école
normale
supérieure
paris—saclay

Accélération de la synthèse de programme : redémarrages pendant la recherche

Félix Yvonnet

Sous la supervision de Nathanaël Fijalkow et Théo Matricon

LaBRI, équipe M2F

4 septembre 2023

Sommaire

Présentation du stage

- Équipe Méthodes et Modèles Formels (M2F)

Présentation du stage

- Équipe Méthodes et Modèles Formels (M2F)
- Projet ProgSynth

Présentation du stage

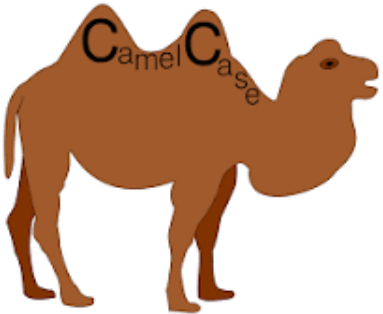
- Équipe Méthodes et Modèles Formels (M2F)
- Projet ProgSynth
- Synthèse de programmes

Présentation du stage

- Équipe Méthodes et Modèles Formels (M2F)
- Projet ProgSynth
- Synthèse de programmes
- Supervisé par Théo Matricon et Nathanaël Fijalkow

Problème

On cherche une fonction qui, à partir de deux mots, donne son abréviation en CamelCase.



Problème

On cherche une fonction qui, à partir de deux mots, donne son abréviation en CamelCase.

- On code soit même :

```
lambda mot1, mot2: (mot1.capitalize() + mot2.capitalize())
```

Problème

On cherche une fonction qui, à partir de deux mots, donne son abréviation en CamelCase.

- On code soit même :

```
lambda mot1, mot2: (mot1.capitalize() + mot2.capitalize())
```

- Synthèse de programme : trouver un programme à partir de critères le définissant

Problème

On cherche une fonction qui, à partir de deux mots, donne son abréviation en CamelCase.

- On code soit même :

```
lambda mot1, mot2: (mot1.capitalize() + mot2.capitalize())
```

- Synthèse de programme : trouver un programme à partir de critères le définissant

  i. par langage naturel (chatGPT : ”code moi ceci”)

Problème

On cherche une fonction qui, à partir de deux mots, donne son abréviation en CamelCase.

- On code soit même :

```
lambda mot1, mot2: (mot1.capitalize() + mot2.capitalize())
```

- Synthèse de programme : trouver un programme à partir de critères le définissant

  i. par langage naturel (chatGPT : ”code moi ceci”)

  ii. par des exemples : un ensemble de couples (entrée, sortie) attendus

La synthèse de nos jours

Mode actuelle : **transformers**

La synthèse de nos jours

Mode actuelle : **transformers**

Avantages : performant pour la compréhension du langage

La synthèse de nos jours

Mode actuelle : **transformers**

Avantages : performant pour la compréhension du langage

Exemples : utilisé par ChatGPT, Copilot ou Tabnine

La synthèse de nos jours

Mode actuelle : **transformers**

Avantages : performant pour la compréhension du langage

Exemples : utilisé par ChatGPT, Copilot ou Tabnine

Mais Il y a aussi des problèmes...

Limites des transformers

- Écologiques : entraîner GPT-3 $\approx$ 200 vols aller-retour Paris et New-York

Limites des transformers

- Écologiques : entraîner GPT-3 $\approx$ 200 vols aller-retour Paris et New-York
- Correction : pas de certification de validité pour les résultats

Limites des transformers

- Écologiques : entraîner GPT-3 $\approx$ 200 vols aller-retour Paris et New-York
- Correction : pas de certification de validité pour les résultats
- Éthiques : base de données pas toujours libre d'accès (Midjourney)

Limites des transformers

- Écologiques : entraîner GPT-3 $\approx$ 200 vols aller-retour Paris et New-York
- Correction : pas de certification de validité pour les résultats
- Éthiques : base de données pas toujours libre d'accès (Midjourney)
- Vie privée : l'Italie a bannie ChatGPT craignant pour la vie privée

Limites des transformers

- Écologiques : entraîner GPT-3 $\approx$ 200 vols aller-retour Paris et New-York
- Correction : pas de certification de validité pour les résultats
- Éthiques : base de données pas toujours libre d'accès (Midjourney)
- Vie privée : l'Italie a bannie ChatGPT craignant pour la vie privée
- Pénaux : quels sont les droits numériques ?

Le projet

L'idée de ce framework est de prendre le contre-pied de ces problèmes.

- L'éthique, la vie privée et le pénal ne posent pas de problème
- Transformers peu écologique $\longrightarrow$
- Certifier la validité $\longrightarrow$

Le projet

L'idée de ce framework est de prendre le contre-pied de ces problèmes.

- L'éthique, la vie privée et le pénal ne posent pas de problème
- Transformers peu écologique $\longrightarrow$ synthèse de programme par exemples
- Certifier la validité $\longrightarrow$

Le projet

L'idée de ce framework est de prendre le contre-pied de ces problèmes.

- L'éthique, la vie privée et le pénal ne posent pas de problème
- Transformers peu écologique $\longrightarrow$ synthèse de programme par exemples
- Certifier la validité $\longrightarrow$ tester tous les programmes

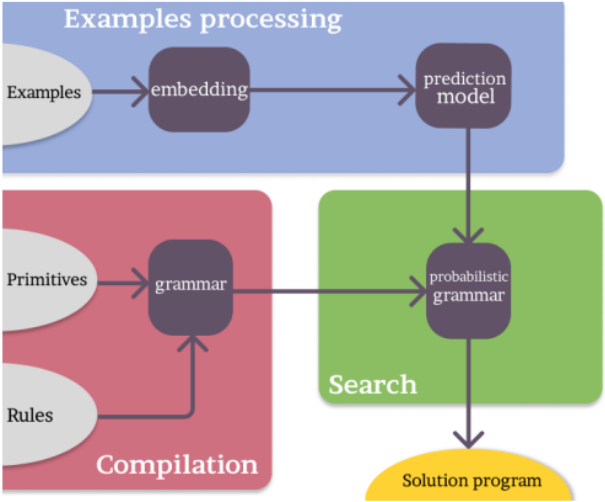
Le framework ProgSynth

Entrée : Exemples + Domain Specific Language (DSL) : langage fonctionnel

Corps :

- DSL $\longrightarrow$ grammaire algébrique (CFG)
- CFG $\longrightarrow$ PCFG par Intelligence Artificielle (IA)
- énumération des programmes jusqu'à un qui convienne

Sortie : Un programme qui satisfait la spécification



Exemple d'application

Voyons comment faire sur un exemple : la contraction en CamelCase de deux mots.

Exemples

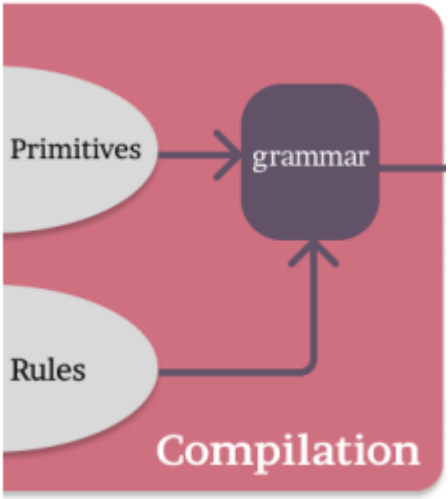
Abréviation en CamelCase de deux mots

```
[
    (["felix", "yvonnet"], "FelixYvonnet"),
    (["Sarah", "Connor"], "SarahConnor"),
    (["une", "fonction"], "UneFonction"),
    (["pascal", "Case"], "PascalCase"),
    (["aBCDEFGH", ""], "ABCDEFGH"),
]
```

Programme $STR \to STR \to STR$

DSL

- On définit les primitives : type et sémantique.
- On ajoute des règles supplémentaires : associativité, commutativité...



Exemple de DSL

```
1 "STR"     : Σ* où Σ = {"a", ··· , "z", "A", ··· , "Z"}
2 "CONCAT"  : STR ⟶ STR ⟶ STR
3 "CAP"     : STR ⟶ STR
4 "REV"     : STR ⟶ STR
```

Exemple de DSL

```
5 "STR"    : str
6 "CONCAT" : lambda x: (lambda y: (x + y))
7 "CAP"    : lambda x: (x.capitalize())
8 "REV"    : lambda x: (x[::-1])
```

Exemple de DSL

```
9  "STR"     : str
10 "CONCAT"  : lambda x: (lambda y: (x + y))
11 "CAP"     : lambda x: (x.capitalize())
12 "REV"     : lambda x: (x[::-1])
```

Ajout $VAR0$, $VAR1$: STR (on cherche $STR \to STR \to STR$)

Exemple de DSL

```
13 "STR"     : str
14 "CONCAT"  : lambda x: (lambda y: (x + y))
15 "CAP"     : lambda x: (x.capitalize())
16 "REV"     : lambda x: (x[::-1])
```

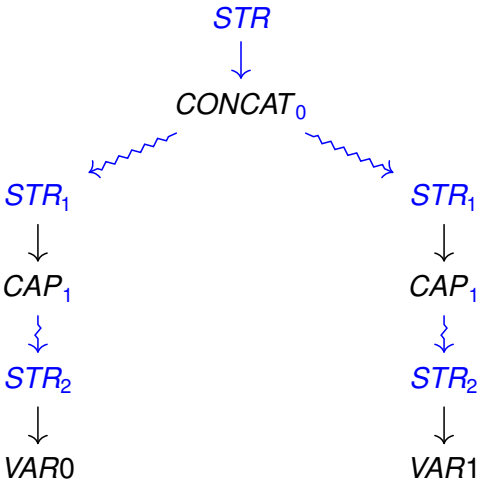
Ajout $VAR0$, $VAR1$: STR (on cherche $STR \rightarrow STR \rightarrow STR$)

Vers grammaire ?

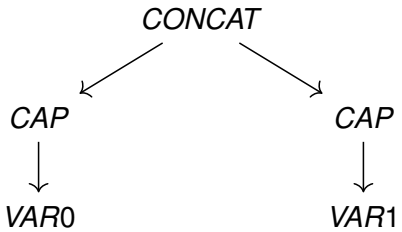
Exemple de grammaire

```
Pour i < max_depth :
STR_i     -> CONCAT_i | CAP_i | REV_i
                      | c ∈ Σ* | "VAR"[int]
CONCAT_i  -> "CONCAT" (STR_{i+1}) (STR_{i+1})
CAP_i     -> "CAP" (STR_{i+1})
REV_i     -> "REV" (STR_{i+1})
```

Un programme ?



Un programme ?



Représente le passage en CamelCase, c'est à dire la fonction :

```
lambda VAR0: (lambda VAR1: (CONCAT(CAP(VAR0), CAP(VAR1))))
```

Nécessité d'un ordonnancement

Énumérer dans un ordre avantageux ?

Nécessité d'un ordonnancement

$u_n =$ nombre de programmes de profondeur exactement n

$u_{n+1} = 2u_n + u_n^2$

$u_1 = 54$, $u_2 = 3024$, $u_3 = 9.150.624...$

$u_n \approx 54^{2^{(n-1)}}$

IA et PCFG

Un réseau de neuronne va deviner les probabilités pour nous

IA et PCFG

Un réseau de neuronne va deviner les probabilités pour nous

Attribue une probabilité à chaque dérivation

IA et PCFG

Un réseau de neuronne va deviner les probabilités pour nous

Attribue une probabilité à chaque dérivation

Lecture des programmes par probabilité decroissante

IA et PCFG

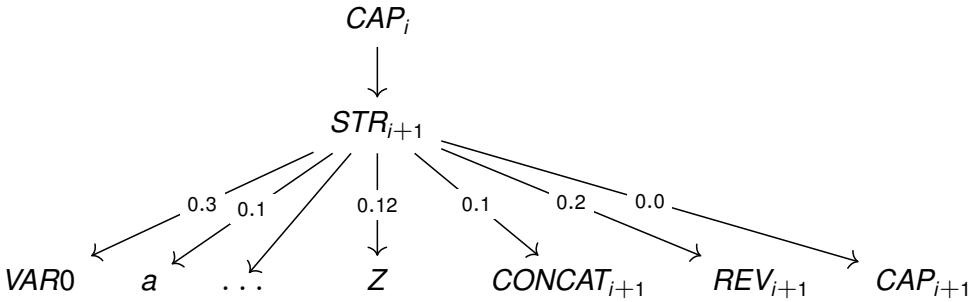
Un réseau de neuronne va deviner les probabilités pour nous

Attribue une probabilité à chaque dérivation

Lecture des programmes par probabilité decroissante

A^* ou HeapSearch

Exemple



Limites

Soit il trouve vite soit il ne sait pas et cherche longtemps.

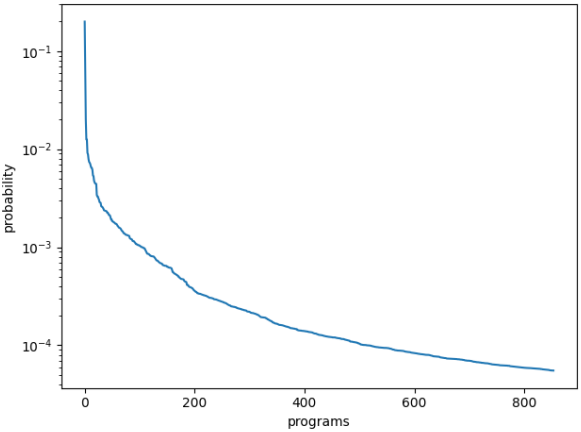


Figure: exemple de loi de distribution de programmes

Limites

Soit il trouve vite soit il ne sait pas et cherche longtemps.

Pas le comportement attendu au début du stage : un changement l'a rendu moins performant.

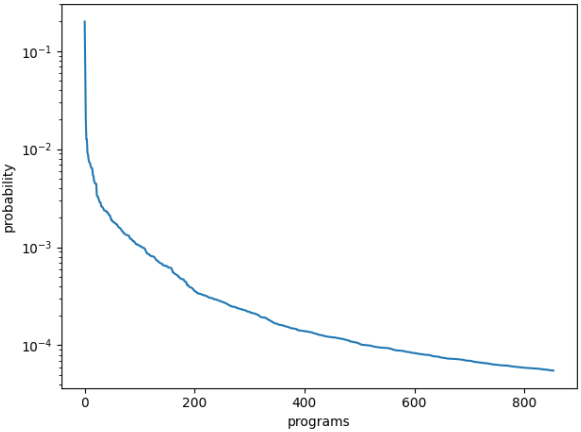


Figure: exemple de loi de distribution de programmes

Redémarrages

Solveurs SAT

Redémarrages

Solveurs SAT

Extraire l'information des précédentes évaluations

Redémarrages

Solveurs SAT

Extraire l'information des précédentes évaluations

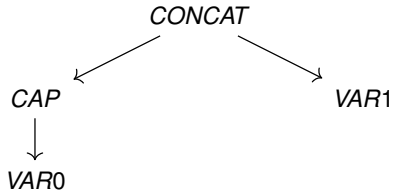
Hypothèse : résultats proches $\approx$ écritures proches

Intuition

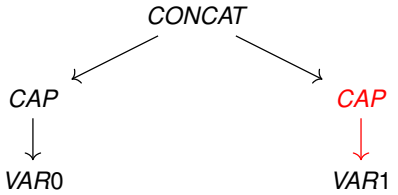
Abréviation en CamelCase de deux mots

```
[
    (["felix", "yvonnet"], "FelixYvonnet"),
    (["Sarah", "Connor"], "SarahConnor"),
    (["une", "fonction"], "UneFonction"),
    (["pascal", "Case"], "PascalCase"),
    (["aBCDEFGH", ""], "ABCDEFGH"),
]
```

Intuition



3/5



5/5

Objectifs

- Trouver quand redémarrer.
- Trouver comment extraire de l'information des précédentes évaluations.

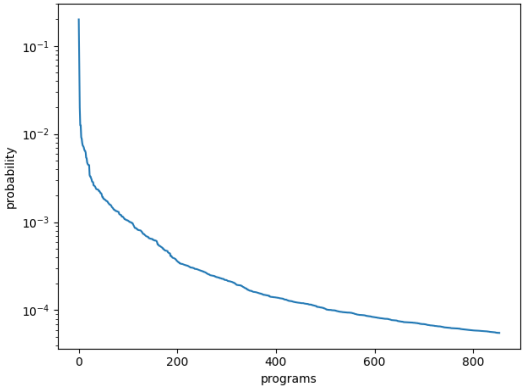


Figure: exemple de loi de distribution de programmes

Séparateur

- On cherche quand le réseau n'arrive plus à différencier les programmes

Séparateur

- On cherche quand le réseau n'arrive plus à différencier les programmes
- Cela correspond à quand la dérivée de la courbe est presque nulle

Séparateur

- On cherche quand le réseau n'arrive plus à différencier les programmes
- Cela correspond à quand la dérivée de la courbe est presque nulle
- Quand $Pr(\#i) - Pr(\#i + 1) < \varepsilon$

Séparateur

- On cherche quand le réseau n'arrive plus à différencier les programmes
- Cela correspond à quand la dérivée de la courbe est presque nulle
- Quand $Pr\big(\#i\big) - Pr\big(\#i + 1\big) < \varepsilon$
- Plus une limite pour avoir un minimum de programmes observés

Séparateur

- On cherche quand le réseau n'arrive plus à différencier les programmes
- Cela correspond à quand la dérivée de la courbe est presque nulle
- Quand $Pr(\#i) - Pr(\#i + 1) < \varepsilon$
- Plus une limite pour avoir un minimum de programmes observés
- $\varepsilon = 10^{-7}$ marche bien

Monte Carlo Tree Search (MCTS)

Quoi ?

- Arbre

Pourquoi ?

Monte Carlo Tree Search (MCTS)

Quoi ?

- Arbre
- Nœud = choix de configuration

Pourquoi ?

Monte Carlo Tree Search (MCTS)

Quoi ?

- Arbre
- Nœud = choix de configuration
- Représente un sous-arbre de l'ensemble des possibles (grammaire)

Pourquoi ?

Monte Carlo Tree Search (MCTS)

Quoi ?

- Arbre
- Nœud = choix de configuration
- Représente un sous-arbre de l'ensemble des possibles (grammaire)

Pourquoi ?

- Ressemble à la grammaire (combinaison rapide)

Monte Carlo Tree Search (MCTS)

Quoi ?

- Arbre
- Nœud = choix de configuration
- Représente un sous-arbre de l'ensemble des possibles (grammaire)

Pourquoi ?

- Ressemble à la grammaire (combinaison rapide)
- Itérable

Monte Carlo Tree Search (MCTS)

Quoi ?

- Arbre
- Nœud = choix de configuration
- Représente un sous-arbre de l'ensemble des possibles (grammaire)

Pourquoi ?

- Ressemble à la grammaire (combinaison rapide)
- Itérable
- Pas besoin de plus, tout se déduit de la forme d'arbre

MCTS

Comment ?

- Arbre de base

MCTS

Comment ?

- Arbre de base

- Ajout des nouveaux programmes en mélangeant aux précédents

MCTS

Comment ?

- Arbre de base
- Ajout des nouveaux programmes en mélangeant aux précédents
- Avec leur valeurs

MCTS

Comment ?

- Arbre de base
- Ajout des nouveaux programmes en mélangeant aux précédents
- Avec leur valeurs
- Déduire des valeurs la nouvelle probabilité de la dérivation correspondante

Exemple

Abréviation en CamelCase de deux mots

```
[
    (["felix", "yvonnet"], "FelixYvonnet"),
    (["Sarah", "Connor"], "SarahConnor"),
    (["une", "fonction"], "UneFonction"),
    (["pascal", "Case"], "PascalCase"),
    (["aBCDEFGH", ""], "ABCDEFGH"),
]
```

Exemple

MCTS :

STR

Exemple

Programme :

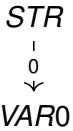
succès : 0/5

MCTS :

STR

Exemple

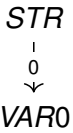
MCTS :



Exemple

Programme :

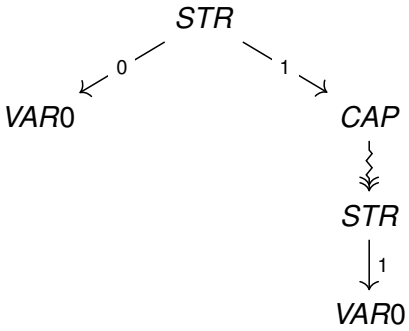
MCTS :



succès : 1/5

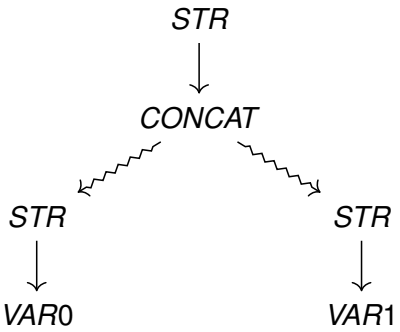
Exemple

MCTS :

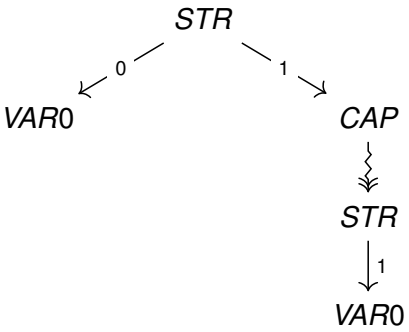


Exemple

Programme :



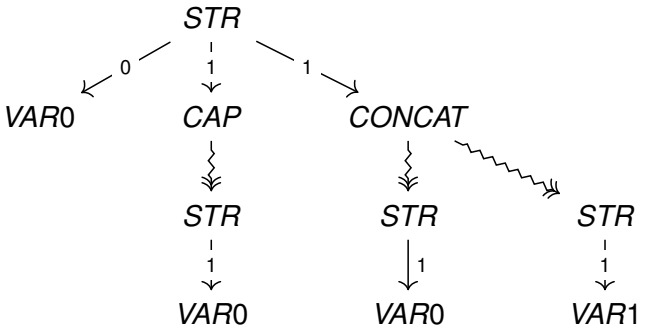
MCTS :



succès : 1/5

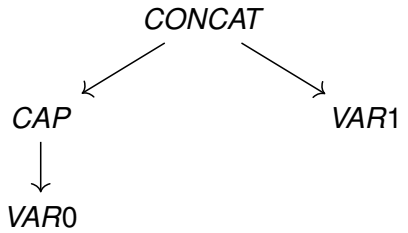
Exemple

MCTS :



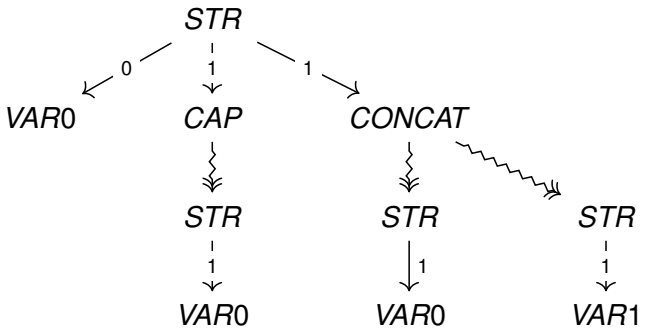
Exemple

Programme :

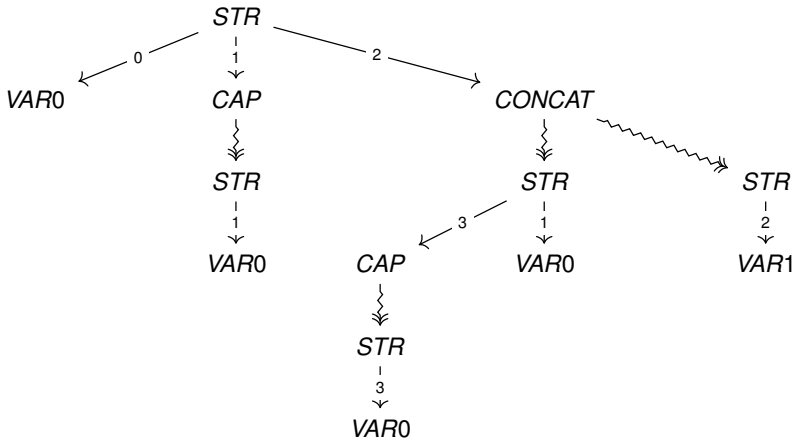


succès : 3/5

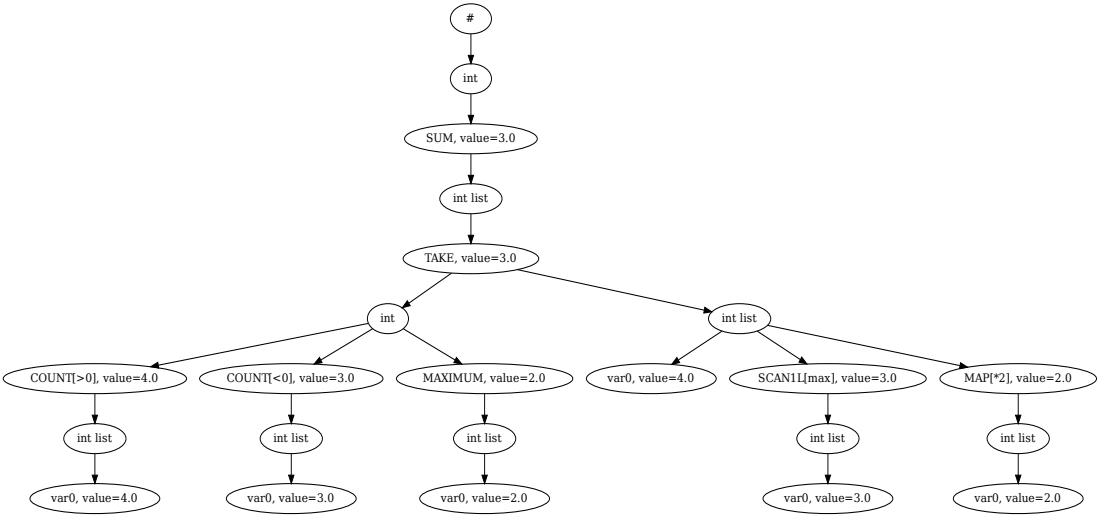
MCTS :



Exemple


STR
0
1
2
VAR0
CAP
CONCAT
STR
1
STR
3
1
STR
2
VAR0
CAP
VAR0
VAR1
STR
3
VAR0

Exemple réel



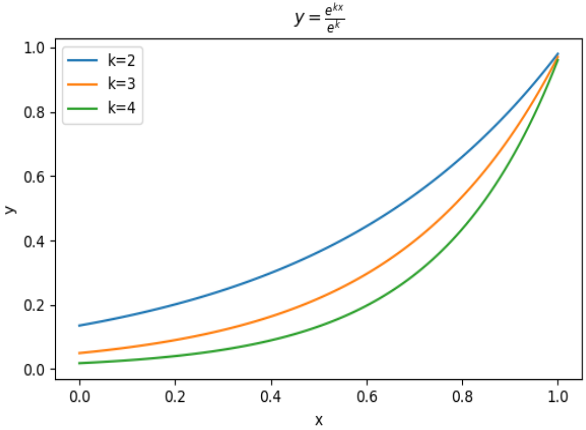
Vers probabilités

On préfère **vraiment** quand c'est grand

Vers probabilités

On préfère **vraiment** quand c'est grand

Courbe convexe croissante (fonction d'échelonnage)

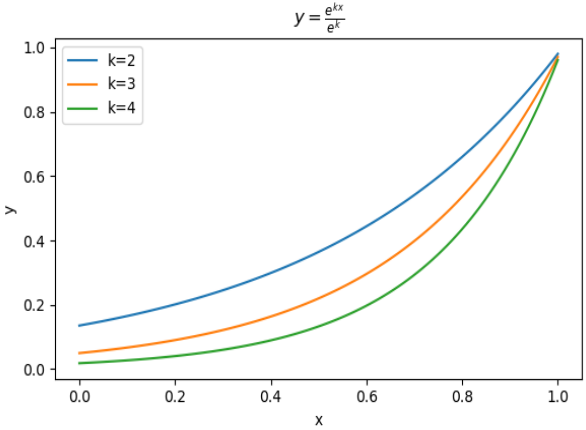


Vers probabilités

On préfère **vraiment** quand c'est grand

Courbe convexe croissante (fonction d'échelonnage)

Mélange avec les anciennes probabilités



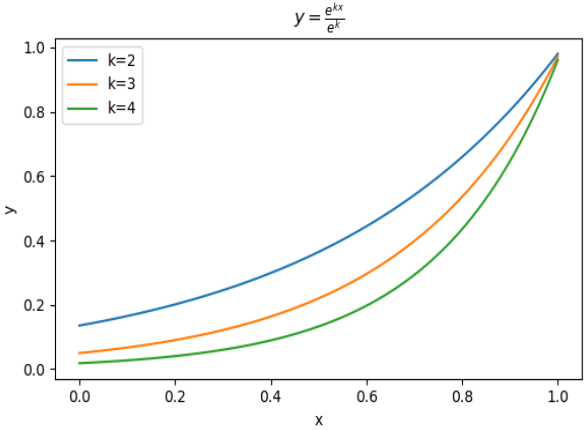
Vers probabilités

On préfère **vraiment** quand c'est grand

Courbe convexe croissante (fonction d'échelonnage)

Mélange avec les anciennes probabilités

Normalisation



Rappel : paramètres

Rappels des différents paramètres à faire varier :

- ε pour choisir quand relancer
- facteur de courbure pour fonction d'échelonnage
- Comment insérer les résultats en probabilité

Choix de la méthode

Limite de temps à 30 secondes :

méthode	temps par tâche réussie (s)	programmes vus en moyenne	solutions trouvées
sans redémarrage	3.52	19.483	49/96

Choix de la méthode

Limite de temps à 30 secondes :

méthode	temps par tâche réussie (s)	programmes vus en moyenne	solutions trouvées
sans redémarrage	3.52	19.483	49/96
(2)	5.79	12.659	51/96

(2) : $k = 3$, on multiplie les anciennes probabilités par la valuation plus un epsilon et on normalise

Choix de la méthode

Limite de temps à 30 secondes :

méthode	temps par tâche réussie (s)	programmes vus en moyenne	solutions trouvées
sans redémarrage	3.52	19.483	49/96
(2)	5.79	12.659	51/96
(3)	4.28	12.097	52/96

(2) : $k = 3$, on multiplie les anciennes probabilités par la valuation plus un epsilon et on normalise

(3) : $k = 3$, la nouvelle probabilité vaut la valuation normalisée

Comparaison à même niveau

Comparaison des résultats pour un même nombre de programmes trouvés :

méthode	temps par tâche réussie (s)	programmes vus en moyenne	solutions trouvées
(3) (30s)	4.28	12.097	52/96

Comparaison à même niveau

Comparaison des résultats pour un même nombre de programmes trouvés :

méthode	temps par tâche réussie (s)	programmes vus en moyenne	solutions trouvées
(3) (30s)	4.28	12.097	52/96
sans redémarrage (60s)	5.52	32.550	52/96

Comparaison à même niveau

Comparaison des résultats pour un même nombre de programmes trouvés :

méthode	temps par tâche réussie (s)	programmes vus en moyenne	solutions trouvées
(3) (30s)	4.28	12.097	52/96
sans redémarrage (60s)	5.52	32.550	52/96
(3) (60s)	7.10	17.124	55/96

Choix de k

Méthode (3) / 30 secondes

constante	temps par tâche réussie (s)	programmes vus en moyenne	solutions trouvées
$k = 3$	4.28	12.097	52/96

Choix de k

Méthode (3) / 30 secondes

constante	temps par tâche réussie (s)	programmes vus en moyenne	solutions trouvées
$k = 2$	4.97	12.193	51/96
$k = 3$	4.28	12.097	52/96

Choix de k

Méthode (3) / 30 secondes

constante	temps par tâche réussie (s)	programmes vus en moyenne	solutions trouvées
$k = 2$	4.97	12.193	51/96
$k = 3$	4.28	12.097	52/96
$k = 4$	5.61	13.197	50/96

Résolution de bugs

Manque de temps

Mises à jour

Optimisations

Corrections pour des cas particuliers

Conclusion

- Développement de méthodes pour repérer quand relancer la recherche

Conclusion

- Développement de méthodes pour repérer quand relancer la recherche
- Développement de méthodes pour extraire l'information des précédentes évaluations

Conclusion

- Développement de méthodes pour repérer quand relancer la recherche
- Développement de méthodes pour extraire l'information des précédentes évaluations
- Corrections de bugs informatiques et optimisations

Conclusion

- Développement de méthodes pour repérer quand relancer la recherche
- Développement de méthodes pour extraire l'information des précédentes évaluations
- Corrections de bugs informatiques et optimisations
- Démonstration de l'impossibilité de comparation entre distances syntaxiques et sémantiques

Pour aller plus loin

- Technique améliorable avec de meilleurs paramètres

Pour aller plus loin

- Technique améliorable avec de meilleurs paramètres
- Améliorations possibles sur le reseau de neurones

Pour aller plus loin

- Technique améliorable avec de meilleurs paramètres
- Améliorations possibles sur le reseau de neurones
- Extension au cas réel (exemple en annexe)

FIN

Merci pour votre attention.

Cas réel

Problème : recherche du terme général d'une suite récurrente $u_{n+1} = f(u_n)$

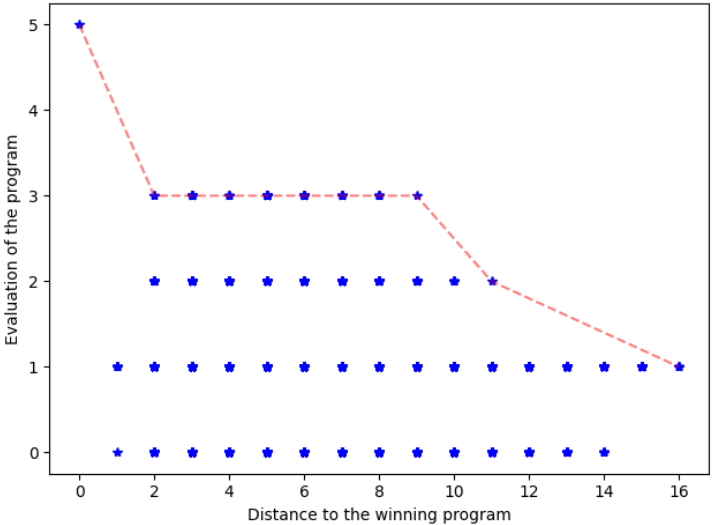
DSL : opérations usuelles sur les réels et entiers : $+, *, /, \hat{}, !\ldots$

Entrée : premiers termes de la suite

Une fonction $u(n)$ convient $\implies$ récurrence automatisée $u_n \overset{?}{=} u(n)$

Recommancer tant qu'on n'en trouve pas une qui convient

Retour sur hypothèse



Autres approches

Estimation de l'évaluation à partir des distances

Autres approches

Estimation de l'évaluation à partir des distances

- $$estimate(p) = \sum_{g \in \mathcal{F}_{eval}} \underbrace{\frac{eval(g)}{dist(p,g)}}_{\substack{\text{proximité syntaxique} \\ \text{implique proximité} \\ \text{des résultats}}} + \underbrace{\frac{dist(p,g) \cdot eval(g)}{\max_{f \in \mathcal{F}, f' \in \mathcal{F}_{eval}} dist(f,f') \cdot |\mathcal{F}_{eval}|^2}}_{\substack{\text{si on est loin de tout} \\ \text{alors la distance} \\ \text{est inconnue}}}$$

Autres approches

Estimation de l'évaluation à partir des distances

- $estimate(p) = \sum_{g \in \mathcal{F}_{eval}} \underbrace{\frac{eval(g)}{dist(p,g)}}_{\text{proximité syntaxique implique proximité des résultats}} + \underbrace{\frac{dist(p,g) \cdot eval(g)}{\max_{f \in \mathcal{F}, f' \in \mathcal{F}_{eval}} dist(f,f') \cdot |\mathcal{F}_{eval}|^2}}_{\text{si on est loin de tout alors la distance est inconnue}}$

- triangulation : la distance entre fonctions permet la localisation de l'évaluation.