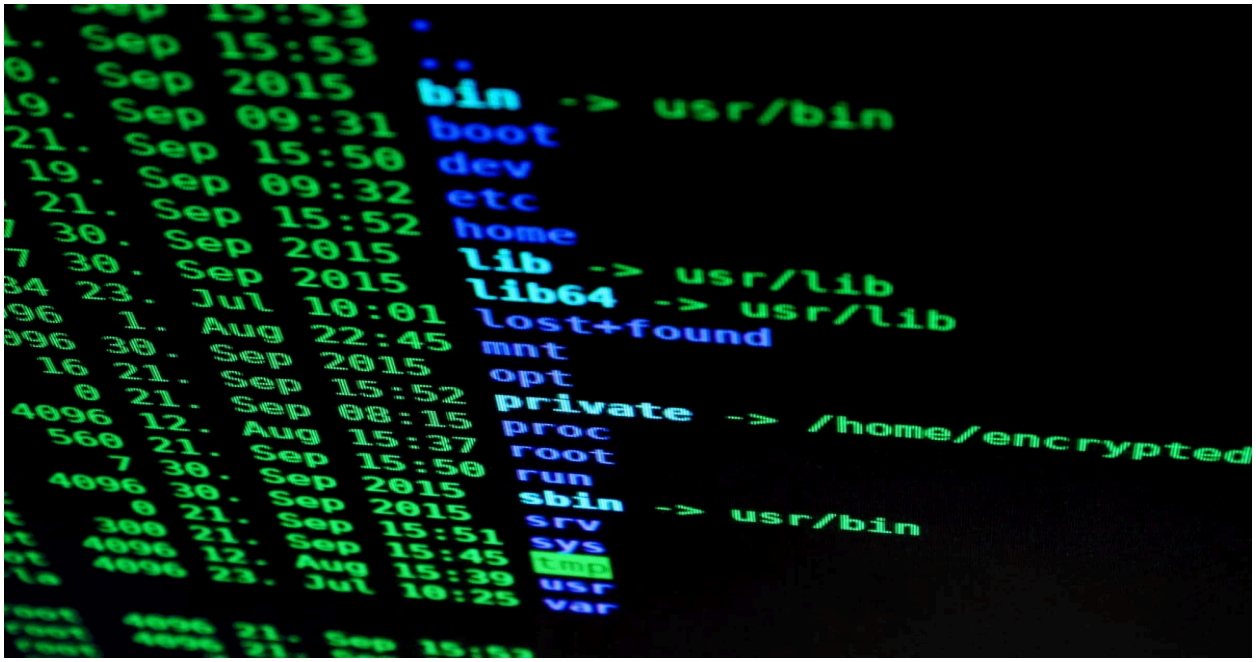

Interpréteur d'Expressions Ensemblistes

13.03.2025

Automates et théorie des langages

L3 Informatique
Université de la Polynésie Française
Groupe 6 : Lennon - Manuarii - Pauline

Vue d'ensemble

Ce projet a été réalisé dans le cadre du cours **Automates et Théorie des Langages (L3 Informatique - S6)**. Il consiste en la conception et l'implémentation d'un **interpréteur d'expressions ensemblistes** utilisant **Flex et Bison**. L'objectif est de permettre l'analyse et l'exécution d'opérations ensemblistes tout en garantissant une **gestion efficace des erreurs** et un **respect des priorités des opérateurs**.

Le développement du projet s'est fait en **trinôme**, avec des contributions spécifiques de chaque membre :

- **Lennon** : Développement de l'analyseur lexical avec Flex.
- **Pauline** : Développement de l'analyseur syntaxique avec Bison.
- **Manuarii** : Intégration de l'analyseur sémantique et exécution des expressions.

Objectifs

L'interpréteur d'expressions ensemblistes vise à :

- **Analyser et exécuter des expressions ensemblistes** définies par l'utilisateur.
- **Gérer les opérations ensemblistes** : `union`, `inter`, `comp`, `-`, `card` et l'affectation (`:=`).
- **Assurer la correction syntaxique et sémantique** des expressions avec Flex et Bison.
- **Optimiser la représentation des ensembles** en utilisant des opérations bit à bit (`unsigned long long`).
- **Éliminer automatiquement les doublons** lors de la définition des ensembles.
- **Fournir une gestion robuste des erreurs** pour guider l'utilisateur en cas de faute de syntaxe ou d'entrée invalide.

Caractéristiques

L'interpréteur intègre plusieurs fonctionnalités clés :

- **Analyse lexicale (Flex)** :

    - Détection des **identificateurs d'ensembles** (lettres A-Z, a-z).
    - Reconnaissance des **ensembles** délimités par `{}` et contenant des entiers entre **1 et 63**.
    - Gestion des **affectations** (`:=`) et des **erreurs lexicales** via `printError()`.
- **Analyse syntaxique (Bison)** :
    - Définition d'une **grammaire LL(1)** pour les opérations ensemblistes.
    - Ajout des **priorités et associativités (`%left`)** pour éviter les conflits `shift/reduce`.
    - Gestion des **expressions imbriquées** (`(A union B) inter C`).
    - Traitement des **expressions incorrectes** avec `yyerror()`.
- **Analyse sémantique et exécution** :
    - Représentation des **ensembles sous forme d'entiers 64 bits** (`unsigned long long`).
    - Suppression automatique des **doublons** dans les ensembles.
    - Gestion des **erreurs sémantiques** (ex : `A := card(A)` → erreur explicite).

Grandes étapes

- ### Développement de l'analyseur lexical (22/02/2025 - Lennon)

Travail effectué :

La première étape du projet a été la mise en place de l'**analyseur lexical**, chargé d'identifier les éléments d'une expression ensembliste et de les convertir en **tokens** exploitables par l'analyseur syntaxique et sémantique. L'objectif principal était de permettre le **scanning** d'une entrée utilisateur et de générer une séquence de tokens bien définie.

Correction d'un problème d'inclusion :

Initialement, le fichier `lexer.h` avait été inclus dans `lexer.flex` afin d'uniformiser la définition des tokens. Cependant, cela a provoqué des **erreurs de compilation**, car Flex ne reconnaissait pas correctement les déclarations externes.

- ### Implémentation de l'analyseur syntaxique (23/02/2025 - Pauline)

Travail effectué :

Après l'implémentation de l'analyseur lexical, la prochaine étape a été de concevoir un **analyseur syntaxique** capable de **valider la structure des expressions ensemblistes**. L'objectif était d'interpréter correctement **les affectations**, **les opérations ensemblistes**, et **les expressions imbriquées**, tout en détectant les erreurs syntaxiques.

Plusieurs étapes ont été nécessaires pour aboutir à un analyseur fonctionnel.

Ajout d'une **grammaire en Bison** pour structurer les expressions ensemblistes et assurer leur bonne reconnaissance. L'analyseur devait reconnaître : les affectations, les opérations ensemblistes, l'utilisation des parenthèses et le calcul de cardinalité.

Modification du Makefile: lors du développement de l'analyseur syntaxique, une modification du Makefile a été nécessaire pour prendre en compte la compilation de l'analyseur Bison et assurer une génération correcte des fichiers nécessaires (parser.c, parser.h).

Un problème est rapidement apparu : sans **priorités d'opérateurs**, l'analyseur ne savait pas dans quel ordre appliquer `union`, `inter` et `-`. Par défaut, Bison effectue un **parsing de gauche à droite**, ce qui pouvait donner des résultats incorrects.

Lors des premiers tests, plusieurs **conflits `shift/reduce`** sont apparus. Ces conflits se produisaient lorsque **Bison ne savait pas s'il devait "décaler" (shift) un token ou "réduire" (reduce) une règle existante**.

- ## Analyse sémantique et exécution (05/03/2025 - Manuarii - Pauline)

Travail effectué :

Après la mise en place de l'analyseur lexical et syntaxique, l'étape suivante consistait à **interpréter et exécuter les expressions ensemblistes**.
L'objectif principal était d'implémenter **une représentation efficace des ensembles**, d'assurer **une exécution rapide des opérations** et de **corriger les incohérences dans les expressions mal formées**.

Correction des erreurs de cardinalité et gestion des expressions mal formées :

Le calcul de la cardinalité des ensembles était incorrect, renvoyant toujours 3. L'erreur venait d'un mauvais comptage des bits actifs, faussant le résultat. La méthode a été corrigée pour assurer un comptage précis du nombre d'éléments distincts.

Suppression des erreurs liées à `TOKEN NEWLINE:` lorsque TOKEN_NEWLINE était rencontré dans certaines expressions, **l'analyse était interrompue prématurément**, ce qui empêchait l'évaluation correcte de certaines opérations imbriquées.

Solution appliquée : ajout d'une **règle explicite dans `set_interpreter.bison`** pour ignorer `TOKEN_NEWLINE` si nécessaire.

- Finalisation et tests (11/03/2025 - Pauline)

Travail effectué :

Avec l'implémentation complète des **analyseurs lexical, syntaxique et sémantique**, la dernière phase du projet a consisté à **finaliser et tester l'interpréteur**, à **stabiliser le parsing**, et à **préparer le livrable final** en s'assurant de son bon fonctionnement.

L'interpréteur a été **stabilisé et finalisé** pour garantir une exécution fluide des expressions ensemblistes. La fonction principale de l'interpréteur (`src/main.c`) a été mise en place pour **orchestrer l'appel des différents modules** (analyseur lexical, syntaxique et sémantique) et gérer les entrées utilisateur.

Correction des erreurs de parsing et stabilisation de `yyparse():` un des problèmes identifiés lors des tests finaux était que **`yyparse()`** ne **générait pas toujours d'erreur explicite** lorsqu'une **expression mal formée** était analysée.

L'interpréteur est **stabilisé, testé et prêt à être livré**. Avec la correction des erreurs de parsing, la mise à jour de la documentation et l'ajout d'un Makefile, le projet est désormais **complet et fonctionnel**.

Compilation avec Makefile :

```
make set_interpreter
```

./set_interpreter < tests/test_interpreter.data