ses spécificités syntaxiques et ses types. Nous abordons ensuite les étapes clés de la construction de notre analyseur : de la définition des tokens à la conception de la grammaire, sans oublier l'intégration des outils Flex et Bison. La gestion des erreurs, un point essentiel dans tout analyseur syntaxique, a aussi été une de nos priorités.

Pour tester notre outil, nous avons élaboré des séries de tests, comprenant à la fois des programmes tpc corrects (rangés dans le dossier 'good') et des programmes erronés (dans 'sys-err'). Ces tests ont permis de vérifier l'efficacité de l'analyseur à détecter les erreurs et à construire correctement les arbres syntaxiques pour des codes valides.

En conclusion, le rapport fait le bilan des réussites et des difficultés rencontrées, et offre une réflexion personnelle sur ce que le projet nous a appris. Nous proposons également des pistes d'amélioration pour l'avenir, ouvrant la voie à de nouvelles avancées dans le domaine de l'analyse syntaxique.

Ce résumé présente de manière claire et concise notre projet, ses objectifs, les méthodes employées, les résultats obtenus et les leçons tirées. Il est adaptable selon les spécificités de votre projet et vos expériences personnelles durant son développement.

Table des matières

2. Introduction

- ## Contexte du Projet

En tant qu'étudiants en informatique, nous sommes constamment confrontés à l'évolution rapide des technologies et des langages de programmation. Notre projet s'inscrit dans ce contexte dynamique, avec un accent particulier sur le langage de programmation tpc, un langage que nous avons spécialement conçu pour cet exercice. Inspiré du langage C, le tpc offre une opportunité unique d'explorer les complexités de l'analyse syntaxique. Ce projet nous a permis d'approfondir nos connaissances sur la structure et l'analyse des langages de programmation.

- ## Objectifs et Motivations

L'objectif principal de ce projet était de développer un analyseur syntaxique pour le langage tpc. Cet outil devait être capable de lire et de transformer le code source tpc en un arbre syntaxique abstrait, facilitant ainsi la compréhension et le traitement ultérieur du code.

- ## Défis et Solutions

Nous avons rencontré plusieurs obstacles durant ce projet. Le premier était de définir précisément les règles lexicales et grammaticales du langage tpc, tout en assurant leur compatibilité avec les outils Flex et Bison. Un autre défi majeur a été la gestion des erreurs syntaxiques, un aspect crucial pour garantir la fiabilité de notre analyseur.

Pour relever ces défis, nous avons adopté une approche itérative, en commençant par des concepts simples et en augmentant progressivement la complexité. Nous avons passé beaucoup de temps à tester et à déboguer notre code, ce qui nous a permis de comprendre en profondeur le fonctionnement interne de l'analyseur syntaxique. Cette expérience nous a enseigné non seulement des compétences techniques, mais aussi l'importance de la persévérance et de l'adaptabilité dans la résolution de problèmes.

3. Description du Projet

- ## Détails du Langage tpc

Le langage tpc est conçu pour ressembler à un sous-ensemble de C, avec des structures de base telles que des fonctions, des déclarations de variables, et des instructions de contrôle de flux. Il supporte les types de données de base tels que int et char, et utilise void pour les fonctions sans retour ou sans arguments.

- ## Fonctionnalités Principales de l'Analyseur

Définition des Lexèmes : Le code définit les motifs (lexèmes) pour les différents éléments du langage tpc, tels que les identifiants, les types, les opérateurs et les mots-clés. Les lexèmes sont essentiels pour le processus d'analyse lexicale, où les chaînes de caractères du code source sont transformées en tokens significatifs.

Grammaire et Parsing : Le code utilise Bison pour définir la grammaire du langage tpc et pour construire un arbre syntaxique à partir des tokens identifiés par Flex. Cette étape est cruciale pour assurer que le code source suit les règles syntaxiques du langage.

Gestion des Erreurs : L'analyseur est capable de signaler des erreurs syntaxiques et lexicales, aidant à localiser et corriger les problèmes dans le code source.

Affichage et Gestion de l'Arbre Syntaxique : En plus de construire l'arbre syntaxique, l'analyseur peut l'afficher, notamment grâce à des fonctions pour l'impression et la suppression de l'arbre.

- ## Utilisation et Options de Ligne de Commande

L'analyseur offre différentes options de ligne de commande pour faciliter son utilisation, telles que l'affichage de l'arbre syntaxique (-t ou --tree), l'ouverture du rapport (-rep ou --rapport), et l'affichage d'un message d'aide (-h ou --help).

4. Méthodologie de Développement

- ## Approche du Projet

Ce projet, axé sur la création d'un analyseur syntaxique pour le langage tpc, a été abordé avec un souci de clarté et d'efficacité. Nous avons commencé par une compréhension des spécificités du langage tpc, puis avons élaboré un plan pour l'analyse lexicale et syntaxique.

- ## Structure du Code

La structure du code et des dossiers a été méticuleusement organisée pour faciliter le développement et la maintenance. Les dossiers principaux sont :

src : Contient les fichiers sources écrits par les développeurs, y compris les fichiers Flex (.l) et Bison (.y).

bin : Destiné au fichier binaire exécutable de l'analyseur (tpcas).

obj : Stocke les fichiers intermédiaires générés lors de la compilation.

tests : Comprend des sous-dossiers good pour les programmes tpc corrects et syn-err pour ceux avec erreurs syntaxiques.

rep : Contient le rapport de projet.

- ## Processus de Développement

Conception des Lexèmes et de la Grammaire : Nous avons défini les lexèmes et la grammaire du langage tpc, en mettant l'accent sur les exigences spécifiques telles que la prise en charge des tableaux et des opérateurs.

Écriture et Test de l'Analyseur : Après avoir écrit les scripts Flex et Bison, nous avons effectué des tests itératifs, en ajustant et en améliorant l'analyseur pour gérer différents cas de test.

Gestion des Erreurs et Affichage de l'Arbre Syntaxique : Un soin particulier a été apporté à la gestion des erreurs et à l'affichage de l'arbre syntaxique abstrait, pour garantir la précision et l'utilité des retours.

Tests Automatiques : Le script de test automatisés a été mis en place pour vérifier l'exactitude de l'analyseur, en utilisant les jeux d'essais fournis.

5. Tests et Résultats du Projet Analyseur Syntaxique tpc

Les tests étaient divisés en deux catégories : 'good' (tests positifs) et 'syn-err' (tests négatifs).

Tests 'good': Ces tests comprenaient des fichiers tpc correctement formulés, censés passer sans erreurs.

Tests 'syn-err': Ces tests étaient constitués de fichiers tpc avec des erreurs syntaxiques délibérées, attendus à échouer.

- ## Méthode de Test

Un script Python a été choisi pour exécuter les tests.

Le script parcourt les répertoires de tests, exécute l'analyseur sur chaque fichier et vérifie si le résultat correspond aux attentes (réussite ou échec).

3. Pourquoi Python Plutôt qu'un Script Shell (.sh)?

Portabilité : Python est généralement plus portable entre différents systèmes d'exploitation que les scripts shell, qui peuvent varier en syntaxe et en fonctionnalités.

Gestion des Erreurs : Python offre une gestion des erreurs plus robuste et détaillée.

Facilité de Maintenance et d’Extension : Les scripts Python sont souvent plus lisibles et plus faciles à maintenir ou étendre que les scripts shell, notamment pour des opérations complexes.

- ## Résultats Obtenus et Interprétation

Les résultats des tests sont affichés à l'écran, y compris le pourcentage de réussite pour les tests 'good' et 'bad', ainsi qu'un score global.

Le script identifie également les résultats inattendus, comme les tests 'good' qui échouent ou les tests 'bad' qui passent, ce qui est crucial pour déceler les anomalies.

L'analyse de ces résultats permet de comprendre les forces et les faiblesses de l'analyseur.

6. Problèmes et Solutions

- ## Les Défis Rencontrés

Traquer les Erreurs : C'était un peu comme jouer au détective. Trouver et gérer les erreurs lexicales et syntaxiques était difficile. Il fallait non seulement les détecter mais aussi comprendre leur origine précise.

Construire l'Arbre Abstrait : Imaginer et créer une structure d'arbre abstrait pour représenter les programmes TPC a été un vrai défi. Comment bien agencer les nœuds, les feuilles, Comment faire pour que tout soit clair et fonctionnel ?
Gestion du Temps : Un des plus grands défis a été la gestion du temps, surtout parce que nous avions plusieurs projets en parallèle. En plus, notre équipe était composée d'un alternant et d'un membre travaillant, ce qui rendait nos disponibilités assez désynchronisées.

7. Conclusion

- ## Nos Accomplissements

Développement d'un Analyseur Syntaxique Performant :

Malgré les contraintes de temps, nous avons réussi à développer un analyseur syntaxique complet pour le TPC, capable de transformer le code source en un arbre abstrait structuré.

Mise en Place d'un Système de Test Efficace :

Nous avons élaboré un système de test automatisé, garantissant ainsi la fiabilité et l'efficacité des tests de notre analyseur.

Optimisation de la Gestion d’Erreurs :

Nous avons amélioré la détection et la gestion des erreurs, rendant notre outil plus intuitif et plus facile à utiliser.

- ## Réflexions d'Équipe

Apprentissages Collectifs : ce projet a été une expérience d'apprentissage collective, nous enseignant non seulement les aspects techniques de l'analyse syntaxique mais aussi l'importance de la communication et de la gestion du temps au sein d'une équipe.

Axes d’Amélioration : à l'avenir, nous pourrions améliorer notre analyseur en intégrant des fonctionnalités pour des structures syntaxiques plus complexes. En outre, développer une interface utilisateur graphique pourrait rendre l'outil plus accessible.