Interpréteur de suites de jetons de Rózsa

Mattéo Delabre
Guillaume Pérution-Kihli
Julien Rodriguez

HLIN203 — Modèles de calcul
16 avril 2019

Fonctions μ-récursives et jetons de Rózsa

Fonctions μ-récursives

Fonctions pouvant être construites en combinant :

- 0 : la constante 0 ;
- I : la fonction identité ;
- S : la fonction successeur ;
- ◀ ▶ : les projections ;
- ⊚ : la composition ;
- R : la récursion ;
- μ : la minimisation.

- Ce sont les fonctions calculables.
- Modèle de calcul équivalent aux machines de Turing.
- Jetons : symboles pour écrire les fonctions μ-récursives.

Pourquoi un interpréteur de suites de jetons ?

- Permettre d'expérimenter avec les suites de jetons plus facilement.
- Recenser des suites pour former une bibliothèque de suites.
- Analyser et comparer les suites.
- Diffuser ce modèle de calcul.
- Simuler les jetons dans un ordinateur.
  $\Rightarrow$ Machines de Turing $\geq$ Jetons de Rózsa

Quiz

Quelle est cette suite ?

Quiz

Quelle est cette suite ?

Quiz

Quelle est cette suite ?

Sommaire

Choix de conception

- Rendre l'application la plus accessible possible $\rightarrow$ le web.
- Pas beaucoup d'autre choix que...

Choix de conception

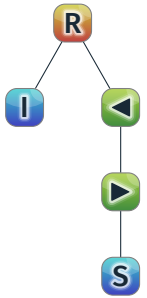
- Rendre l'application la plus accessible possible $\rightarrow$ le web.
- Pas beaucoup d'autre choix que… JavaScript.
- Pour le meilleur :
  - tout le monde peut l'exécuter sans installation particulière ;
  - prototypage rapide ;

Choix de conception

- Rendre l'application la plus accessible possible $\rightarrow$ le web.
- Pas beaucoup d'autre choix que… JavaScript.
- Pour le meilleur :
  - tout le monde peut l'exécuter sans installation particulière ;
  - prototypage rapide ;
- et pour le pire :
  - langage conçu en 10 jours ;
  - langage impératif ;
  - pas de typage statique ;
  - propension aux comportements contre-intuitifs.

Première étape : construction d'un arbre

- Représentation linéaire des jetons : se prête peu au calcul.
- Création d'un arbre de syntaxe à partir de la suite.
- Relativement simple car langage en notation préfixe.
- On connaît assez facilement le nombre d'arguments attendus par chaque constructeur.



Seconde étape : évaluation de l'arbre

Idée simple : transposer la définition des jetons sous forme de fonction JavaScript. Appliquer la bonne fonction selon le jeton en racine d'arbre.

Fonctions de base

- 0 $= 0$
- I $(x) = x$
- S $(x) = x + 1$

Seconde étape : évaluation de l'arbre

Idée simple : transposer la définition des jetons sous forme de fonction JavaScript. Appliquer la bonne fonction selon le jeton en racine d'arbre.

Fonctions de base

- 0 $= 0$
- I $(x) = x$
- S $(x) = x + 1$

Constructeurs

- ◀ $f = (x, \bar{y}) \mapsto f(\bar{y})$
- ▶ $f = (\bar{x}, y) \mapsto f(\bar{x})$

Seconde étape : évaluation de l'arbre

Idée simple : transposer la définition des jetons sous forme de fonction JavaScript. Appliquer la bonne fonction selon le jeton en racine d'arbre.

Fonctions de base

- 0 $= 0$
- I $(x) = x$
- S $(x) = x + 1$

Constructeurs

- ◀ $f = (x, \bar{y}) \mapsto f(\bar{y})$
- ▶ $f = (\bar{x}, y) \mapsto f(\bar{x})$
- @ $f\ g_1\ \ldots\ g_n = \bar{x} \mapsto f(g_1(\bar{x}), \ldots, g_n(\bar{x}))$

Seconde étape : évaluation de l'arbre

Idée simple : transposer la définition des jetons sous forme de fonction JavaScript. Appliquer la bonne fonction selon le jeton en racine d'arbre.

Fonctions de base

- 0 $= 0$
- I $(x) = x$
- S $(x) = x + 1$

Constructeurs

- ◀ $f = (x, \bar{y}) \mapsto f(\bar{y})$
- ▶ $f = (\bar{x}, y) \mapsto f(\bar{x})$
- @ $f\ g_1\ \ldots\ g_n = \bar{x} \mapsto f(g_1(\bar{x}), \ldots, g_n(\bar{x}))$
- R $f\ g = (x, \bar{y}) \mapsto$

$$\begin{cases} f(\bar{y}) & \text{si } x = 0 \\ g(x-1, \text{R}\ f\ g(x-1, \bar{y}), \bar{y}) & \text{sinon} \end{cases}$$

Seconde étape : évaluation de l'arbre

Idée simple : transposer la définition des jetons sous forme de fonction JavaScript. Appliquer la bonne fonction selon le jeton en racine d'arbre.

Fonctions de base

- 0 $= 0$
- I $(x) = x$
- S $(x) = x + 1$

Constructeurs

- ◀ $f = (x, \bar{y}) \mapsto f(\bar{y})$
- ▶ $f = (\bar{x}, y) \mapsto f(\bar{x})$
- @ $f\ g_1\ \ldots\ g_n = \bar{x} \mapsto f(g_1(\bar{x}), \ldots, g_n(\bar{x}))$
- R $f\ g = (x, \bar{y}) \mapsto$

$$\begin{cases} f(\bar{y}) & \text{si } x = 0 \\ g(x-1, \text{R}\ f\ g(x-1, \bar{y}), \bar{y}) & \text{sinon} \end{cases}$$

- μ $f = \min\{x : f(x) = 0\}$

Premier problème : calculs inutiles

Un long calcul

Premier problème : calculs inutiles

Un long calcul

Premier problème : calculs inutiles

Un long calcul

Premier problème : calculs inutiles

Un long calcul

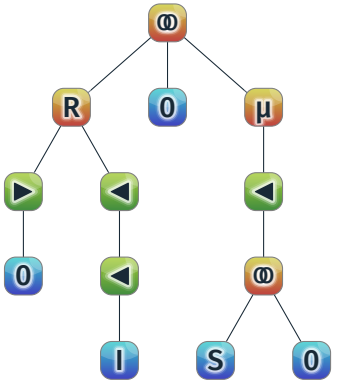
- Évaluation « stricte » des arguments conduit à des calculs inutiles.
- Peut-on fixer une sémantique paresseuse des jetons ?

Premier problème : calculs inutiles

Que fait ce code ?

```
bool diverge()
{
    while (true) {}
}

int main()
{
    return false && diverge();
}
```



Second problème : dépassement de pile

- JavaScript ne gère pas la récursion profonde.
- L'évaluation de 

$(1000, 1) = 1000 + 1$ fait dépasser la pile.

Solution à ces problèmes

- On veut **contrôler** le mécanisme d'évaluation plus finement.
- S'inspirer du mode d'évaluation paresseux du langage Haskell (ou de certains autres langages fonctionnels).
- Créer un graphe d'expression et le réduire étape par étape.
- On suit des règles de réduction qui reprennent les définitions des jetons.

Solution à ces problèmes

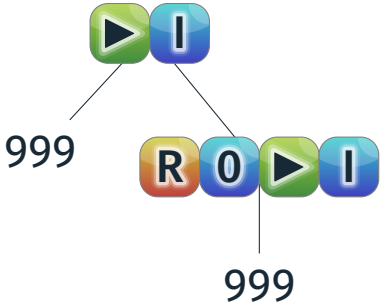
Un exemple

Étape de calcul n°1

Solution à ces problèmes

Un exemple

Étape de calcul n°2



Solution à ces problèmes
Un exemple

Étape de calcul n°3

999

Conséquences

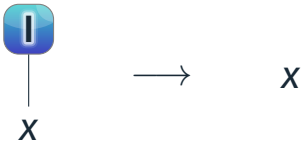
- Le moteur JavaScript ne gère plus la récursion $\Rightarrow$ moins de risque de dépassement de pile.
- Réduction de graphe pouvant être paresseuse : évaluation en commençant par la racine et en ne descendant que lorsque c'est nécessaire.

Règles de réduction

Fonction zéro

Règles de réduction

Fonction identité



Règles de réduction

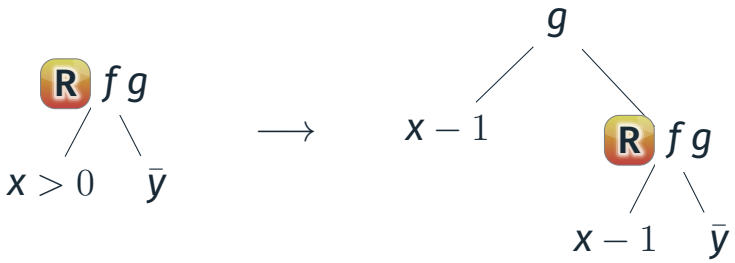
Fonction successeur

Règles de réduction
Projections

Règles de réduction
Composition

Règles de réduction

Récursion : cas général

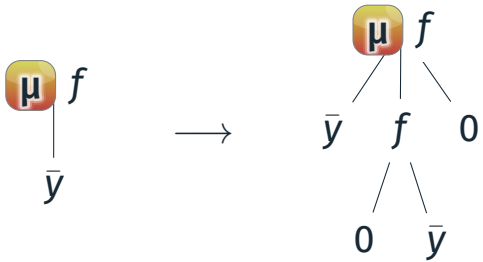


Règles de réduction

Récursion : cas de base

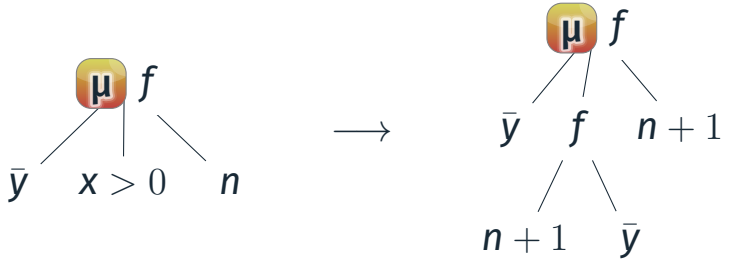
Règles de réduction

Minimisation : initialisation



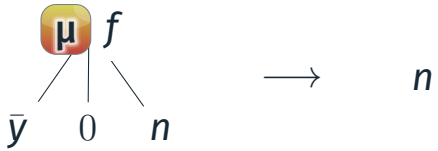
Règles de réduction

Minimisation : itération



Règles de réduction

Minimisation : terminaison



Sommaire

Démonstration

Sommaire

Conclusion

`rozsa-tokens.info`

`bitbucket.org/julienrodriguez/interpreteur-de-jetons-de-rozsa`

Travaux connexes

- *Primitive recursive functions*, Nayuki
  `https://www.nayuki.io/page/primitive-recursive-functions`
- *From recursive functions to Turing machines*, François Schwarzentruber
  `http://people.irisa.fr/Francois.Schwarzentruber/recursive_functions_to_turing_machines/`