Automates implicites en logique linéaire et théorie catégorique des transducteurs

Soutenance de thèse de Lê Thành Dũng (Tito) Nguyễn
sous la direction de Stefano Guerrini et Thomas Seiller
Laboratoire d'informatique de Paris Nord, vendredi 3 décembre 2021

Les ordinateurs nous servent à communiquer
à nous divertir
à travailler
à organiser nos vies
...

Les ordinateurs nous servent à communiquer
à nous divertir
à travailler
à organiser nos vies
…
à *calculer* : nécessaire pour le reste,
et usage historique

Informatique et calcul

- Ada Lovelace (XIX$^{\text{ème}}$) : programme pour calculer des suites mathématiques
- Alan Turing : décryptage de communications chiffrées des nazis
- Années 1940–50 : calcul scientifique (bombes thermonucléaires, météo, ...)

Informatique et calcul

- Ada Lovelace (XIX$^{\text{ème}}$) : programme pour calculer des suites mathématiques
  + idée de la numérisation du monde (par ex. musique)
- Alan Turing : décryptage de communications chiffrées des nazis

- Années 1940–50 : calcul scientifique (bombes thermonucléaires, météo, ...)

Informatique et calcul

- Ada Lovelace (XIX$^{\text{ème}}$) : programme pour calculer des suites mathématiques
  + idée de la numérisation du monde (par ex. musique)
- Alan Turing : décryptage de communications chiffrées des nazis
  + invention de l'*informatique théorique* (années 1930, avant les ordinateurs)
- Années 1940–50 : calcul scientifique (bombes thermonucléaires, météo, ...)

Informatique et calcul

- Ada Lovelace (XIX$^{\text{ème}}$) : programme pour calculer des suites mathématiques
  + idée de la numérisation du monde (par ex. musique)
- Alan Turing : décryptage de communications chiffrées des nazis
  + invention de l'*informatique théorique* (années 1930, avant les ordinateurs)
- Années 1940–50 : calcul scientifique (bombes thermonucléaires, météo, ...)

Une question centrale de l'informatique théorique :

que peut-on calculer, et avec quels moyens ?

$\mathsf{P} \neq \mathsf{NP}$ (« problème du millénaire » à 1M$) : « que peut-on calculer *rapidement* ? »

Théorie classique de la calculabilité

Naissance de l'informatique théorique :

que peut-on calculer, *sans restriction de moyens* ? **Pas tout !**

Théorie classique de la calculabilité

Naissance de l'informatique théorique :

que peut-on calculer, *sans restriction de moyens* ? **Pas tout !**

Et surtout, que signifie rigoureusement cette question ?

Théorie classique de la calculabilité

Naissance de l'informatique théorique :

que peut-on calculer, *sans restriction de moyens* ? **Pas tout !**

Et surtout, que signifie rigoureusement cette question ? Différentes approches :

- « matériel » : *machines de Turing* (abstraction mathématique d'un ordinateur)

Théorie classique de la calculabilité

Naissance de l'informatique théorique :

que peut-on calculer, *sans restriction de moyens* ? **Pas tout !**

Et surtout, que signifie rigoureusement cette question ? Différentes approches :

- « matériel » : *machines de Turing* (abstraction mathématique d'un ordinateur)
- « logiciel » : divers *langages de programmation* théoriques,
  en particulier le *λ-calcul* (ancêtre de Lisp, Haskell, OCaml ...)

Théorie classique de la calculabilité

Naissance de l'informatique théorique :

que peut-on calculer, *sans restriction de moyens* ? **Pas tout !**

Et surtout, que signifie rigoureusement cette question ? Différentes approches :

- « matériel » : *machines de Turing* (abstraction mathématique d'un ordinateur)
- « logiciel » : divers *langages de programmation* théoriques,
  en particulier le λ-*calcul* (ancêtre de Lisp, Haskell, OCaml ...)

Ces définitions sont *équivalentes* :

on a trouvé la « bonne » notion de « tâche entrée $\to$ sortie *calculable* »

Théorie des automates vs langages de programmation

Que peut-on calculer, avec très peu de moyens ?

Automates finis : machines de Turing à mémoire bridée (« matériel »)

(applications : algorithmique du texte (regexp), modélisation de systèmes réactifs, ...)

Théorie des automates vs langages de programmation

Que peut-on calculer, avec très peu de moyens ?

Automates finis : machines de Turing à mémoire bridée (« matériel »)

(applications : algorithmique du texte (regexp), modélisation de systèmes réactifs, ...)

Par contre, la théorie contemporaine du λ-calcul (« logiciel ») ne s'intéresse pas à « que peut-on calculer » mais à des questions de *structure* des programmes

(applications : conception de langages de programmation, fiabilité des logiciels, ...)

Théorie des automates vs langages de programmation

Que peut-on calculer, avec très peu de moyens ?

Automates finis : machines de Turing à mémoire bridée (« matériel »)

(applications : algorithmique du texte (regexp), modélisation de systèmes réactifs, ...)

Par contre, la théorie contemporaine du λ-calcul (« logiciel ») ne s'intéresse pas à « que peut-on calculer » mais à des questions de *structure* des programmes

(applications : conception de langages de programmation, fiabilité des logiciels, ...)

Une exception : un sous-domaine nommé complexité implicite

Propose des langages de programmation (théoriques) dont le « pouvoir calculatoire » correspond à des *classes de complexité algorithmique* (P, NP, ...)

Théorie des automates vs langages de programmation

Que peut-on calculer, avec très peu de moyens ?

Automates finis : machines de Turing à mémoire bridée (« matériel »)

(applications : algorithmique du texte (regexp), modélisation de systèmes réactifs, ...)

Par contre, la théorie contemporaine du λ-calcul (« logiciel ») ne s'intéresse pas à « que peut-on calculer » mais à des questions de *structure* des programmes

(applications : conception de langages de programmation, fiabilité des logiciels, ...)

Une exception : un sous-domaine nommé complexité implicite

Propose des langages de programmation (théoriques) dont le « pouvoir calculatoire » correspond à des *classes de complexité algorithmique* (P, NP, ...)

$\longrightarrow$ restrictions du λ-calcul avec des *systèmes de types*

Théorie des automates vs langages de programmation

Que peut-on calculer, avec très peu de moyens ?

Automates finis : machines de Turing à mémoire bridée (« matériel »)

(applications : algorithmique du texte (regexp), modélisation de systèmes réactifs, ...)

Par contre, la théorie contemporaine du λ-calcul (« logiciel ») ne s'intéresse pas à « que peut-on calculer » mais à des questions de *structure* des programmes

(applications : conception de langages de programmation, fiabilité des logiciels, ...)

Une exception : un sous-domaine nommé complexité implicite

Propose des langages de programmation (théoriques) dont le « pouvoir calculatoire » correspond à des *classes de complexité algorithmique* (P, NP, ...)

$\longrightarrow$ restrictions du λ-calcul avec des *systèmes de types*

Et si on remplaçait P, NP, etc. par des automates dans la complexité implicite ?

Contributions de cette thèse

- *automates implicites* : correspondances entre λ-calculs typés (avec typage *linéaire*) et modèles d'automates

Contributions de cette thèse

- *automates implicites* : correspondances entre λ-calculs typés (avec typage *linéaire*) et modèles d'automates
- explication des « raisons profondes » de ces correspondances, avec des structures mathématiques provenant de la *théorie des catégories*

Contributions de cette thèse

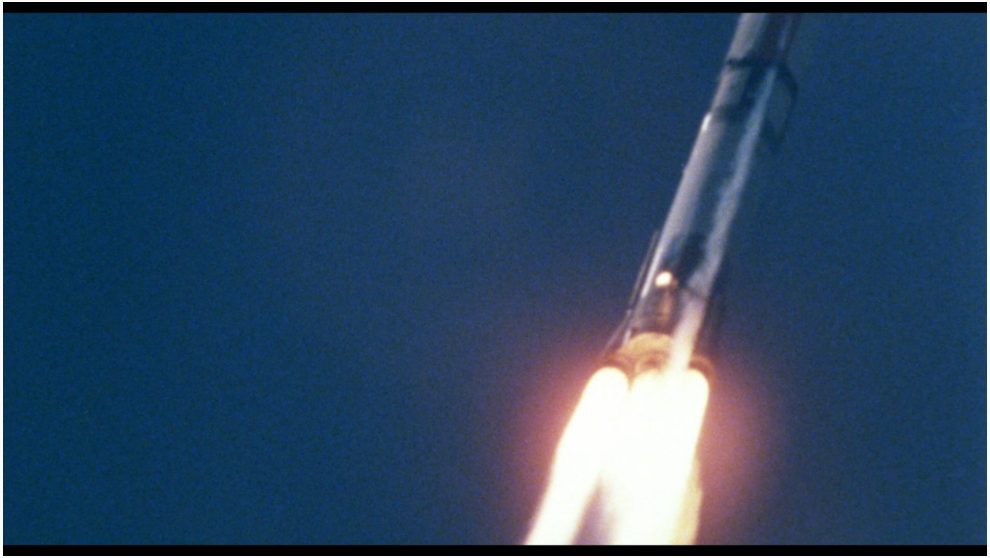
- *automates implicites* : correspondances entre λ-calculs typés (avec typage *linéaire*) et modèles d'automates
- explication des « raisons profondes » de ces correspondances, avec des structures mathématiques provenant de la *théorie des catégories*
- étude de nouveaux modèles d'automates – plus précisément, de *transducteurs* = automates à sortie plus sophistiquée que oui/non

Contributions de cette thèse

- *automates implicites* : correspondances entre λ-calculs typés (avec typage *linéaire*) et modèles d'automates
- explication des « raisons profondes » de ces correspondances, avec des structures mathématiques provenant de la *théorie des catégories*
- étude de nouveaux modèles d'automates – plus précisément, de *transducteurs* = automates à sortie plus sophistiquée que oui/non

Lien entre deux communautés qui se parlent peu, aux cultures différentes. Ex :

Koyaanisqatsi (Godfrey Reggio, 1982)



Koyaanisqatsi (Godfrey Reggio, 1982)

Contributions de cette thèse

- *automates implicites* : correspondances entre λ-calculs typés (avec typage *linéaire*) et modèles d'automates
- explication des « raisons profondes » de ces correspondances, avec des structures mathématiques provenant de la *théorie des catégories*
- étude de nouveaux modèles d'automates – plus précisément, de *transducteurs* = automates à sortie plus sophistiquée que oui/non

Lien entre deux communautés qui se parlent peu, aux cultures différentes.

Ex : éviter les bugs dans des systèmes critiques (pilotage de fusée)

$\longrightarrow$ *vérification de modèles* (automates) ou *preuves de programmes* (λ-calcul) ?

Contributions de cette thèse (avec la collaboration essentielle de C. Pradic !)

- *automates implicites* : correspondances entre λ-calculs typés (avec typage *linéaire*) et modèles d'automates
- explication des « raisons profondes » de ces correspondances, avec des structures mathématiques provenant de la *théorie des catégories*
- étude de nouveaux modèles d'automates – plus précisément, de *transducteurs* = automates à sortie plus sophistiquée que oui/non

Lien entre deux communautés qui se parlent peu, aux cultures différentes.

Ex : éviter les bugs dans des systèmes critiques (pilotage de fusée)

$\longrightarrow$ *vérification de modèles* (automates) ou *preuves de programmes* (λ-calcul) ?

Rappel : le λ-calcul et les codages de Church

Une théorie syntaxique naïve des fonctions :

$$\begin{array}{rcll} f\,x & & \approx & f(x) \\ \lambda x.\, t & & \approx & x \mapsto t \\ (\lambda x.\, t)\, u & \to_\beta\ t\{x := u\} & \approx & (x \mapsto x^2 + 1)(42) = 42^2 + 1 \end{array}$$

Rappel : le λ-calcul et les codages de Church

Une théorie syntaxique naïve des fonctions :

$$\begin{array}{rcl} f\,x & \approx & f(x) \\ \lambda x.\, t & \approx & x \mapsto t \\ (\lambda x.\, t)\, u \;\to_\beta\; t\{x := u\} & \approx & (x \mapsto x^2 + 1)(42) = 42^2 + 1 \end{array}$$

Pas de types de données primitifs (entiers, mots, ...) en λ-calcul ;
on représente les données par des fonctions (*codages de Church*)

Rappel : le λ-calcul et les codages de Church

Une théorie syntaxique naïve des fonctions :

$$\begin{array}{rcl} f\,x & \approx & f(x) \\ \lambda x.\, t & \approx & x \mapsto t \\ (\lambda x.\, t)\, u \;\; \to_\beta \;\; t\{x := u\} & \approx & (x \mapsto x^2 + 1)(42) = 42^2 + 1 \end{array}$$

Pas de types de données primitifs (entiers, mots, ...) en λ-calcul ;
on représente les données par des fonctions (*codages de Church*)

Idée : $abb \in \{a, b\}^*$ est codé par $(f_a, f_b) \mapsto f_a \circ f_b \circ f_b$ $\qquad$ ($\{a, b\}^*$ = mots sur $\{a, b\}$)

$$\overline{abb} = \lambda f_a.\, \lambda f_b.\, \lambda x.\, f_a\,(f_b\,(f_b\,x))$$

Le λ-calcul simplement typé

On ajoute un *système de type* : spécifications pour les λ-termes

$$t : A \to B \quad \approx \quad \text{« } t \text{ est une fonction de } A \text{ dans } B \text{ »}$$

Types simples : engendrés par « $\to$ » à partir d'un type de base o

Le λ-calcul simplement typé

On ajoute un *système de type* : spécifications pour les λ-termes

$$t : A \to B \quad \approx \quad \text{« } t \text{ est une fonction de } A \text{ dans } B \text{ »}$$

Types simples : engendrés par « $\to$ » à partir d'un type de base o

$$\dfrac{f_a : o \to o \qquad \dfrac{f_b : o \to o \qquad \dfrac{f_b : o \to o \qquad x : o}{f_b\ x : o}}{f_b\ (f_b\ x) : o}}{f_a\ (f_b\ (f_b\ x)) : o}$$

Le λ-calcul simplement typé

On ajoute un *système de type* : spécifications pour les λ-termes

$$t : A \to B \quad \approx \quad \text{« } t \text{ est une fonction de } A \text{ dans } B \text{ »}$$

Types simples : engendrés par « $\to$ » à partir d'un type de base o

$$\dfrac{f_a : o \to o \qquad \dfrac{f_b : o \to o \qquad \dfrac{f_b : o \to o \qquad x : o}{f_b\ x : o}}{f_b\ (f_b\ x) : o}}{f_a\ (f_b\ (f_b\ x)) : o}$$

$$\overline{abb} = \lambda f_a.\ \lambda f_b.\ \lambda x.\ f_a\ (f_b\ (f_b\ x)) : \overbrace{(o \to o) \to (o \to o) \to o \to o}^{\mathsf{Str}_{\{a,b\}}}$$

Le λ-calcul simplement typé

On ajoute un *système de type* : spécifications pour les λ-termes

$$t : A \to B \quad \approx \quad \text{« } t \text{ est une fonction de } A \text{ dans } B \text{ »}$$

Types simples : engendrés par « $\to$ » à partir d'un type de base o

$$\dfrac{f_a : o \to o \qquad \dfrac{f_b : o \to o \qquad \dfrac{f_b : o \to o \qquad x : o}{f_b\ x : o}}{f_b\ (f_b\ x) : o}}{f_a\ (f_b\ (f_b\ x)) : o}$$

$$\overline{abb} = \lambda f_a.\ \lambda f_b.\ \lambda x.\ f_a\ (f_b\ (f_b\ x)) : \overbrace{(o \to o) \to (o \to o) \to o \to o}^{\mathsf{Str}_{\{a,b\}}}$$

Plus généralement, $t : \mathsf{Str}_{\{a,b\}} \iff \exists w \in \{a,b\}^* : t =_{\beta\eta} \overline{w}$

Langages rationnels et types simples

On a aussi des types Str_Σ pour tout alphabet fini Σ, et Bool (booléens)

Langages rationnels et types simples

On a aussi des types Str_Σ pour tout alphabet fini Σ, et Bool (booléens)

Notation : $B[A] = B\{o := A\}$ par ex. $\mathsf{Str}_{\{a,b\}}[A] = (A \to A) \to (A \to A) \to A \to A$

Théorème (Hillebrand & Kanellakis 1996)

Le langage $L \subseteq \Sigma^*$ *est* rationnel $\iff$ *il existe un type simple* A *et* $t : \mathsf{Str}_\Sigma[A] \to \mathsf{Bool}$ *tels que* $\forall w \in \Sigma^*,\ w \in L \Leftrightarrow t\ \overline{w} =_\beta \mathtt{true}$

Langages rationnels : objet central de la théorie des automates (diapo suivante)

Langages rationnels et types simples

On a aussi des types Str_Σ pour tout alphabet fini Σ, et Bool (booléens)

Notation : $B[A] = B\{o := A\}$ par ex. $\mathsf{Str}_{\{a,b\}}[A] = (A \to A) \to (A \to A) \to A \to A$

Théorème (Hillebrand & Kanellakis 1996)

Le langage $L \subseteq \Sigma^*$ *est* rationnel $\iff$ *il existe un type simple* A *et* $t : \mathsf{Str}_\Sigma[A] \to \mathsf{Bool}$ *tels que* $\forall w \in \Sigma^*,\ w \in L \Leftrightarrow t\ \overline{w} =_\beta \mathtt{true}$

Langages rationnels : objet central de la théorie des automates (diapo suivante)

Exemple : $t = \lambda s.\ s\ \mathtt{id}\ \mathtt{not}\ \mathtt{true} : \mathsf{Str}_{\{a,b\}}[\mathsf{Bool}] \to \mathsf{Bool}$ (nombre pair de b)

$$t\ \overline{abb} \longrightarrow_\beta \overline{abb}\ \mathtt{id}\ \mathtt{not}\ \mathtt{true} \longrightarrow_\beta \mathtt{id}\ (\mathtt{not}\ (\mathtt{not}\ \mathtt{true})) \longrightarrow_\beta \mathtt{true}$$

Langages rationnels et types simples

On a aussi des types Str_Σ pour tout alphabet fini Σ, et Bool (booléens)

Notation : $B[A] = B\{o := A\}$ par ex. $\mathsf{Str}_{\{a,b\}}[A] = (A \to A) \to (A \to A) \to A \to A$

Théorème (Hillebrand & Kanellakis 1996)

Le langage $L \subseteq \Sigma^*$ *est* rationnel $\iff$ *il existe un type simple* A *et* $t : \mathsf{Str}_\Sigma[A] \to \mathsf{Bool}$ *tels que* $\forall w \in \Sigma^*,\ w \in L \Leftrightarrow t\ \overline{w} =_\beta \mathtt{true}$

Langages rationnels : objet central de la théorie des automates (diapo suivante)

Exemple : $t = \lambda s.\ s\ \mathtt{id}\ \mathtt{not}\ \mathtt{true} : \mathsf{Str}_{\{a,b\}}[\mathsf{Bool}] \to \mathsf{Bool}$ (nombre pair de b)

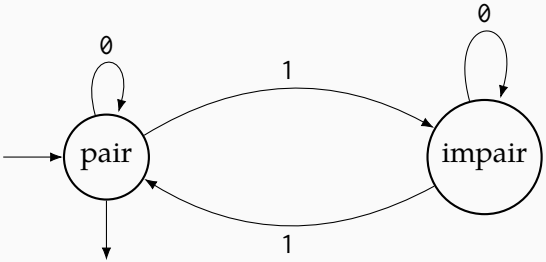
$$t\ \overline{abb} \longrightarrow_\beta \overline{abb}\ \mathtt{id}\ \mathtt{not}\ \mathtt{true} \longrightarrow_\beta \mathtt{id}\ (\mathtt{not}\ (\mathtt{not}\ \mathtt{true})) \longrightarrow_\beta \mathtt{true}$$

C'est le point de départ de cette thèse ! On va étendre ça aux transducteurs

Rappel : langages rationnels

La canonicité / robustesse de la classe des *langages rationnels* (*regular languages*) est attestée par ses nombreuses définitions équivalentes :

- *expressions rationnelles* : `0*(10*10*)*` = « que des `0` et `1` & nombre pair de `1` »
- *automates finis* (non-)déterministes : exemple ci-dessous



Rappel : langages rationnels

La canonicité / robustesse de la classe des *langages rationnels* (*regular languages*) est attestée par ses nombreuses définitions équivalentes :

- *expressions rationnelles* : `0*(10*10*)*` = « que des `0` et `1` & nombre pair de `1` »
- *automates finis* (non-)déterministes
- définition *algébrique* (proche des automates déterministes), ex : $M = \mathbb{Z}/2\mathbb{Z}$

Théorème (classique)

Un langage $L \subseteq \Sigma^*$ *est* rationnel $\iff$ *il existe un* morphisme $\varphi : \Sigma^* \to M$ *vers un* monoïde fini M *et une partie* $P \subseteq M$ *tels que* $L = \varphi^{-1}(P) = \{w \in \Sigma^* \mid \varphi(w) \in P\}$.

Rappel : langages rationnels

La canonicité / robustesse de la classe des *langages rationnels* (*regular languages*) est attestée par ses nombreuses définitions équivalentes :

- *expressions rationnelles* : `0*(10*10*)*` = « que des `0` et `1` & nombre pair de `1` »
- *automates finis* (non-)déterministes
- définition *algébrique* (proche des automates déterministes), ex : $M = \mathbb{Z}/2\mathbb{Z}$

Théorème (classique)

Un langage $L \subseteq \Sigma^*$ *est* rationnel $\iff$ *il existe un* morphisme $\varphi : \Sigma^* \to M$ *vers un* monoïde fini M *et une partie* $P \subseteq M$ *tels que* $L = \varphi^{-1}(P) = \{w \in \Sigma^* \mid \varphi(w) \in P\}$.

Quelle est la bonne généralisation à des fonctions $\Gamma^* \to \Sigma^*$ (*transductions*) ?

Classes habituelles de transductions

Quelle est la bonne généralisation des langages rationnels (en anglais : réguliers) à des *transductions* $\Gamma^* \to \Sigma^*$? Il y en a plusieurs !

$$\underbrace{\text{fonctions séquentielles}}_{\text{transducteurs finis } \textit{déterministes}} \subsetneq \underbrace{\text{fonctions rationnelles}}_{\text{transducteurs } \textit{non-déterministes}} \subsetneq \text{fonctions régulières}$$

Classes habituelles de transductions

Quelle est la bonne généralisation des langages rationnels (en anglais : réguliers) à des *transductions* $\Gamma^* \to \Sigma^*$? Il y en a plusieurs !

$$\underbrace{\text{fonctions séquentielles}}_{\text{transducteurs finis } \textit{déterministes}} \subsetneq \underbrace{\text{fonctions rationnelles}}_{\text{transducteurs } \textit{non-déterministes}} \subsetneq \text{fonctions régulières}$$

- ces classes sont *stables par composition*
- préservation : L rationnel $\implies f^{-1}(L)$ rationnel

Classes habituelles de transductions

Quelle est la bonne généralisation des langages rationnels (en anglais : réguliers)
à des *transductions* $\Gamma^* \to \Sigma^*$? Il y en a plusieurs !

$$\underbrace{\text{fonctions séquentielles}}_{\text{transducteurs finis } \textit{déterministes}} \subsetneq \underbrace{\text{fonctions rationnelles}}_{\text{transducteurs } \textit{non-déterministes}} \subsetneq \text{fonctions régulières}$$

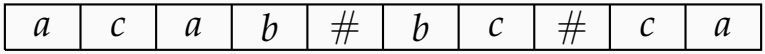
- ces classes sont *stables par composition*
- préservation : L rationnel $\implies f^{-1}(L)$ rationnel

On s'intéresse ici aux régulières ; plein de définitions (c'est canonique !),
par ex. *transducteurs à registres* (*streaming string transducers*)

Transducteurs à registres [Alur & Černý 2010]

Automate fini déterministe + *registres* contenant des mots. Exemple :

$$\begin{array}{lrcl} \texttt{mapReverse}: & \{a,b,c,\#\}^* & \to & \{a,b,c,\#\}^* \\ & w_1\#\ldots\#w_n & \mapsto & \texttt{reverse}(w_1)\#\ldots\#\texttt{reverse}(w_n) \end{array}$$

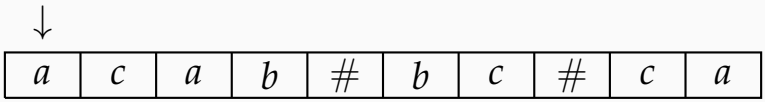


$$X = \varepsilon \qquad Y = \varepsilon$$

Transducteurs à registres [Alur & Černý 2010]

Automate fini déterministe + *registres* contenant des mots. Exemple :

$$\begin{array}{lrcl} \texttt{mapReverse}: & \{a,b,c,\#\}^* & \to & \{a,b,c,\#\}^* \\ & w_1\#\ldots\#w_n & \mapsto & \texttt{reverse}(w_1)\#\ldots\#\texttt{reverse}(w_n) \end{array}$$

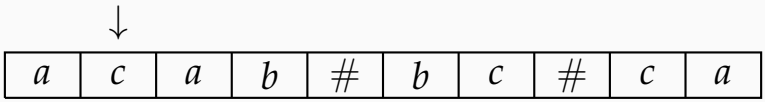


$X = a \qquad Y = \varepsilon$

Transducteurs à registres [Alur & Černý 2010]

Automate fini déterministe + *registres* contenant des mots. Exemple :

$$\begin{array}{lrcl} \texttt{mapReverse}: & \{a,b,c,\#\}^* & \to & \{a,b,c,\#\}^* \\ & w_1\#\ldots\#w_n & \mapsto & \texttt{reverse}(w_1)\#\ldots\#\texttt{reverse}(w_n) \end{array}$$

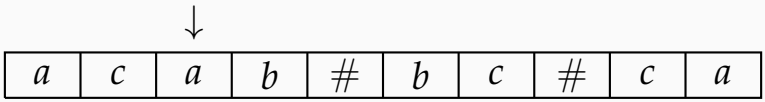


$$X = ca \qquad Y = \varepsilon$$

Transducteurs à registres [Alur & Černý 2010]

Automate fini déterministe + *registres* contenant des mots. Exemple :

$$\begin{array}{lrcl} \texttt{mapReverse}: & \{a,b,c,\#\}^* & \to & \{a,b,c,\#\}^* \\ & w_1\#\ldots\#w_n & \mapsto & \texttt{reverse}(w_1)\#\ldots\#\texttt{reverse}(w_n) \end{array}$$

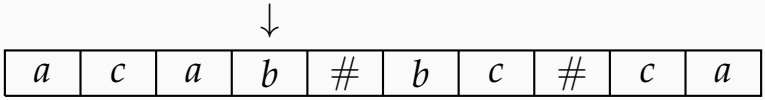


$$X = aca \qquad Y = \varepsilon$$

Transducteurs à registres [Alur & Černý 2010]

Automate fini déterministe + *registres* contenant des mots. Exemple :

$$\begin{array}{lrcl} \texttt{mapReverse}: & \{a,b,c,\#\}^* & \to & \{a,b,c,\#\}^* \\ & w_1\#\ldots\#w_n & \mapsto & \texttt{reverse}(w_1)\#\ldots\#\texttt{reverse}(w_n) \end{array}$$

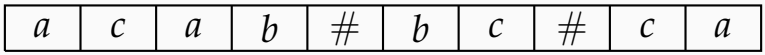


$$X = baca \qquad Y = \varepsilon$$

Transducteurs à registres [Alur & Černý 2010]

Automate fini déterministe + *registres* contenant des mots. Exemple :

$$\begin{array}{lrcl} \texttt{mapReverse}: & \{a,b,c,\#\}^* & \to & \{a,b,c,\#\}^* \\ & w_1\#\ldots\#w_n & \mapsto & \texttt{reverse}(w_1)\#\ldots\#\texttt{reverse}(w_n) \end{array}$$

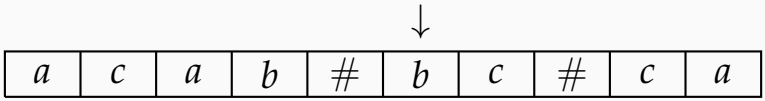


$$X = \varepsilon \qquad Y = baca\#$$

Transducteurs à registres [Alur & Černý 2010]

Automate fini déterministe + *registres* contenant des mots. Exemple :

$$\begin{array}{lrcl} \texttt{mapReverse}: & \{a,b,c,\#\}^* & \to & \{a,b,c,\#\}^* \\ & w_1\#\ldots\#w_n & \mapsto & \texttt{reverse}(w_1)\#\ldots\#\texttt{reverse}(w_n) \end{array}$$

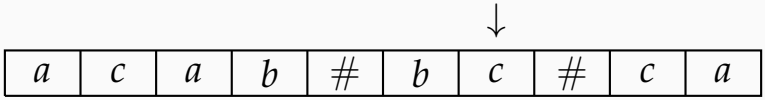


$X = b \qquad Y = baca\#$

Transducteurs à registres [Alur & Černý 2010]

Automate fini déterministe + *registres* contenant des mots. Exemple :

$$\begin{array}{lrcl} \texttt{mapReverse}: & \{a,b,c,\#\}^* & \to & \{a,b,c,\#\}^* \\ & w_1\#\ldots\#w_n & \mapsto & \texttt{reverse}(w_1)\#\ldots\#\texttt{reverse}(w_n) \end{array}$$

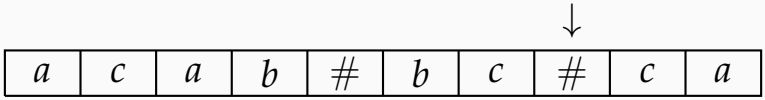


$X = cb \qquad Y = baca\#$

Transducteurs à registres [Alur & Černý 2010]

Automate fini déterministe + *registres* contenant des mots. Exemple :

$$\begin{array}{lrcl} \texttt{mapReverse}: & \{a,b,c,\#\}^* & \to & \{a,b,c,\#\}^* \\ & w_1\#\ldots\#w_n & \mapsto & \texttt{reverse}(w_1)\#\ldots\#\texttt{reverse}(w_n) \end{array}$$

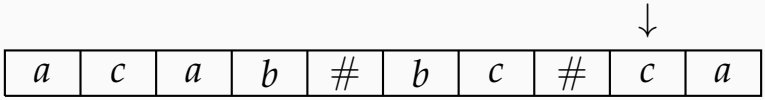


$$X = \varepsilon \qquad Y = baca\#cb\#$$

Transducteurs à registres [Alur & Černý 2010]

Automate fini déterministe + *registres* contenant des mots. Exemple :

$$\begin{array}{lrcl} \texttt{mapReverse}: & \{a,b,c,\#\}^* & \to & \{a,b,c,\#\}^* \\ & w_1\#\ldots\#w_n & \mapsto & \texttt{reverse}(w_1)\#\ldots\#\texttt{reverse}(w_n) \end{array}$$

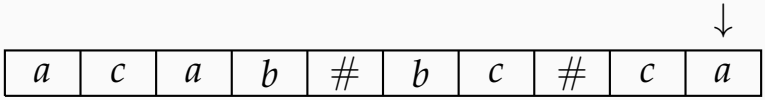


$$X = c \qquad Y = baca\#cb\#$$

Transducteurs à registres [Alur & Černý 2010]

Automate fini déterministe + *registres* contenant des mots. Exemple :

$$\begin{array}{lrcl} \texttt{mapReverse}: & \{a,b,c,\#\}^* & \to & \{a,b,c,\#\}^* \\ & w_1\#\ldots\#w_n & \mapsto & \texttt{reverse}(w_1)\#\ldots\#\texttt{reverse}(w_n) \end{array}$$

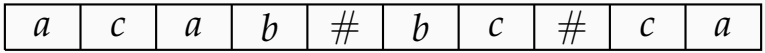


$$X = ac \qquad Y = baca\#cb\#$$

Transducteurs à registres [Alur & Černý 2010]

Automate fini déterministe + *registres* contenant des mots. Exemple :

$$\begin{array}{lrcl} \texttt{mapReverse}: & \{a,b,c,\#\}^* & \to & \{a,b,c,\#\}^* \\ & w_1\#\ldots\#w_n & \mapsto & \texttt{reverse}(w_1)\#\ldots\#\texttt{reverse}(w_n) \end{array}$$



$X = ac \qquad Y = baca\#cb\# \qquad \texttt{mapReverse}(\ldots) = YX = baca\#cb\#ac$

Transducteurs à registres [Alur & Černý 2010]

Automate fini déterministe + *registres* contenant des mots. Exemple :

$$\begin{array}{rlll} \texttt{mapReverse}: & \{a,b,c,\#\}^* & \to & \{a,b,c,\#\}^* \\ & w_1\#\ldots\#w_n & \mapsto & \texttt{reverse}(w_1)\#\ldots\#\texttt{reverse}(w_n) \end{array}$$

a	c	a	b	$\#$	b	c	$\#$	c	a

$X = ac$ $\quad Y = baca\#cb\#$ $\quad \texttt{mapReverse}(\ldots) = YX = baca\#cb\#ac$

Fonctions régulières = calculées par transducteurs sans copie

$$a \mapsto \begin{cases} X := aX \\ Y := Y \end{cases} \qquad \# \mapsto \begin{cases} X := \varepsilon \\ Y := YX\# \end{cases}$$

chaque registre apparaît $\leqslant 1$ fois à droite d'un $:=$ dans une transition

Linéarité

Fonctions régulières = calculées par transducteurs *sans copie*

Restriction du « pouvoir de copie » : vieux thème de la théorie des automates

(par ex. Schützenberger, années 1960 : séries $\Sigma^* \to \mathbb{Z}$ rationnelles à coefficients $|w|^{O(1)}$)

Linéarité

Fonctions régulières = calculées par transducteurs *sans copie*

Restriction du « pouvoir de copie » : vieux thème de la théorie des automates

(par ex. Schützenberger, années 1960 : séries $\Sigma^* \to \mathbb{Z}$ rationnelles à coefficients $|w|^{O(1)}$)

Équivalent λ-calcul : typage linéaire (Girard 1987)

- $t : A \to B = t$ utilise son argument 0, 1 ou plusieurs fois
- $t : A \multimap B = t$ utilise son argument *une seule fois* par ex. $t = \lambda x.\ (\overbrace{\text{bla bla bla}}^{\text{où } x \text{ apparaît 1 seule fois}})$

Linéarité

Fonctions régulières = calculées par transducteurs *sans copie*

Restriction du « pouvoir de copie » : vieux thème de la théorie des automates

(par ex. Schützenberger, années 1960 : séries $\Sigma^* \to \mathbb{Z}$ rationnelles à coefficients $|w|^{O(1)}$)

Équivalent λ-calcul : typage linéaire (Girard 1987)

- $t : A \to B = t$ utilise son argument 0, 1 ou plusieurs fois
- $t : A \multimap B = t$ utilise son argument *une seule fois* par ex. $t = \lambda x.\ (\overbrace{\text{bla bla bla}}^{\text{où } x \text{ apparaît 1 seule fois}})$

Cette thèse utilise le « $\lambda\ell^{\oplus\&}$-calcul » = *Dual Intuitionistic Linear Logic*
+ connecteurs additifs $\oplus, \&$

Linéarité

Fonctions régulières = calculées par transducteurs *sans copie*
Restriction du « pouvoir de copie » : vieux thème de la théorie des automates
(par ex. Schützenberger, années 1960 : séries $\Sigma^* \to \mathbb{Z}$ rationnelles à coefficients $|w|^{O(1)}$)

Équivalent λ-calcul : typage linéaire (Girard 1987)

- $t : A \to B = t$ utilise son argument 0, 1 ou plusieurs fois
- $t : A \multimap B = t$ utilise son argument *une seule fois* par ex. $t = \lambda x.\ (\overbrace{\text{bla bla bla}}^{\text{où } x \text{ apparaît 1 seule fois}})$

Cette thèse utilise le « $\lambda\ell^{\oplus\&}$-calcul » = *Dual Intuitionistic Linear Logic*
+ connecteurs additifs $\oplus, \&$

(linéaire = *exactement* 1 fois ; les additifs permettent de simuler *au plus* 1 fois (affine))

Premier résultat de transducteurs implicites

Codages de Church linéaires

$\mathsf{Str}_{\{a,b\}} = (o \multimap o) \to (o \multimap o) \to (o \multimap o)$; *mutatis mutandis* pour Str_Σ

Définition : un type du $\lambda\ell^{\oplus\&}$-calcul est *purement linéaire* s'il ne contient pas « $\to$ »

Théorème (cette thèse)

$f : \Gamma^* \to \Sigma^*$ *est* régulière $\iff$ $\exists$ *un type* A purement linéaire *et* $t : \mathsf{Str}_\Gamma[A] \multimap \mathsf{Str}_\Sigma$ *en* $\lambda\ell^{\oplus\&}$*-calcul tels que* $\forall w \in \Gamma^*$, $\overline{f(w)} =_\beta t\ \overline{w}$

Premier résultat de transducteurs implicites

Codages de Church linéaires

$\mathsf{Str}_{\{a,b\}} = (o \multimap o) \to (o \multimap o) \to (o \multimap o)$; *mutatis mutandis* pour Str_Σ

Définition : un type du $\lambda\ell^{\oplus\&}$-calcul est *purement linéaire* s'il ne contient pas « $\to$ »

Théorème (cette thèse)

$f : \Gamma^* \to \Sigma^*$ *est* régulière $\iff$ $\exists$ *un type* A purement linéaire *et* $t : \mathsf{Str}_\Gamma[A] \multimap \mathsf{Str}_\Sigma$ *en* $\lambda\ell^{\oplus\&}$*-calcul tels que* $\forall w \in \Gamma^*,\ \overline{f(w)} =_\beta t\ \overline{w}$

Et si on remplace $\mathsf{Str}_\Gamma[A] \multimap \mathsf{Str}_\Sigma$ par $\mathsf{Str}_\Gamma[A] \to \mathsf{Str}_\Sigma$, quelle classe de fonctions ?

Premier résultat de transducteurs implicites

Codages de Church linéaires

$\mathsf{Str}_{\{a,b\}} = (o \multimap o) \to (o \multimap o) \to (o \multimap o)$; *mutatis mutandis* pour Str_Σ

Définition : un type du $\lambda\ell^{\oplus\&}$-calcul est *purement linéaire* s'il ne contient pas « $\to$ »

Théorème (cette thèse)

$f : \Gamma^* \to \Sigma^*$ *est* régulière $\iff$ $\exists$ *un type* A purement linéaire *et* $t : \mathsf{Str}_\Gamma[A] \multimap \mathsf{Str}_\Sigma$ *en* $\lambda\ell^{\oplus\&}$*-calcul tels que* $\forall w \in \Gamma^*$, $\overline{f(w)} =_\beta t\ \overline{w}$

Et si on remplace $\mathsf{Str}_\Gamma[A] \multimap \mathsf{Str}_\Sigma$ par $\mathsf{Str}_\Gamma[A] \to \mathsf{Str}_\Sigma$, quelle classe de fonctions ?

- Mauvaise nouvelle : cette classe n'existait pas encore dans la littérature

Premier résultat de transducteurs implicites

Codages de Church linéaires

$\mathsf{Str}_{\{a,b\}} = (o \multimap o) \to (o \multimap o) \to (o \multimap o)$; *mutatis mutandis* pour Str_Σ

Définition : un type du $\lambda\ell^{\oplus\&}$-calcul est *purement linéaire* s'il ne contient pas « $\to$ »

Théorème (cette thèse)

$f : \Gamma^* \to \Sigma^*$ *est* régulière $\iff$ $\exists$ *un type* A purement linéaire *et* $t : \mathsf{Str}_\Gamma[A] \multimap \mathsf{Str}_\Sigma$ *en* $\lambda\ell^{\oplus\&}$*-calcul tels que* $\forall w \in \Gamma^*$, $\overline{f(w)} =_\beta t\ \overline{w}$

Et si on remplace $\mathsf{Str}_\Gamma[A] \multimap \mathsf{Str}_\Sigma$ par $\mathsf{Str}_\Gamma[A] \to \mathsf{Str}_\Sigma$, quelle classe de fonctions ?

- Mauvaise nouvelle : cette classe n'existait pas encore dans la littérature
- Bonne nouvelle : elle est naturelle, surtout au vu des travaux récents sur les *fonctions polyrégulières* [Bojańczyk 2018]

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*

$\triangleright$	a	b	c	$\#$	b	a	c	$\#$	c	a	$\triangleleft$

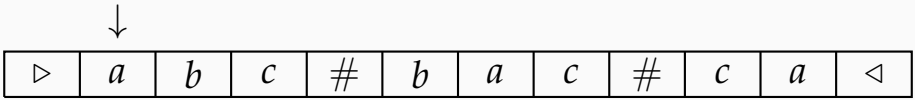
Sortie :

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



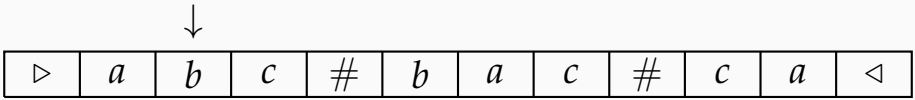
Sortie :

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



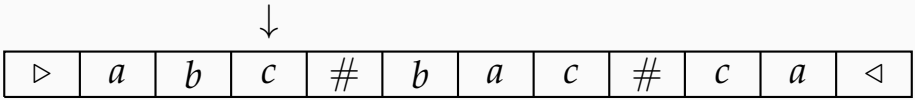
Sortie :

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



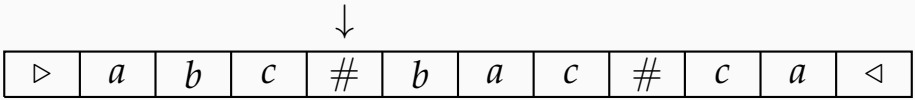
Sortie :

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



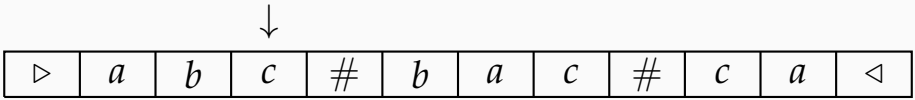
Sortie :

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



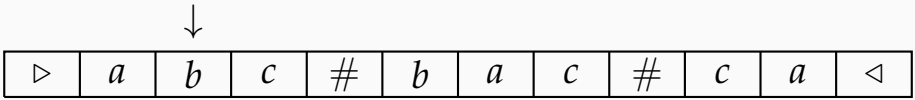
Sortie : c

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



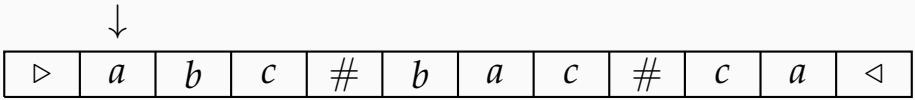
Sortie : cb

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



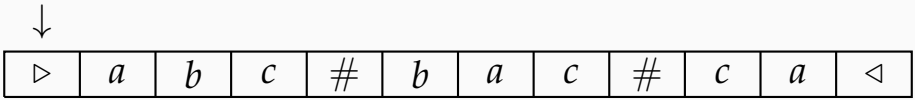
Sortie : *cba*

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



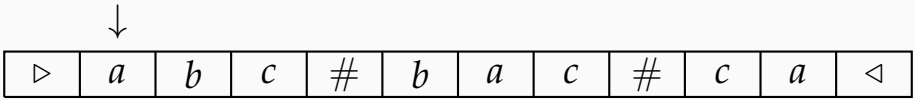
Sortie : *cba*

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



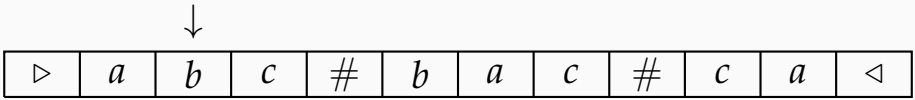
Sortie : *cba*

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



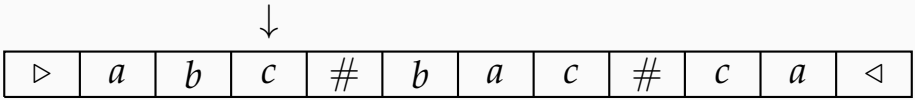
Sortie : *cba*

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



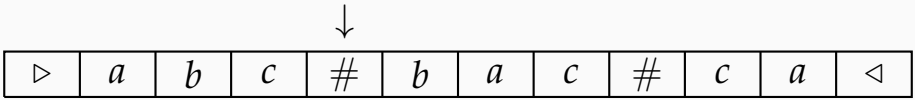
Sortie : *cba*

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



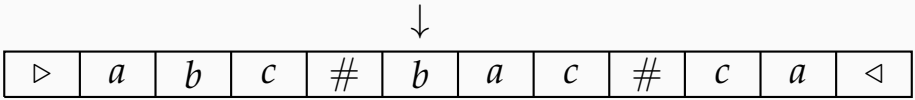
Sortie : $cba\#$

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



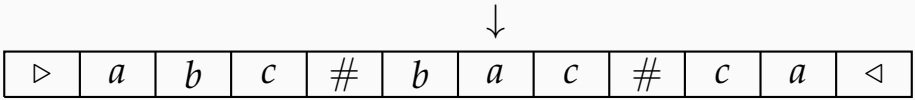
Sortie : $cba\#$

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



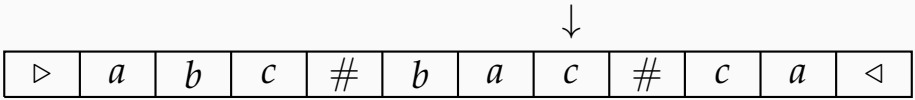
Sortie : $cba\#$

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



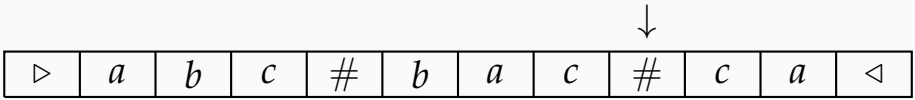
Sortie : $cba\#$

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



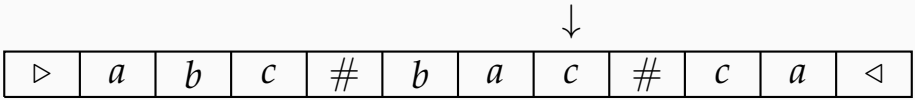
Sortie : $cba\#$

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



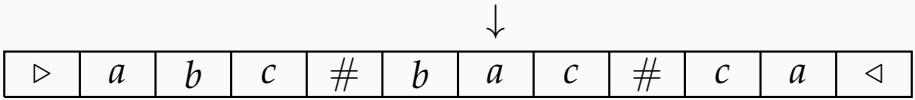
Sortie : $cba\#c$

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



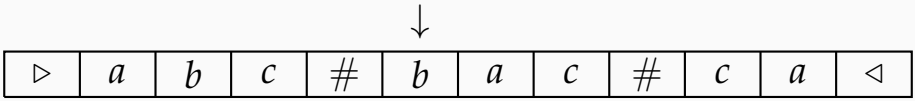
Sortie : $cba\#ca$

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



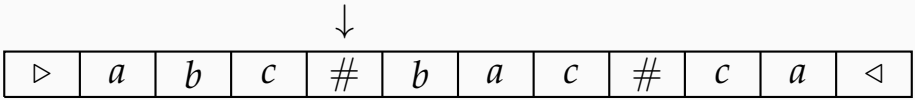
Sortie : $cba\#cab$

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



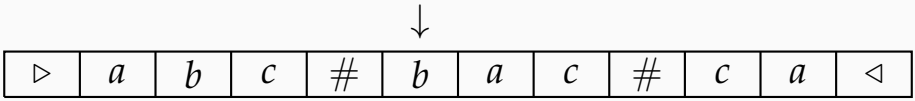
Sortie : $cba\#cab$

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



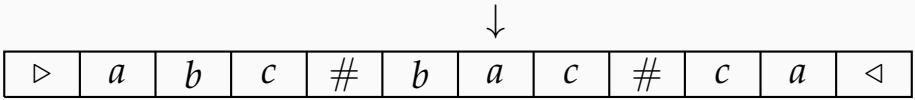
Sortie : $cba\#cab$

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



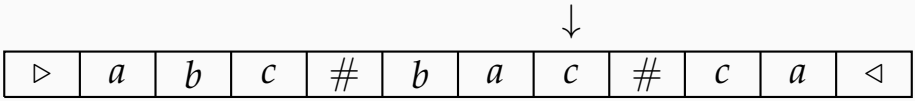
Sortie : $cba\#cab$

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



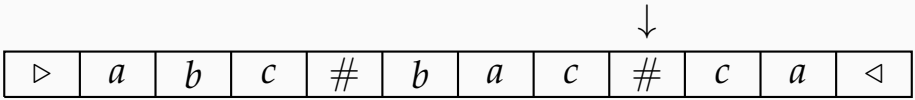
Sortie : $cba\#cab$

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



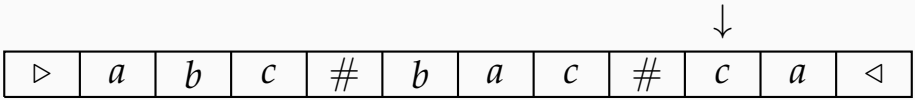
Sortie : $cba\#cab\#$

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



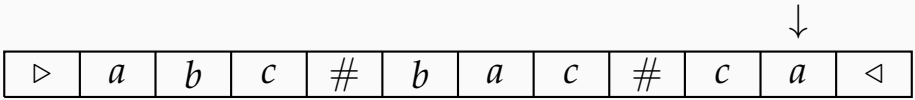
Sortie : $cba\#cab\#$

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



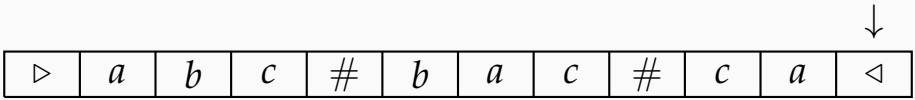
Sortie : $cba\#cab\#$

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



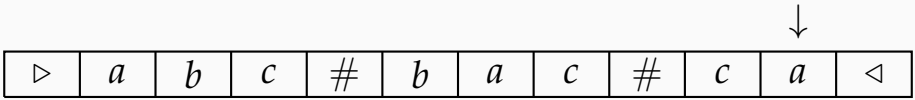
Sortie : $cba\#cab\#$

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



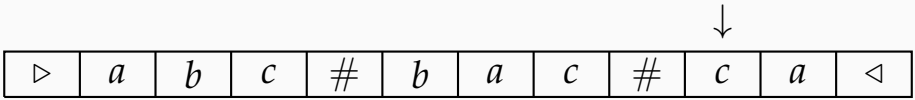
Sortie : $cba\#cab\#a$

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



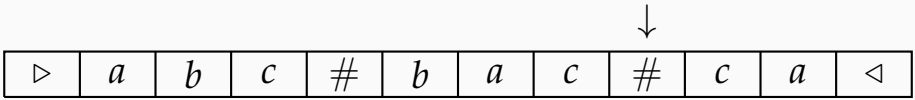
Sortie : $cba\#cab\#ac$

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



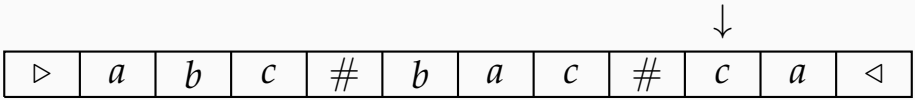
Sortie : $cba\#cab\#ac$

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



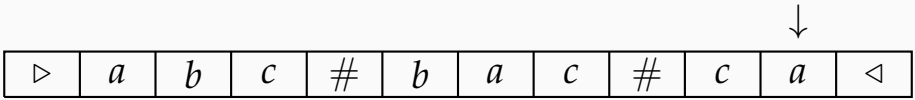
Sortie : $cba\#cab\#ac$

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



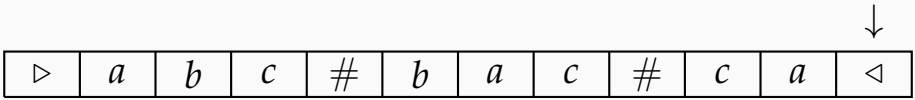
Sortie : $cba\#cab\#ac$

Un prérequis pour introduire les fonctions polyrégulières...

Transducteurs bidirectionnels (ou « boustrophédon » / EN : two-way)

Tête de lecture pouvant bouger à gauche ou à droite à chaque transition

$\longrightarrow$ autre définition équivalente des *fonctions régulières*



Sortie : $cba\#cab\#ac$

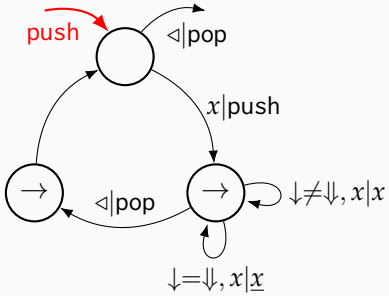
Transducteurs à jetons[1] (EN : pebble / PL : kamyki) [Milo, Suciu & Vianu 2000]

Fonctions polyrégulières = calculées par des transducteurs à k jetons

Ensemble fini d'états + *pile* de hauteur $\leqslant k$ de têtes bidirectionnelles («jetons»)

Les positions des têtes peuvent être *comparées*. Exemple :

$$\begin{array}{ccc} \Sigma^* & \to & (\Sigma \cup \underline{\Sigma})^* \\ abc & \mapsto & \underline{a}bca\underline{b}cab\underline{c} \end{array}$$



▷	a	b	c	◁

Sortie :

[1] https://fr.wikipedia.org/wiki/Automate_à_jetons

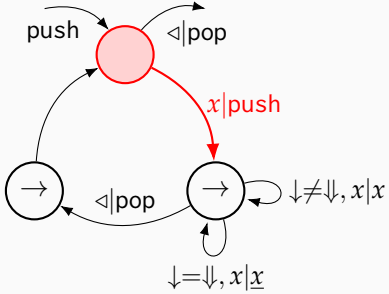
Transducteurs à jetons[1] (EN : pebble / PL : kamyki) [Milo, Suciu & Vianu 2000]

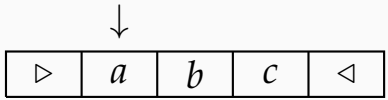
Fonctions polyrégulières = calculées par des transducteurs à k jetons

Ensemble fini d'états + *pile* de hauteur $\leqslant k$ de têtes bidirectionnelles («jetons»)

Les positions des têtes peuvent être *comparées*. Exemple :

$$\begin{array}{ccc} \Sigma^* & \to & (\Sigma \cup \underline{\Sigma})^* \\ abc & \mapsto & \underline{a}bca\underline{b}cab\underline{c} \end{array}$$





Sortie :

[1] https://fr.wikipedia.org/wiki/Automate_à_jetons

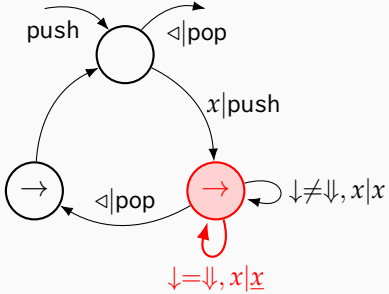
Transducteurs à jetons[1] (EN : pebble / PL : kamyki) [Milo, Suciu & Vianu 2000]

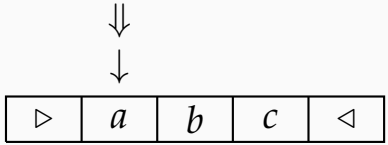
Fonctions polyrégulières = calculées par des transducteurs à k jetons

Ensemble fini d'états + *pile* de hauteur $\leqslant k$ de têtes bidirectionnelles («jetons»)

Les positions des têtes peuvent être *comparées*. Exemple :

$$\begin{array}{ccc} \Sigma^* & \to & (\Sigma \cup \underline{\Sigma})^* \\ abc & \mapsto & \underline{a}bca\underline{b}ca\underline{bc} \end{array}$$





Sortie :

[1] https://fr.wikipedia.org/wiki/Automate_à_jetons

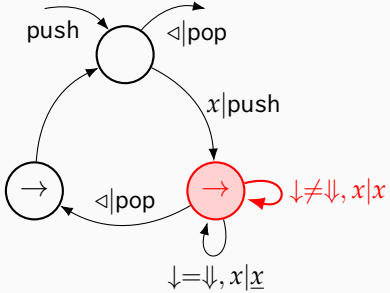
Transducteurs à jetons[1] (EN : pebble / PL : kamyki) [Milo, Suciu & Vianu 2000]

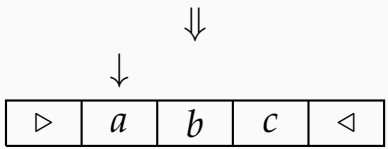
Fonctions polyrégulières = calculées par des transducteurs à k jetons

Ensemble fini d'états + *pile* de hauteur $\leqslant k$ de têtes bidirectionnelles («jetons»)

Les positions des têtes peuvent être *comparées*. Exemple :

$$\begin{array}{ccc} \Sigma^* & \to & (\Sigma \cup \underline{\Sigma})^* \\ abc & \mapsto & \underline{a}bca\underline{b}cab\underline{c} \end{array}$$





Sortie : $\underline{a}$

[1] https://fr.wikipedia.org/wiki/Automate_à_jetons

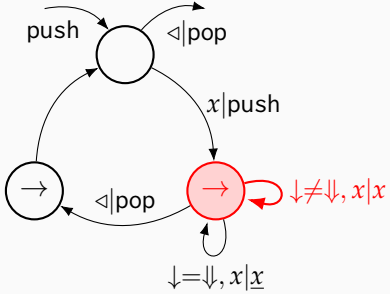
Transducteurs à jetons[1] (EN : pebble / PL : kamyki) [Milo, Suciu & Vianu 2000]

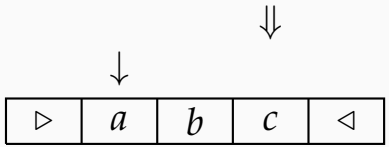
Fonctions polyrégulières = calculées par des transducteurs à k jetons

Ensemble fini d'états + *pile* de hauteur $\leqslant k$ de têtes bidirectionnelles («jetons»)

Les positions des têtes peuvent être *comparées*. Exemple :

$$\begin{array}{ccc} \Sigma^* & \to & (\Sigma \cup \underline{\Sigma})^* \\ abc & \mapsto & \underline{a}bca\underline{b}cab\underline{c} \end{array}$$





Sortie : $\underline{a}b$

[1] https://fr.wikipedia.org/wiki/Automate_à_jetons

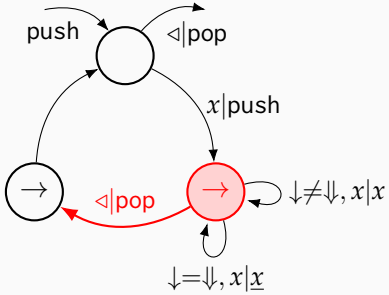
Transducteurs à jetons[1] (EN : pebble / PL : kamyki) [Milo, Suciu & Vianu 2000]

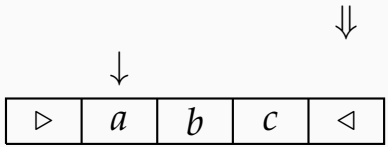
Fonctions polyrégulières = calculées par des transducteurs à k jetons

Ensemble fini d'états + *pile* de hauteur $\leqslant k$ de têtes bidirectionnelles («jetons»)

Les positions des têtes peuvent être *comparées*. Exemple :

$$\begin{array}{ccc} \Sigma^* & \to & (\Sigma \cup \underline{\Sigma})^* \\ abc & \mapsto & \underline{a}bc\underline{a}\underline{b}ca\underline{b}\underline{c} \end{array}$$





Sortie : $\underline{a}bc$

[1] https://fr.wikipedia.org/wiki/Automate_à_jetons

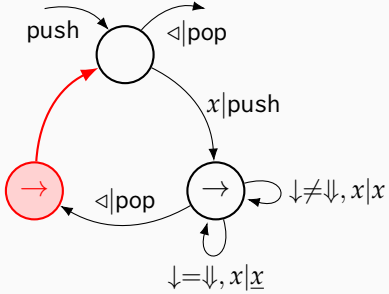
Transducteurs à jetons[1] (EN : pebble / PL : kamyki) [Milo, Suciu & Vianu 2000]

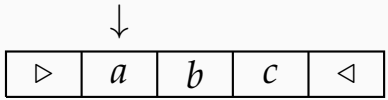
Fonctions polyrégulières = calculées par des transducteurs à k jetons

Ensemble fini d'états + *pile* de hauteur $\leqslant k$ de têtes bidirectionnelles («jetons»)

Les positions des têtes peuvent être *comparées*. Exemple :

$$\begin{array}{ccc} \Sigma^* & \to & (\Sigma \cup \underline{\Sigma})^* \\ abc & \mapsto & \underline{a}bc\underline{a}\underline{b}ca\underline{b}\underline{c} \end{array}$$





Sortie : $\underline{a}bc$

[1] https://fr.wikipedia.org/wiki/Automate_à_jetons

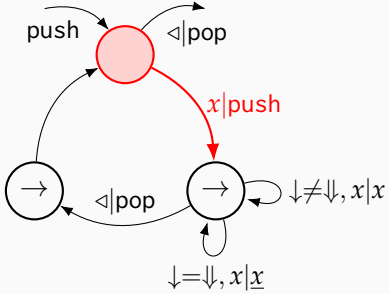
Transducteurs à jetons[1] (EN : pebble / PL : kamyki) [Milo, Suciu & Vianu 2000]

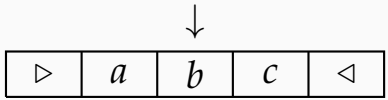
Fonctions polyrégulières = calculées par des transducteurs à k jetons

Ensemble fini d'états + *pile* de hauteur $\leqslant k$ de têtes bidirectionnelles («jetons»)

Les positions des têtes peuvent être *comparées*. Exemple :

$$\begin{array}{ccc} \Sigma^* & \to & (\Sigma \cup \underline{\Sigma})^* \\ abc & \mapsto & \underline{a}bca\underline{b}cab\underline{c} \end{array}$$





Sortie : $\underline{a}bc$

[1] https://fr.wikipedia.org/wiki/Automate_à_jetons

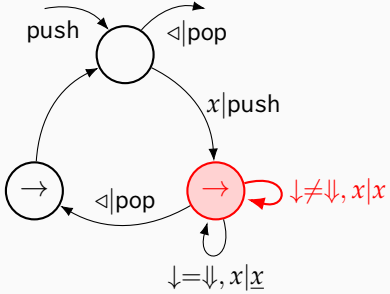
Transducteurs à jetons[1] (EN : pebble / PL : kamyki) [Milo, Suciu & Vianu 2000]

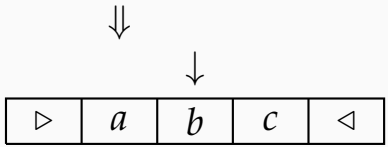
Fonctions polyrégulières = calculées par des transducteurs à k jetons

Ensemble fini d'états + *pile* de hauteur $\leqslant k$ de têtes bidirectionnelles («jetons»)

Les positions des têtes peuvent être *comparées*. Exemple :

$$\begin{array}{ccc} \Sigma^* & \to & (\Sigma \cup \underline{\Sigma})^* \\ abc & \mapsto & \underline{a}bca\underline{b}cab\underline{c} \end{array}$$





Sortie : $\underline{a}bc$

[1] https://fr.wikipedia.org/wiki/Automate_à_jetons

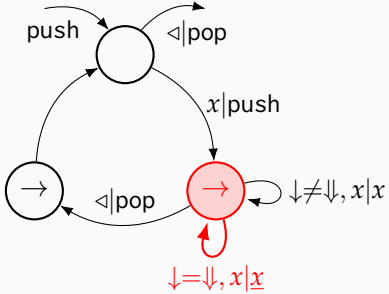
Transducteurs à jetons[1] (EN : pebble / PL : kamyki) [Milo, Suciu & Vianu 2000]

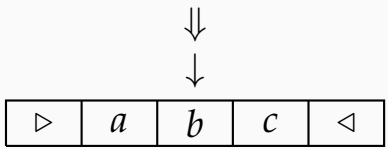
Fonctions polyrégulières = calculées par des transducteurs à k jetons

Ensemble fini d'états + *pile* de hauteur $\leqslant k$ de têtes bidirectionnelles («jetons»)

Les positions des têtes peuvent être *comparées*. Exemple :

$$\begin{array}{ccc} \Sigma^* & \to & (\Sigma \cup \underline{\Sigma})^* \\ abc & \mapsto & \underline{a}bca\underline{b}cab\underline{c} \end{array}$$





Sortie : $\underline{a}bca$

[1] https://fr.wikipedia.org/wiki/Automate_à_jetons

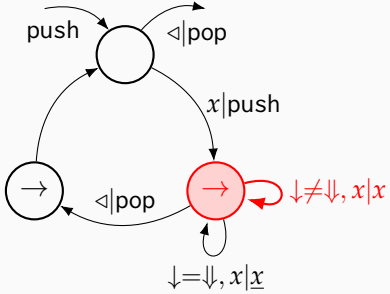
Transducteurs à jetons[1] (EN : pebble / PL : kamyki) [Milo, Suciu & Vianu 2000]

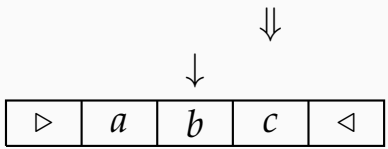
Fonctions polyrégulières = calculées par des transducteurs à k jetons

Ensemble fini d'états + *pile* de hauteur $\leqslant k$ de têtes bidirectionnelles («jetons»)

Les positions des têtes peuvent être *comparées*. Exemple :

$$\begin{array}{ccc} \Sigma^* & \to & (\Sigma \cup \underline{\Sigma})^* \\ abc & \mapsto & \underline{a}bca\underline{b}ca\underline{bc} \end{array}$$





Sortie : $\underline{a}bca\underline{b}$

[1] https://fr.wikipedia.org/wiki/Automate_à_jetons

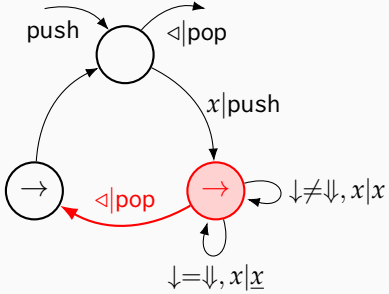
Transducteurs à jetons[1] (EN : pebble / PL : kamyki) [Milo, Suciu & Vianu 2000]

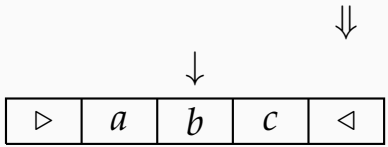
Fonctions polyrégulières = calculées par des transducteurs à k jetons

Ensemble fini d'états + *pile* de hauteur $\leqslant k$ de têtes bidirectionnelles («jetons»)

Les positions des têtes peuvent être *comparées*. Exemple :

$$\begin{array}{ccc} \Sigma^* & \to & (\Sigma \cup \underline{\Sigma})^* \\ abc & \mapsto & \underline{a}bca\underline{b}cab\underline{c} \end{array}$$





Sortie : $\underline{a}bca\underline{b}c$

[1] https://fr.wikipedia.org/wiki/Automate_à_jetons

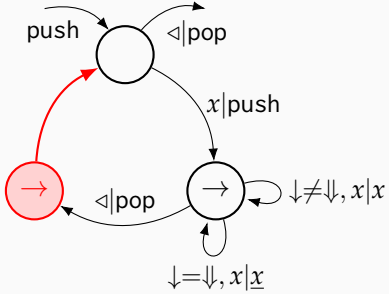
Transducteurs à jetons[1] (EN : pebble / PL : kamyki) [Milo, Suciu & Vianu 2000]

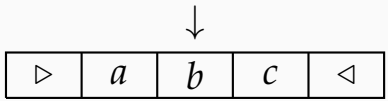
Fonctions polyrégulières = calculées par des transducteurs à k jetons

Ensemble fini d'états + *pile* de hauteur $\leqslant k$ de têtes bidirectionnelles («jetons»)

Les positions des têtes peuvent être *comparées*. Exemple :

$$\begin{array}{ccc} \Sigma^* & \to & (\Sigma \cup \underline{\Sigma})^* \\ abc & \mapsto & \underline{a}bc\underline{a}\underline{b}c\underline{a}\underline{b}\underline{c} \end{array}$$





Sortie : $\underline{a}bc\underline{a}\underline{b}c$

[1] https://fr.wikipedia.org/wiki/Automate_à_jetons

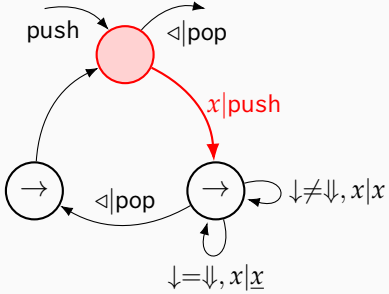
Transducteurs à jetons[1] (EN : pebble / PL : kamyki) [Milo, Suciu & Vianu 2000]

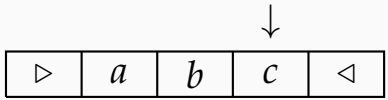
Fonctions polyrégulières = calculées par des transducteurs à k jetons

Ensemble fini d'états + *pile* de hauteur $\leqslant k$ de têtes bidirectionnelles («jetons»)

Les positions des têtes peuvent être *comparées*. Exemple :

$$\begin{array}{ccc} \Sigma^* & \to & (\Sigma \cup \underline{\Sigma})^* \\ abc & \mapsto & \underline{a}bca\underline{b}ca\underline{bc} \end{array}$$





Sortie : $\underline{a}bca\underline{b}c$

[1] https://fr.wikipedia.org/wiki/Automate_à_jetons

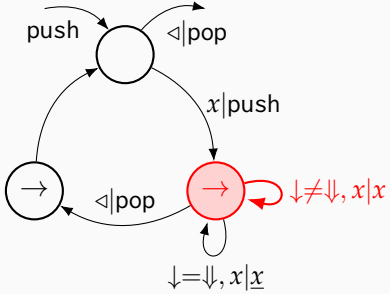
Transducteurs à jetons[1] (EN : pebble / PL : kamyki) [Milo, Suciu & Vianu 2000]

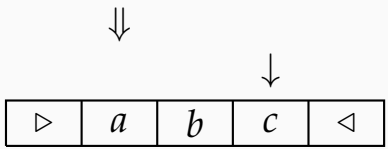
Fonctions polyrégulières = calculées par des transducteurs à k jetons

Ensemble fini d'états + *pile* de hauteur $\leqslant k$ de têtes bidirectionnelles («jetons»)

Les positions des têtes peuvent être *comparées*. Exemple :

$$\begin{array}{ccc} \Sigma^* & \to & (\Sigma \cup \underline{\Sigma})^* \\ abc & \mapsto & \underline{a}bca\underline{b}ca\underline{bc} \end{array}$$





Sortie : $\underline{a}bca\underline{b}c$

[1] https://fr.wikipedia.org/wiki/Automate_à_jetons

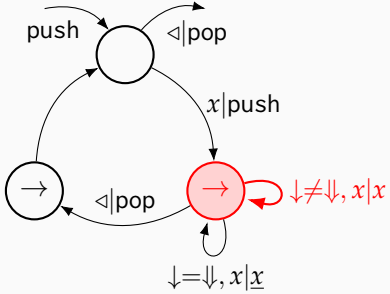
Transducteurs à jetons[1] (EN : pebble / PL : kamyki) [Milo, Suciu & Vianu 2000]

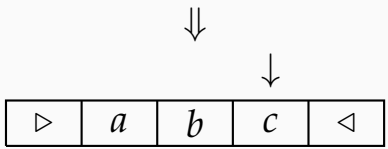
Fonctions polyrégulières = calculées par des transducteurs à k jetons

Ensemble fini d'états + *pile* de hauteur $\leqslant k$ de têtes bidirectionnelles («jetons»)

Les positions des têtes peuvent être *comparées*. Exemple :

$$\begin{array}{ccc} \Sigma^* & \to & (\Sigma \cup \underline{\Sigma})^* \\ abc & \mapsto & \underline{a}bca\underline{b}ca\underline{bc} \end{array}$$





Sortie : $\underline{a}bca\underline{b}ca$

[1] https://fr.wikipedia.org/wiki/Automate_à_jetons

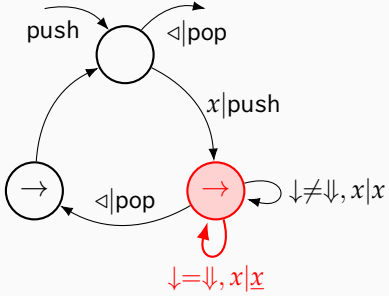
Transducteurs à jetons[1] (EN : pebble / PL : kamyki) [Milo, Suciu & Vianu 2000]

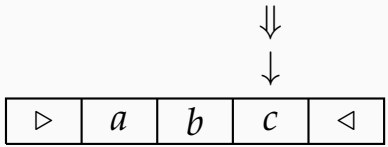
Fonctions polyrégulières = calculées par des transducteurs à k jetons

Ensemble fini d'états + *pile* de hauteur $\leqslant k$ de têtes bidirectionnelles («jetons»)

Les positions des têtes peuvent être *comparées*. Exemple :

$$\begin{array}{ccc} \Sigma^* & \to & (\Sigma \cup \underline{\Sigma})^* \\ abc & \mapsto & \underline{a}bca\underline{b}cab\underline{c} \end{array}$$





Sortie : $\underline{a}bca\underline{b}cab$

[1] https://fr.wikipedia.org/wiki/Automate_à_jetons

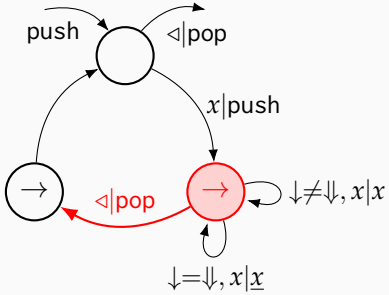
Transducteurs à jetons[1] (EN : pebble / PL : kamyki) [Milo, Suciu & Vianu 2000]

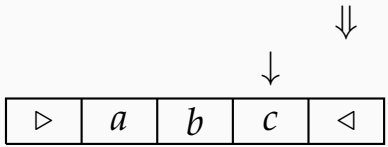
Fonctions polyrégulières = calculées par des transducteurs à k jetons

Ensemble fini d'états + *pile* de hauteur $\leqslant k$ de têtes bidirectionnelles («jetons»)

Les positions des têtes peuvent être *comparées*. Exemple :

$$\begin{array}{ccc} \Sigma^* & \to & (\Sigma \cup \underline{\Sigma})^* \\ abc & \mapsto & \underline{a}bca\underline{b}cab\underline{c} \end{array}$$





Sortie : $\underline{a}bca\underline{b}cab\underline{c}$

[1] https://fr.wikipedia.org/wiki/Automate_à_jetons

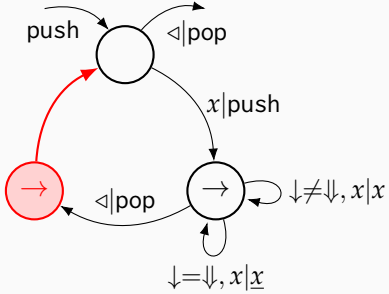
Transducteurs à jetons[1] (EN : pebble / PL : kamyki) [Milo, Suciu & Vianu 2000]

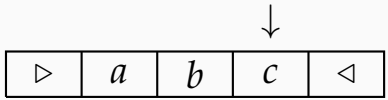
Fonctions polyrégulières = calculées par des transducteurs à k jetons

Ensemble fini d'états + *pile* de hauteur $\leqslant k$ de têtes bidirectionnelles («jetons»)

Les positions des têtes peuvent être *comparées*. Exemple :

$$\begin{array}{ccc} \Sigma^* & \to & (\Sigma \cup \underline{\Sigma})^* \\ abc & \mapsto & \underline{a}bca\underline{b}cab\underline{c} \end{array}$$





Sortie : $\underline{a}bca\underline{b}cab\underline{c}$

[1] https://fr.wikipedia.org/wiki/Automate_à_jetons

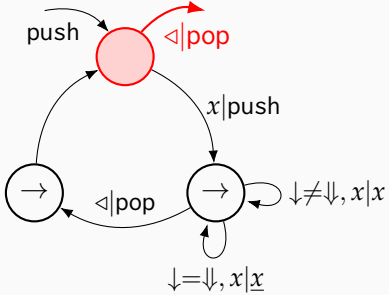
Transducteurs à jetons[1] (EN : pebble / PL : kamyki) [Milo, Suciu & Vianu 2000]

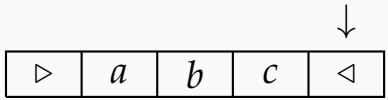
Fonctions polyrégulières = calculées par des transducteurs à k jetons

Ensemble fini d'états + *pile* de hauteur $\leqslant k$ de têtes bidirectionnelles («jetons»)

Les positions des têtes peuvent être *comparées*. Exemple :

$$\begin{array}{ccc} \Sigma^* & \to & (\Sigma \cup \underline{\Sigma})^* \\ abc & \mapsto & \underline{a}bca\underline{b}cab\underline{c} \end{array}$$





Sortie : $\underline{a}bca\underline{b}cab\underline{c}$

[1] https://fr.wikipedia.org/wiki/Automate_à_jetons

Fonctions polyrégulières sans comparaison (1)

Les *comparaisons* entre têtes de lecture (jetons) sont-elles vraiment nécessaires ?

Fonctions polyrégulières sans comparaison (1)

Les *comparaisons* entre têtes de lecture (jetons) sont-elles vraiment nécessaires ?

- **Résultat de cette thèse :** *oui,* si on les enlève le modèle *perd* en expressivité (contre-ex : $f(a^n) = a\#aa\#\ldots\#a^n$ et $g(a^{n_1}\#\ldots\#a^{n_k}) = a^{n_1\times n_1}\#\ldots\#a^{n_k\times n_k}$)

Fonctions polyrégulières sans comparaison (1)

Les *comparaisons* entre têtes de lecture (jetons) sont-elles vraiment nécessaires ?

- **Résultat de cette thèse :** *oui,* si on les enlève le modèle *perd* en expressivité (contre-ex : $f(a^n) = a\#aa\#\ldots\#a^n$ et $g(a^{n_1}\#\ldots\#a^{n_k}) = a^{n_1\times n_1}\#\ldots\#a^{n_k\times n_k}$)
- Mais définit-on une classe de fonctions naturelle/canonique ainsi ?

Fonctions polyrégulières sans comparaison (1)

Les *comparaisons* entre têtes de lecture (jetons) sont-elles vraiment nécessaires ?

- **Résultat de cette thèse :** *oui*, si on les enlève le modèle *perd* en expressivité (contre-ex : $f(a^n) = a\#aa\#\ldots\#a^n$ et $g(a^{n_1}\#\ldots\#a^{n_k}) = a^{n_1\times n_1}\#\ldots\#a^{n_k\times n_k}$)
- Mais définit-on une classe de fonctions naturelle/canonique ainsi ?
  Oui, elles apparaissent en étudiant le $\lambda\ell^{\oplus\&}$-calcul !

Théorème (cette thèse – répond à la question de tout à l'heure)

$f : \Gamma^* \to \Sigma^*$ *est* polyrégulière sans comparaison

$\iff$

$\exists A$ purement linéaire *et* $t : \mathsf{Str}_\Gamma[A] \to \mathsf{Str}_\Sigma$ *tels que* $\forall w \in \Gamma^*$, $\overline{f(w)} =_\beta t\ \overline{w}$

Fonctions polyrégulières sans comparaison (2)

On étudie aussi les fonctions polyrég. sans comp. « pour elles-mêmes »

$\longrightarrow$ article publié à ICALP 2021

Fonctions polyrégulières sans comparaison (2)

On étudie aussi les fonctions polyrég. sans comp. « pour elles-mêmes »

$\longrightarrow$ article publié à ICALP 2021

Exemple dans cette thèse : théorème de minimisation des jetons

$f : \Gamma^* \to \Sigma^*$ est polyrégulière sans comparaison et $|f(w)| = O(|w|^k)$

$\iff$

f est calculable par un transducteur à k jetons sans comparaison

Inspiration : min. de jetons pour les polyrég. générales [Lhote 2020]

Fonctions polyrégulières sans comparaison (2)

On étudie aussi les fonctions polyrég. sans comp. « pour elles-mêmes »

$\longrightarrow$ article publié à ICALP 2021

Exemple dans cette thèse : théorème de minimisation des jetons

$f : \Gamma^* \to \Sigma^*$ est polyrégulière sans comparaison et $|f(w)| = O(|w|^k)$

$\iff$

f est calculable par un transducteur à k jetons sans comparaison

Inspiration : min. de jetons pour les polyrég. générales [Lhote 2020]

... on adapte les techniques de l'article alors que son résultat est *faux* :

$w_1\# \dots \#w_n \mapsto (w_1)^n \dots (w_n)^n$ requiert 3 jetons ! (et n'est donc pas sans comp.)

(travail en cours avec Mikołaj Bojańczyk, Gaëtan Douéneau-Tabot, Sandra Kiefer et Cécilia Pradic ;
on a par contre un théorème de minimisation de dimension MSO)

Évaluation sémantique et théorie catégorique des automates

Comment passer des λ-termes typés aux automates ?

Évaluation sémantique et théorie catégorique des automates

Comment passer des λ-termes typés aux automates ?
Par la *sémantique dénotationnelle* : traduction λ-termes $\rightsquigarrow$ structures algébriques

- sémantique naïve du λ-calcul simplement typé dans les *ensembles finis*
  $\longrightarrow$ automates finis (théorème de Hillebrand & Kanellakis)

Évaluation sémantique et théorie catégorique des automates

Comment passer des λ-termes typés aux automates ?
Par la *sémantique dénotationnelle* : traduction λ-termes $\rightsquigarrow$ structures algébriques

- sémantique naïve du λ-calcul simplement typé dans les *ensembles finis*
  $\longrightarrow$ automates finis (théorème de Hillebrand & Kanellakis)

Approche pour nos résultats de « transducteurs implicites »

- Trouver une *catégorie monoïdale close* $\mathcal{C}$
  (donne une sémantique du fragment purement linéaire du $\lambda\ell^{\oplus\&}$-calcul)

Évaluation sémantique et théorie catégorique des automates

Comment passer des λ-termes typés aux automates ?
Par la *sémantique dénotationnelle* : traduction λ-termes $\rightsquigarrow$ structures algébriques

- sémantique naïve du λ-calcul simplement typé dans les *ensembles finis*
  $\longrightarrow$ automates finis (théorème de Hillebrand & Kanellakis)

Approche pour nos résultats de « transducteurs implicites »

- Trouver une *catégorie monoïdale close* $\mathcal{C}$
  (donne une sémantique du fragment purement linéaire du $\lambda\ell^{\oplus\&}$-calcul)
- telle que les $\mathcal{C}$*-automates* calculent les fonctions régulières
  (notion d'automate sur une catégorie ici : [Colcombet & Petrişan 2017])

Transducteurs à registres sans copie $\simeq$ $\mathcal{SR}_{\oplus}$-automates

- $\mathcal{SR}$ = catégorie des transitions sans copie entre ensembles finis de registres
- $(-)_{\oplus}$ = complétion libre par coproduits finis $\simeq$ ajoute des états finis

Transducteurs à registres sans copie $\simeq \mathcal{SR}_{\oplus}$-automates

- $\mathcal{SR}$ = catégorie des transitions sans copie entre ensembles finis de registres
- $(-)_{\oplus}$ = complétion libre par coproduits finis $\simeq$ ajoute des états finis

- Souci : $\mathcal{SR}_{\oplus}$ n'est que « partiellement » monoïdale close…

Transducteurs à registres sans copie $\simeq \mathcal{SR}_{\oplus}$-automates

- $\mathcal{SR}$ = catégorie des transitions sans copie entre ensembles finis de registres
- $(-)_{\oplus}$ = complétion libre par coproduits finis $\simeq$ ajoute des états finis

- Souci : $\mathcal{SR}_{\oplus}$ n'est que « partiellement » monoïdale close...
- Par contre, $(\mathcal{SR}_{\&})_{\oplus}$ est monoïdale close ($(-)_{\&}$ = complétion produit)

Catégories monoïdales closes et transducteurs à registres

Transducteurs à registres sans copie $\simeq \underline{\mathcal{SR}_{\oplus}}$-automates

- $\mathcal{SR}$ = catégorie des transitions sans copie entre ensembles finis de registres
- $(-)_{\oplus}$ = complétion libre par coproduits finis $\simeq$ ajoute des états finis

- Souci : $\mathcal{SR}_{\oplus}$ n'est que « partiellement » monoïdale close…
- Par contre, $(\mathcal{SR}_{\&})_{\oplus}$ est monoïdale close ($(-)_{\&}$ = complétion produit)

Complétion $((-)_{\&})_{\oplus} \simeq$ ajout d'états avec <u>non-déterminisme</u>

- les trans. à registres sans copie sont déterminisables [Alur & Deshmukh 2011]

Catégories monoïdales closes et transducteurs à registres

Transducteurs à registres sans copie $\simeq$ $\mathcal{SR}_\oplus$-automates

- $\mathcal{SR}$ = catégorie des transitions sans copie entre ensembles finis de registres
- $(-)_\oplus$ = complétion libre par coproduits finis $\simeq$ ajoute des états finis

- Souci : $\mathcal{SR}_\oplus$ n'est que « partiellement » monoïdale close...
- Par contre, $(\mathcal{SR}_\&)_\oplus$ est monoïdale close ($(-)_\&$ = complétion produit)

Complétion $((-)_\&)_\oplus \simeq$ ajout d'états avec non-déterminisme

- les trans. à registres sans copie sont déterminisables [Alur & Deshmukh 2011]
- **cette thèse :** déterminisation catégorique utilisant l'existence de certains « espaces de fonctions » $A \multimap B$ = clôture monoïdale « partielle » !

Structure monoïdale close de la catégorie $(\mathcal{SR}_{\&})_{\oplus}$

Pour $A, B \in \mathsf{Obj}(\mathcal{SR})$: décomposition des morphismes (technique classique)

$$\begin{cases} X := abXcY \\ Y := ba \end{cases} \quad \rightsquigarrow \quad \text{forme} \begin{cases} X := Z_1 X Z_2 Y \\ Y := Z_3 \end{cases} \quad + \quad \text{étiquettes } Z_1 = ab, \ldots$$

sans copie $\implies$ nombre *fini* de formes possibles $\implies$ $A \multimap B \in \mathsf{Obj}(\mathcal{SR}_{\oplus})$

Structure monoïdale close de la catégorie $(\mathcal{SR}_{\&})_{\oplus}$

Pour $A, B \in \mathsf{Obj}(\mathcal{SR})$: décomposition des morphismes (technique classique)

$$\begin{cases} X := abXcY \\ Y := ba \end{cases} \quad \rightsquigarrow \quad \text{forme} \quad \begin{cases} X := Z_1 X Z_2 Y \\ Y := Z_3 \end{cases} \quad + \quad \text{étiquettes } Z_1 = ab, \ldots$$

$$\textit{sans copie} \implies \text{nombre } \textit{fini} \text{ de formes possibles} \implies A \multimap B \in \mathsf{Obj}(\mathcal{SR}_{\oplus})$$

Théorème (cette thèse – autre exemple d'application de cette technique)

Les fonctions polyrégulières sans comparaison sont stables par composition.

Structure monoïdale close de la catégorie $(\mathcal{SR}_{\&})_{\oplus}$

Dans $(\mathcal{C}_{\&})_{\oplus}$, calcul $\bigoplus_u \&_x A_{u,x} \multimap \bigoplus_v \&_y B_{v,y} \;=\; \&_u \bigoplus_v \&_y \bigoplus_x A_{u,x} \multimap B_{v,y}$

Théorème (cette thèse – construction « à la Dialectica » [Gödel, de Paiva, Hofstra])

Soit $\mathcal{C}$ une catégorie monoïdale symétrique. Si un espace de fonctions $A \multimap B$ existe dans $\mathcal{C}_{\oplus}$ pour tous $A, B \in \mathsf{Obj}(\mathcal{C})$, alors $(\mathcal{C}_{\&})_{\oplus}$ est monoïdale close.

Pour $A, B \in \mathsf{Obj}(\mathcal{SR})$: décomposition des morphismes (technique classique)

$$\begin{cases} X := abXcY \\ Y := ba \end{cases} \quad \rightsquigarrow \quad \text{forme} \quad \begin{cases} X := Z_1 X Z_2 Y \\ Y := Z_3 \end{cases} \quad + \quad \text{étiquettes } Z_1 = ab, \ldots$$

sans copie $\implies$ nombre *fini* de formes possibles $\implies$ $A \multimap B \in \mathsf{Obj}(\mathcal{SR}_{\oplus})$

Théorème (cette thèse – autre exemple d'application de cette technique)

Les fonctions polyrégulières sans comparaison sont stables par composition.

Conclusion

- caractérisations de classes de transductions dans un λ-calcul linéaire
- lien entre sémantique dénotationnelle et théorie catégorique des automates
- étude des fonctions polyrégulières sans comparaison

Conclusion

- caractérisations de classes de transductions dans un λ-calcul linéaire
  - non mentionné : lien entre *apériodicité* et typage non commutatif (ICALP 2020) ; fonctions régulières sur les *arbres* et « single use restriction »
- lien entre sémantique dénotationnelle et théorie catégorique des automates

- étude des fonctions polyrégulières sans comparaison

Conclusion

- caractérisations de classes de transductions dans un λ-calcul linéaire
  - non mentionné : lien entre *apériodicité* et typage non commutatif (ICALP 2020) ; fonctions régulières sur les *arbres* et « single use restriction »
- lien entre sémantique dénotationnelle et théorie catégorique des automates
  - non mentionné : catégories monoïdales closes & stabilité par précomposition ; propriété universelle de la catégorie de registres $\mathcal{SR}$
- étude des fonctions polyrégulières sans comparaison

Conclusion

- caractérisations de classes de transductions dans un λ-calcul linéaire
  - non mentionné : lien entre *apériodicité* et typage non commutatif (ICALP 2020) ; fonctions régulières sur les *arbres* et « single use restriction »
- lien entre sémantique dénotationnelle et théorie catégorique des automates
  - non mentionné : catégories monoïdales closes & stabilité par précomposition ; propriété universelle de la catégorie de registres $\mathcal{SR}$
- étude des fonctions polyrégulières sans comparaison

Plein de perspectives : la brèche est ouverte, reste à s'y engouffrer !

- vieux problème ouvert : $\mathsf{Str}_\Gamma[A] \to \mathsf{Str}_\Sigma$ en λ-calcul simplement typé
- géométrie de l'interaction (planaire) et trans. bidirectionnels [Hines 2003]
- ensembles nominaux, λ-calcul parcimonieux, polymorphisme, ...

Merci pour votre attention !

- caractérisations de classes de transductions dans un λ-calcul linéaire
  - non mentionné : lien entre *apériodicité* et typage non commutatif (ICALP 2020) ; fonctions régulières sur les *arbres* et « single use restriction »
- lien entre sémantique dénotationnelle et théorie catégorique des automates
  - non mentionné : catégories monoïdales closes & stabilité par précomposition ; propriété universelle de la catégorie de registres $\mathcal{SR}$
- étude des fonctions polyrégulières sans comparaison

Plein de perspectives : la brèche est ouverte, reste à s'y engouffrer !

- vieux problème ouvert : $\mathsf{Str}_\Gamma[A] \to \mathsf{Str}_\Sigma$ en λ-calcul simplement typé
- géométrie de l'interaction (planaire) et trans. bidirectionnels [Hines 2003]
- ensembles nominaux, λ-calcul parcimonieux, polymorphisme, ...