

Compilation

Rapport de projet

Louka DOZ
Le 3 juin 2022

1. Lancer le projet

Le projet peut être compilé à l'aide du Makefile en lançant la commande : *make*.

Il peut ensuite être lancé avec la commande : *./prog*.

Il est possible de passer un fichier en argument grâce à la commande : *./prog <fichier>*.

2. Description technique

2.1. Stockage des polynômes

Les polynômes sont stockés sous forme de listes chaînées (structure *polynomial*) dont les attributs sont :

- Un nom
- Un nom de variable
- Une liste chaînées de monômes composants la définition

J'ai fais le choix de stocker la variables dans le polynôme plutôt que dans chaque monôme (structure *monomial*) pour des questions de performance. Cependant, les monômes étant créés avant d'être rassemblés en polynôme, il fallait un moyen de vérifier que la variable d'un monôme était la même que celle déclarée dans le polynôme.

Pour cela, j'ai une structure temporaire nommé *tmp_monomial* qui contient un monôme et sa variable. Une fois que j'ai vérifié que la variable était la bonne, je me débarrasse de *tmp_monomial* pour ne conserver que le monôme.

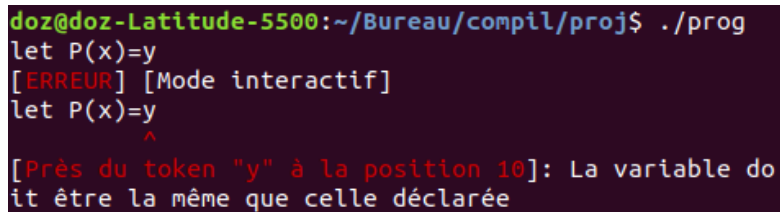
La structure *monomial* stocke le coefficient et le degré du monome.

Avant d'être stockés ou affichés, les monômes sont réduits. C'est-à-dire que les monômes de même degrés sont additionnés et que ceux valant zéro sont supprimés.

2.2. Affichage

Lorsqu'une erreur survient, j'ai fais le choix de ne pas exécuter la commande en cours. Cependant, cela m'a obligé a faire remonter les informations au niveau des grammaires de départ (*Keyword*) à l'aide de structures temporaires, afin de pouvoir empêcher l'affichage d'informations à l'utilisateur en cas d'erreur. Les informations sont donc calculées tout au long de la lecture de la commande mais ne sont affichées quand dans la grammaire *Keyword*, point de départ de toutes les commandes.

Afin de faciliter la vie de l'utilisateur, j'ai aussi tenté d'afficher le token ayant conduit à l'erreur ou celui qui s'en rapproche, ainsi que sa position sur la ligne, c'est-à-dire : le nombre de caractères qui le sépare du début de la ligne.



```
doz@doz-Latitude-5500:~/Bureau/compil/proj$ ./prog
let P(x)=y
[ERREUR] [Mode interactif]
let P(x)=y
    ^
[Près du token "y" à la position 10]: La variable doit être la même que celle déclarée
```

Pour faire cela, je stocke l'ensemble des token rencontrés (réinitialisé à chaque nouvelle commande) afin de pouvoir afficher la ligne et y désigner le token erroné à l'aide d'un curseur.

Bien sûr, cette fonctionnalité possède encore des défaut, mais peut tout de même aider la compréhension des erreurs.

2.3. Fichiers en argument

Lorsqu'un fichier est passé en argument, j'affiche la ligne, qu'il y ait eu erreur ou non, afin de mieux retrouver la ligne concernée par le résultat affiché.

Les lignes peuvent être séparées d'un ';' ou d'un saut de ligne, y compris pour la dernière commande, je n'ai pas été en mesure d'accepter que la dernière commande du fichier ne finisse par rien.

3. Commandes

3.1. LET

LET peut être utilisée pour définir des polynômes sous forme de liste de monômes ou d'opérations arithmétiques, les degrés étant ≥ 0 .

Exemples :

Let $P(x) = [(1,2), (0,3), (-4,1)] \rightsquigarrow P(x) = x^2 - 4x$

Let $P(x) = (1.x^2 + 3*3)^2 + 2x \rightsquigarrow P(x) = x^4 + 18.x^2 + 2.x + 81$

3.2. Opérations arithmétiques

Les opérations arithmétiques permettent de :

- additionner/soustraire/multiplier des polynôme ou monômes
- mettre des polynôme à la puissance $n \geq 0$
- mettre des polynôme entre parenthèses autant de fois que voulu
- déclarer des polynôme sous forme de liste ou non
- utiliser des polynômes déjà existants en les dérivant $n \geq 0$ fois

Elles sont affichés en triant les monômes par ordre décroissant pour la lisibilité.

Exemples :

(ici, $P(x) = x^2 - 4x$)

$((1.x^2 + [(3,0)])^3)^2 + 2x \rightsquigarrow x^4 + 18.x^2 + 2.x + 81$

$P'(x) \rightsquigarrow 2.x - 4$

$P''(x) \rightsquigarrow 2$

3.3. SHOW

Show affiche les polynômes sous forme de liste ou non, avec les monômes par ordre croissant, décroissant, ou classique. L'ordre classique (quand aucun ordre n'est spécifié), affiche les monômes dans l'ordre dans lequel ils ont été entrés par l'utilisateur (le polynôme aura été réduit, certains monômes auront donc été additionnés ou supprimés).

Exemples :

SHOW P DESC $\rightsquigarrow P(x) = x^2 - 4.x$

SHOW P ASC $\rightsquigarrow P(x) = -4.x + x^2$

3.4. EVAL

EVAL permet d'évaluer un polynôme en une variable, une constante symbolique, un entier ou un nombre flottant. Le polynôme est affiché en triant les monômes par ordre décroissant pour la lisibilité.

Exemples :

(ici, $P(x) = x^2 - 4x$)

EVAL P AT y ~> $P(y) = y^2 - 4.y$

EVAL P AT pi ~> $P(y) = \pi^2 - 4.\pi$

EVAL P AT 4 ~> $P(y) = 0$ EVAL P AT 0.5 ~> $P(0.5) = -1.75$

3.5 DERIVE

DERIVE dérive un polynôme mais peut aussi l'évaluer une fois dérivé tel que décrit dans la partie « 3.4. EVAL ». Pour montrer que le polynôme est dérivé, j'affiche une apostrophe après le nom du polynôme. Celle-ci n'indique pas le nombre de dérivations, juste le fait que le polynôme a été dérivé. Celui-ci n'est pas stocké, juste affiché.

Exemples :

(ici, $P(x) = x^2 - 4x$)

DERIVE 2 TIMES P ~> $P'(x) = 2$

DERIVE P AT y ~> $P'(y) = 2.y - 4$

3.6. FIND ROOT

Finalement, la fonctionnalité pour trouver une racine par dichotomie a aussi été implémentée. Celle-ci ne fonctionne évidemment que si $a < b$ et $P(a)P(b) \leq 0$. Les étapes intermédiaires peuvent aussi être stockés dans un fichier Markdown.

Pour reconnaître un nom de fichier je reconnait en fait un nom de polynôme ou de constante symbolique compris entre 1 et 256 caractères. J'y ajouter ensuite l'extension « .md » lors de la création du fichier.

Pour différencier un nom de polynôme ou de constante symbolique d'un nom de fichier, je retourne un token *POLY* (nom polynôme) ou *SYMCONST* (nom constante symbolique) uniquement si la taille est entre 1 et 15 caractères.

Exemple :

(ici, $P(x) = x$)

FIND ROOT FOR P BETWEEN 0 AND 150 ~> 0.000000

4. Conclusion

Seule les intégrales n'ont pas pu être implémentées car cela demandait beaucoup trop de changements. J'ai donc préféré me concentrer sur le perfectionnement des messages d'erreur et la suppression des conflits.

Je pense évidemment que beaucoup de choses peuvent être améliorées comme la raison d'une erreur par exemple, qui reste malgré tout pas toujours claire. J'aurais voulu pouvoir permettre de dériver un polynôme non défini précédemment dans les opérations arithmétiques. Par exemple faire : $(x+1)' \sim > 1$. J'aurais aussi aimé pouvoir faire en sorte de ne pas avoir à mettre de saut de ligne ou de ' ; ' à la fin d'un fichier de commandes.

Enfin, le choix de faire remonter toutes les informations jusqu'à la grammaire *Keyword* pour ne pas afficher le résultat des commandes en cas d'erreur n'est peut-être pas la meilleure solution car cela rajoute beaucoup de code et structures temporaires, mais je n'ai pas trouvé d'autre solutions.