Automatic Synthesis of Systems with Data

Synthèse automatique de systèmes avec données

Léo Exibard

Monday, September 20th, 2021

Introduction's Introduction

- The introduction will be in French

Technical slides in English	`https://cutt.ly/2EszvV5`	
Manuscript Introduction p. 24	`https://cutt.ly/tEszWtP`	
These slides	`https://cutt.ly/0EszRvM`	

Programmes

Programme : séquence d'instructions

Exemple : le tri par insertion

Entrée : Une liste d'entiers L

Résultat : La même liste, triée, T

$T \leftarrow$ une liste vide

tant que L *n'est pas vide* **faire**

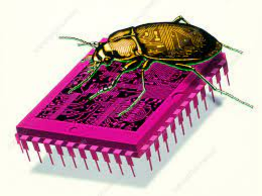
  retirer le premier élément x de L

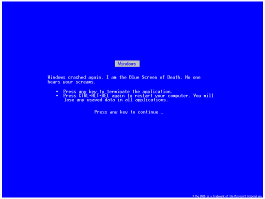
  parcourir la liste T (indice courant : i)

  dès que $x > T_i$, insérer x

fin

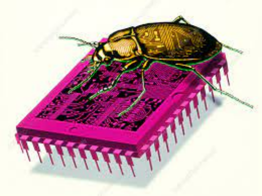
Méthodes formelles

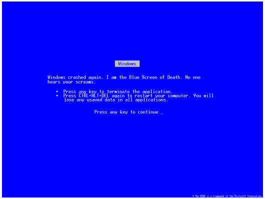




- ➔ Systèmes informatiques critiques
- ➔ Nécessité d'en garantir le bon fonctionnement

Méthodes formelles





- ➔ Systèmes informatiques critiques
- ➔ Nécessité d'en garantir le bon fonctionnement avec une certitude *mathématique*

Spécification

- ➔ Formalisation d'un « bon comportement »

Algorithme = *comment*

Spécification = *quoi*

Spécification

➔ Formalisation d'un « bon comportement »

Algorithme = *comment*

Spécification = *quoi*

Le tri

- Entrée : une liste d'entiers
- Sortie : la même liste, *triée*

Spécification

→ Formalisation d'un « bon comportement »

Algorithme = *comment*

Spécification = *quoi*

Le tri

- Entrée : $L \in \mathbb{N}^n$
- Sortie : $T \in \mathbb{N}^n$ telle que
  $\exists \sigma : \{1, \ldots, n\} \to \{1, \ldots, n\}$ bijective,
  $\sigma(L) = T$ et
  T est triée

Spécification

➔ Formalisation d'un « bon comportement »

Algorithme = *comment*

Spécification = *quoi*

Le tri

- Entrée : $L \in \mathbb{N}^n$
- Sortie : $T \in \mathbb{N}^n$ telle que

$$\exists \sigma : \{1, \ldots, n\} \to \{1, \ldots, n\},$$
$$\forall 1 \leq j \leq n, \exists ! 1 \leq i \leq n, \sigma(i) = j \wedge$$
$$\sigma(L) = T \wedge$$
$$\forall 1 \leq i, j \leq n, O_i \leq O_j$$

Spécification

➔ Formalisation d'un « bon comportement »

Algorithme = *comment*

Spécification = *quoi*

Le tri

- Entrée : $L \in \mathbb{N}^n$
- Sortie : $T \in \mathbb{N}^n$ telle que

$$
\begin{aligned}
&\exists \sigma : \{1, \ldots, n\} \to \{1, \ldots, n\},\\
&\forall 1 \leq j \leq n, \exists ! 1 \leq i \leq n, \sigma(i) = j \wedge\\
&\sigma(L) = T \wedge\\
&\forall 1 \leq i, j \leq n, O_i \leq O_j
\end{aligned}
$$

➔ L'algorithme de tri par insertion *satisfait* cette spécification

Vérification et synthèse

Entrées In — Sorties Out

Programme (traite les entrées, produit des sorties) $P : \mathsf{In} \to \mathsf{Out}$

Environnement (fournit les entrées) $E : \to \mathsf{In}$

Spécification (définit les entrées/sorties acceptables) $S \subseteq \mathsf{In} \times \mathsf{Out}$

$$\text{Programme } P \parallel \text{Environnement } E \models \text{Spécification } S$$

Vérification

Étant donnés P et S, vérifier que pour tout E, P *satisfait* S

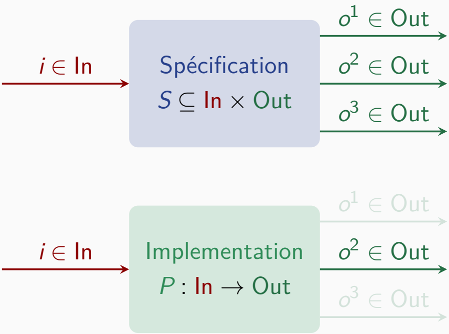
Synthèse

Étant donnée S, trouver P tel que pour tout E, P *satisfait* S

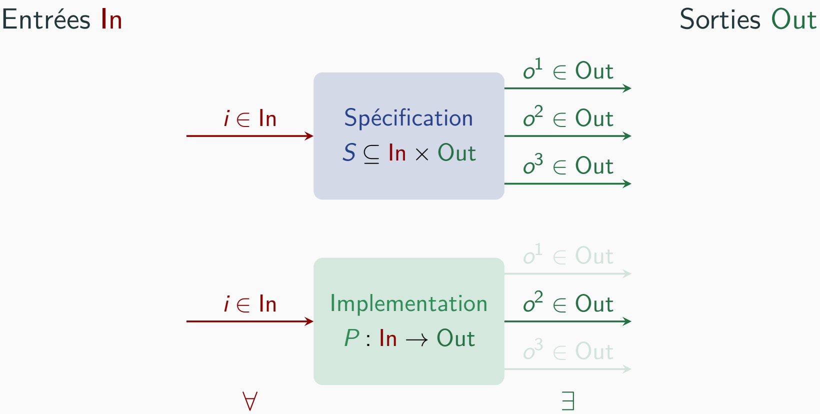
Synthèse

Entrées In

Sorties Out



Synthèse



Entrées In
Sorties Out
$i \in \mathrm{In}$
Spécification
$S \subseteq \mathrm{In} \times \mathrm{Out}$
$o^1 \in \mathrm{Out}$
$o^2 \in \mathrm{Out}$
$o^3 \in \mathrm{Out}$
$i \in \mathrm{In}$
Implementation
$P : \mathrm{In} \to \mathrm{Out}$
$o^1 \in \mathrm{Out}$
$o^2 \in \mathrm{Out}$
$o^3 \in \mathrm{Out}$
$\forall$
$\exists$

Le problème de la synthèse

Entrées In | Sorties Out

$\mathcal{S}$ classe de spécifications $S \subseteq \mathsf{In} \times \mathsf{Out}$

$\mathcal{I}$ classe d'implementations $P : \mathsf{In} \to \mathsf{Out}$

P satisfait S, noté $P \models S$, si pour tous $i \in \mathsf{In}$, $(i, P(i)) \in S$

Problème de la synthèse pour $\mathcal{S}$ et $\mathcal{I}$

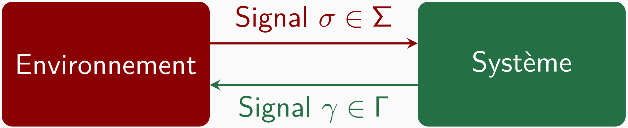
Entrée : $S \in \mathcal{S}$

Sortie :
- $P \in \mathcal{I}$ t. q. $P \models S$ si P existe
- **Non** sinon

La synthèse réactive

$\text{In} = \Sigma^\omega$ $\text{Out} = \Gamma^\omega$

Les systèmes réactifs



Interaction $\rightsquigarrow$ $\sigma_1\gamma_1\sigma_2\gamma_2\sigma_3\gamma_3\ldots$

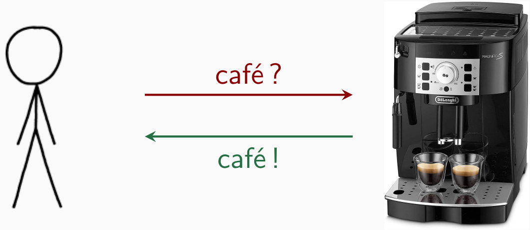
Spécification $S \subseteq (\Sigma \cdot \Gamma)^\omega$

Implémentation = machine à états finis = système réactif

La synthèse réactive

$\mathsf{In} = \Sigma^\omega$ $\mathsf{Out} = \Gamma^\omega$

Les systèmes réactifs



Interaction $\rightsquigarrow$ $\sigma_1\gamma_1\sigma_2\gamma_2\sigma_3\gamma_3\dots$

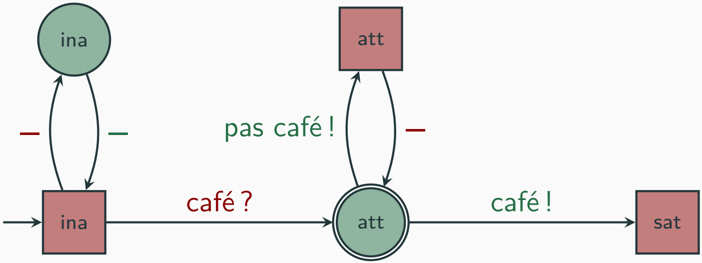
Spécification $S \subseteq (\Sigma \cdot \Gamma)^\omega$

Implémentation = machine à états finis = système réactif

Exemple

- Machine à café
- Chaque fois qu'un utilisateur commande un café, il doit finir par être satisfait
  ➔ $G(\text{req} \Rightarrow F(\text{grt}))$

ω-Automates

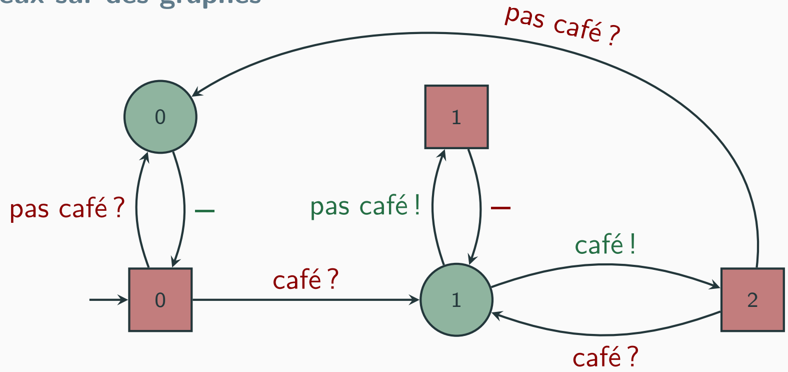


Un automate universel de co-Büchi vérifiant que chaque utilisateur finit par être satisfait.

Comment synthétiser un système réactif ?

- Construire un automate reconnaissant les exécutions correctes
- Construire un jeu à deux joueurs à partir de l'automate
- Stratégie gagnante $\Leftrightarrow$ implémentation

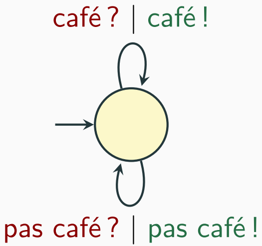
Jeux sur des graphes



Un jeu de parité correspondant à $G(\text{req} \Rightarrow F(\text{grt}))$

Les stratégies gagnantes ont une mémoire finie

Transducteurs séquentiels



- Automates avec des sorties
- Produit de manière déterministe une lettre en lisant une lettre
- Tous les états sont acceptants

Limites

Observations

- ➔ L'entrée et la sortie sont des ensembles finis
- ➔ Les ensembles de grande taille sont difficiles à gérer

Retour à l'exemple

- Ensemble $C = \{1, \ldots, n\}$ d'utilisateurs
- $\Sigma = \{\text{café?}_1, \ldots, \text{café?}_n, \text{pas café?}\}$ et
  $\Gamma = \{\text{café!}_1, \ldots, \text{café!}_n, \text{pas café!}\}$
- Désormais, chaque utilisateur a une requête spécifique
- Chaque commande de l'utilisateur i finit par être satisfaite :

$$\bigwedge_{1 \leq i \leq n} G\Big(\text{café?}_i \rightarrow F\big(\text{café!}_i\big)\Big)$$

Limites

Observations

- ➔ L'entrée et la sortie sont des ensembles finis
- ➔ Les ensembles de grande taille sont difficiles à gérer

Retour à l'exemple

- Ensemble $C = \{1, \ldots, n\}$ d'utilisateurs
- $\Sigma = \{\text{café?}_1, \ldots, \text{café?}_n, \text{pas café?}\}$ et
  $\Gamma = \{\text{café!}_1, \ldots, \text{café!}_n, \text{pas café!}\}$
- Désormais, chaque utilisateur a une requête spécifique
- Chaque commande de l'utilisateur i finit par être satisfaite :

$$\bigwedge_{1 \leq i \leq n} G\Big(\text{café?}_i \to F(\text{café!}_i)\Big)$$

➔ Nous considérons le cas où C est *infini* et doté d'une *structure*.

Quick recap

- Synthesis: generate a system from a specification
- Specification = MSO formula $\equiv$ ω-automaton
- System = finite-state machine (transducer)
- Reactive synthesis is *decidable* for MSO specifications
- We aim at extending the result to *infinite alphabets*

Objectives of the thesis

Main goal

Lift existing synthesis techniques to infinite alphabets

- ➔ Models for specifications and implementations
- ➔ Decidability and complexity of synthesis procedures
- ➔ Theoretical study of transducers over infinite alphabets

How to Represent Executions? Data Words

- *Data domain* $\mathcal{D} = (\mathbb{D}, \mathfrak{R}, \mathfrak{C})$: infinite set of *data* with predicates and constants
  ➔ e.g. $(\mathbb{N}, =)$, $(\mathbb{Q}, <)$, $(\mathbb{N}, <, 0)$
- Σ finite alphabet of *labels*
- *Data words*: sequences of pairs $(a, d) \in \Sigma \times \mathbb{D}$

1	4	2	2	3	1	5	3	...
req	¬grt	req	grt	req	grt	¬req	grt	...

- $\Sigma = \{\mathsf{req}, \mathsf{grt}, \neg\mathsf{req}, \neg\mathsf{grt}\}$
- $\mathcal{D} = (\mathbb{N}, =)$

Extending Automata to Data Words

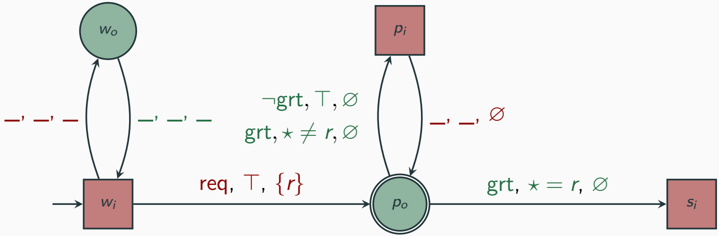
Register Automata (Kaminski and Francez 1994)

Finite automata with a finite set **R** of registers

- Store data
- Test register content

Transitions $q \xrightarrow{\sigma,\varphi,A} q'$

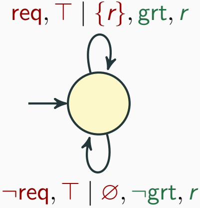
- $\sigma \in \Sigma$: label
- $\varphi \in \mathrm{QF}(R,\star)$: test
- $A \subseteq R$: assignment



An URA checking that every request is eventually granted.

Synchronous Sequential Register Transducers

- Transitions $q \xrightarrow{i,\varphi \mid A,o,r} q'$
  - i input letter, o output letter
  - φ test over $\star$
  - A registers assigned $\star$
  - r register whose content is output
- Sequentiality: tests are mutually exclusive



A register transducer immediately satisfying each user.

Outline

Part I: Reactive Synthesis

- Specifications: synchronous register automata
- Implementations: synchronous sequential register transducers
- Decidability border + compromise expressivity vs complexity

Part II: Computability

- Specifications: non-deterministic asynchronous register transducers
- Implementations: any algorithm
- Theory of asynchronous register transducers

Reactive Synthesis over Data Words

Synthesis of Register Transducers

$\mathcal{S}$: specification register automata

$\mathcal{I}$: synchronous sequential register transducers

Unbounded Synthesis Problem

Input: S a register automaton

Output:
- M a synchronous sequential register transducer such that $M \models S$ if it exists
- **No** otherwise

Theorem

The unbounded synthesis problem is undecidable for S given as a Universal Register Automaton with ≥ 3 registers, already over $(\mathbb{D}, =)$.

Register-Bounded Synthesis of Register Transducers

$\mathcal{S}$: specification register automata

$\mathcal{I}$: synchronous sequential register transducers with k registers

Register-Bounded Synthesis Problem

Input: S a register automaton, k a number of registers

Output:
- M a synchronous sequential register transducer with k registers (and arbitrarily many states) such that $M \models S$ if it exists
- **No** otherwise

Theorem

The register-bounded synthesis problem for S given as a Universal Register Automaton is in 2-ExpTime over $(\mathbb{D}, =)$ and $(\mathbb{Q}, <)$.

The Case of Universal Specifications

Transfer Theorem

S is realisable by a sequential register transducer with k registers iff $W_{S,k} = \{\alpha \mid \mathrm{Comp}(\alpha) \subseteq S\}$ is realisable by a (register-free) sequential transducer.

- → $W_{S,k}$ is ω-regular for S URA

The Case of Universal Specifications

Transfer Theorem

S is realisable by a sequential register transducer with k registers iff $W_{S,k} = \{\alpha \mid \mathsf{Comp}(\alpha) \subseteq S\}$ is realisable by a (register-free) sequential transducer.

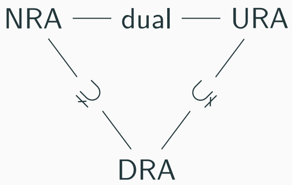
- ➔ $W_{S,k}$ is ω-regular for S URA
- ➔ Reduces to ω-regular synthesis

Theorem

The register-bounded synthesis problem for S given as a Universal Register Automaton is in 2-$\textsc{ExpTime}$ over $(\mathbb{D}, =)$ and $(\mathbb{Q}, <)$.

Results

	URA	DRA	NRA	test-free NRA
Register-bounded synthesis	2ExpTime	2ExpTime	Undecidable ($k \geq 1$)	2ExpTime
Unbounded Synthesis	Undecidable	ExpTime-c	Undecidable	Open



The Case of Deterministic Specifications: $(\mathbb{N}, <)$

Theorem

The unbounded synthesis problem for S given as a Deterministic Register Automaton over $(\mathbb{N}, <)$ is undecidable.

➔ Simulate counting using antagonism between the players

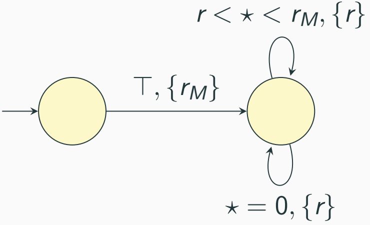
The Case of Deterministic Specifications: $(\mathbb{N}, <)$

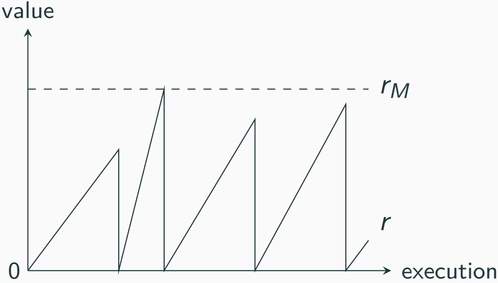
Theorem

The unbounded synthesis problem for S given as a Deterministic Register Automaton over $(\mathbb{N}, <)$ is undecidable.

➔ Simulate counting using antagonism between the players

Non-regular behaviours





➔ The set of feasible action words is not regular

The Case of Deterministic Specifications: $(\mathbb{N}, <)$

Theorem

The unbounded synthesis problem for S given as a one-sided Deterministic Register Automaton over $(\mathbb{N}, <)$ is $\textsc{ExpTime}$-c.

- Target finite-memory implementations
  $\rightsquigarrow$ regular approximation is enough.

Summary

	URA	DRA	NRA	test-free NRA
Register-bounded synthesis	2ExpTime	2ExpTime	Undecidable ($k \geq 1$)	2ExpTime
Unbounded Synthesis	Undecidable	ExpTime-c	Undecidable	Open

Decidability picture over $(\mathbb{D}, =)$ and $(\mathbb{Q}, <)$

- Generalises to *oligomorphic data domains*
- Over $(\mathbb{N}, <)$, only the unbounded synthesis for one-sided DRA is known to be decidable

Related publications

- E., Filiot and Reynier (CONCUR 2019 and LMCS 2021). "Synthesis of Data Word Transducers"
- E., Filiot and Khalimov (STACS 2021). "Church Synthesis on Register Automata over Linearly Ordered Data Domains"

Closely Related Works

Synthesis from register automata

- Khalimov, Maderbacher, and Bloem 2018
- Khalimov and Kupferman 2019
- Ehlers, Seshia, and Kress-Gazit 2014

Synthesis from automata with arithmetic

Faran and Kupferman 2020

Synthesis from Logic of Repeating Values

Figueira, Majumdar, and Praveen 2020

Synthesis over timed automata

D'Souza and Madhusudan 2002

Computability over Data Words

Asynchronicity

- It can be worth waiting for additional input before outputting something
- Growing body of research on generalised transducers

Asynchronous Register Transducers

$$p \xrightarrow{a, \varphi \mid A, \overbrace{(b_0, r_0) \dots (b_k, r_k)}^{\text{possibly empty}}} q$$

Asynchronicity

Theorem (Carayol and Löding 2015)

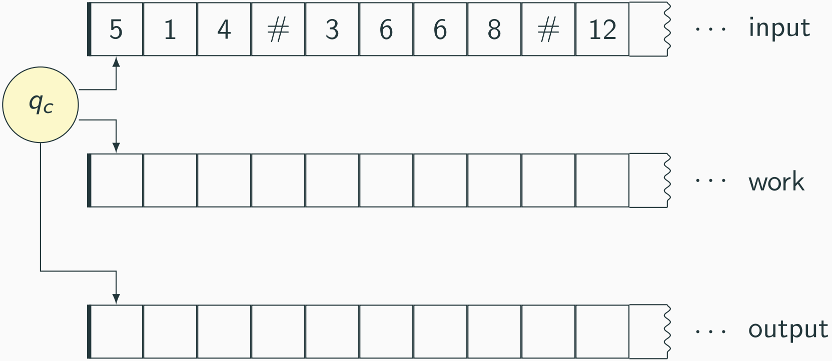
The synthesis problem from non-deterministic (register-free) asynchronous transducers to sequential ones is undecidable.

➔ Relax finite-memory requirement $\rightsquigarrow$ computable implementations.

Computability

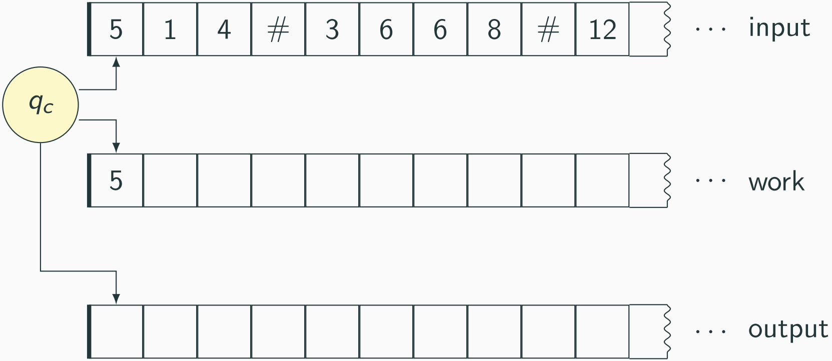
Example

$f_{\text{swap}} : w_1 d_1 \# w_2 d_2 \# \ldots \mapsto d_1 w_1 \# d_2 w_2 \ldots$



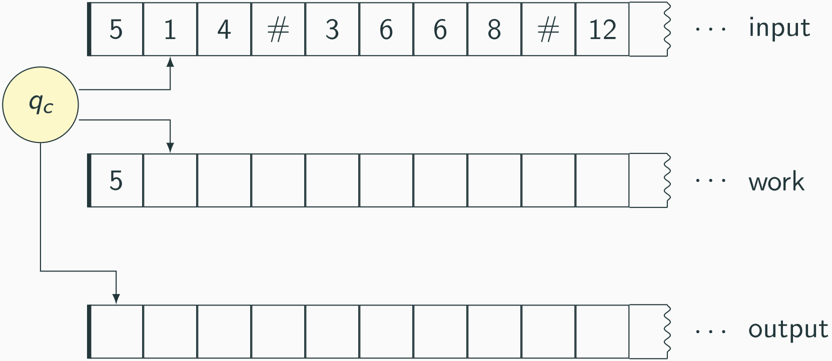
Example

$f_{\text{swap}} : w_1 d_1 \# w_2 d_2 \# \ldots \mapsto d_1 w_1 \# d_2 w_2 \ldots$



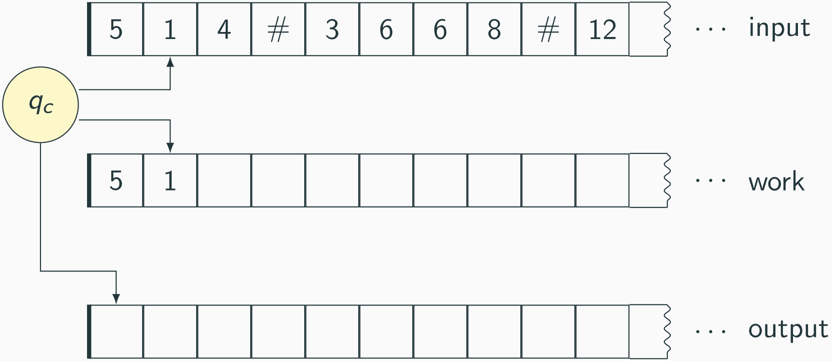
Example

$f_{\text{swap}} : w_1 d_1 \# w_2 d_2 \# \ldots \mapsto d_1 w_1 \# d_2 w_2 \ldots$



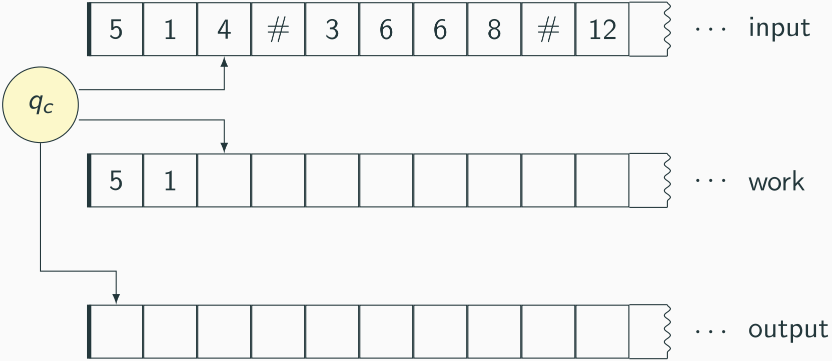
Example

$f_{\text{swap}} : w_1 d_1 \# w_2 d_2 \# \ldots \mapsto d_1 w_1 \# d_2 w_2 \ldots$



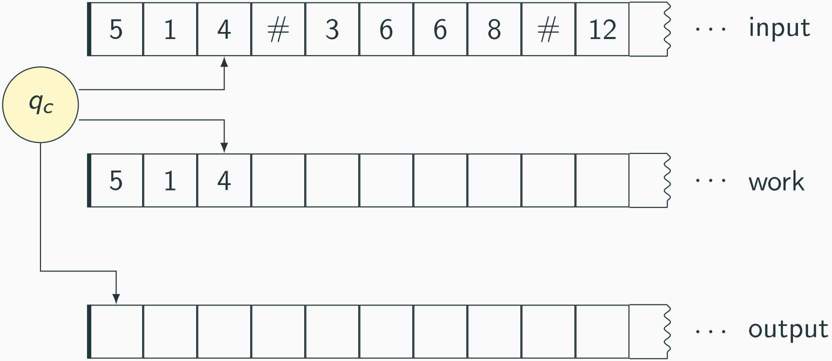
Example

$f_{\text{swap}} : w_1 d_1 \# w_2 d_2 \# \ldots \mapsto d_1 w_1 \# d_2 w_2 \ldots$



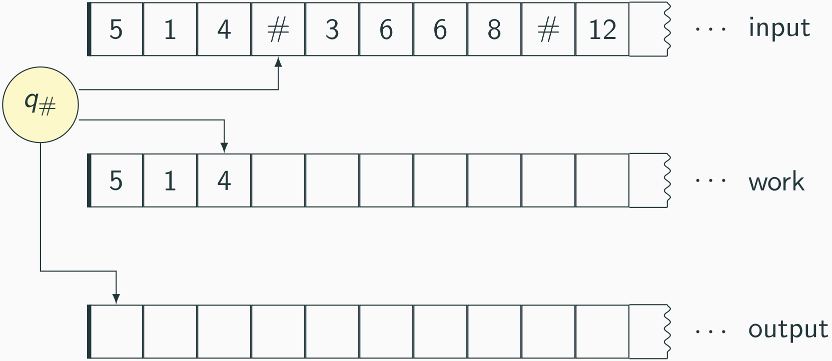
Example

$f_{\text{swap}} : w_1 d_1 \# w_2 d_2 \# \ldots \mapsto d_1 w_1 \# d_2 w_2 \ldots$



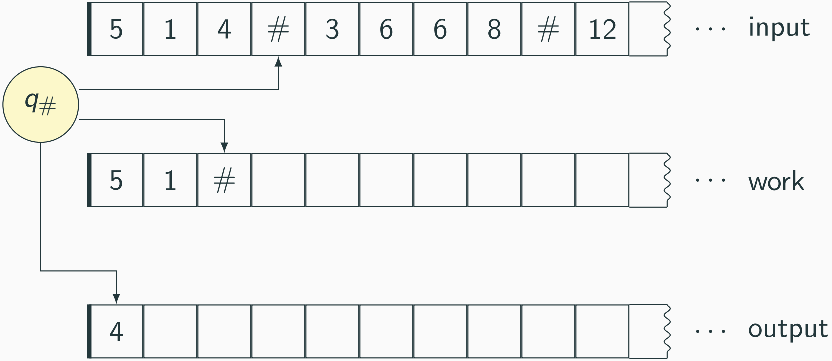
Example

$f_{\text{swap}} : w_1 d_1 \# w_2 d_2 \# \ldots \mapsto d_1 w_1 \# d_2 w_2 \ldots$



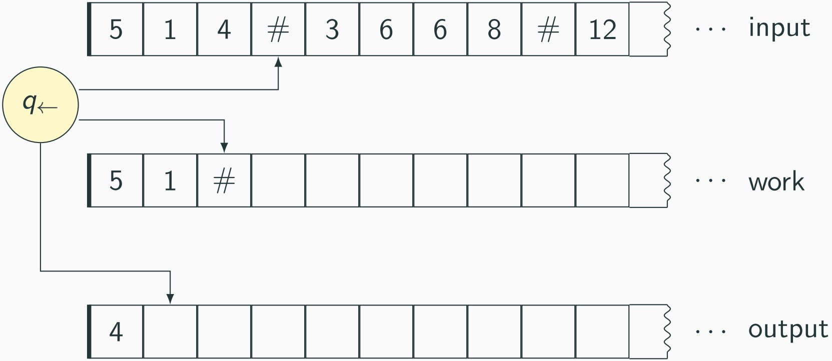
Example

$f_{\text{swap}} : w_1 d_1 \# w_2 d_2 \# \ldots \mapsto d_1 w_1 \# d_2 w_2 \ldots$



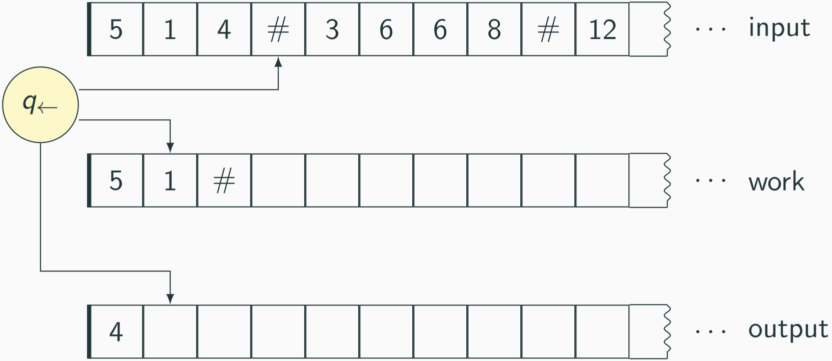
Example

$f_{\text{swap}} : w_1 d_1 \# w_2 d_2 \# \ldots \mapsto d_1 w_1 \# d_2 w_2 \ldots$



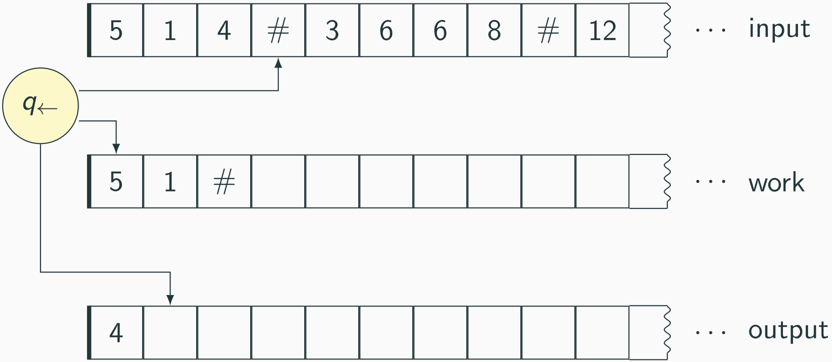
Example

$f_{\text{swap}} : w_1 d_1 \# w_2 d_2 \# \ldots \mapsto d_1 w_1 \# d_2 w_2 \ldots$



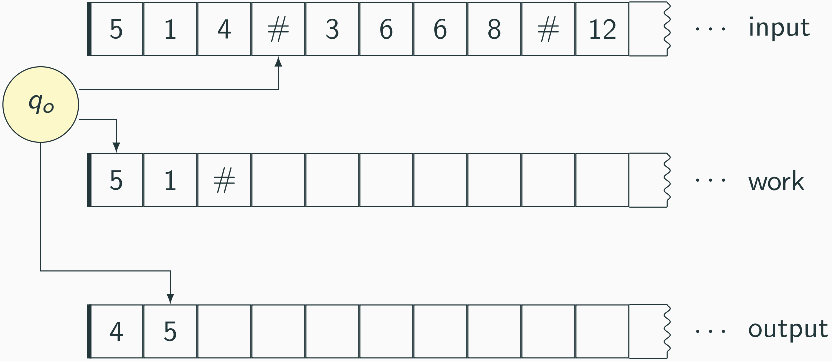
Example

$f_{\text{swap}} : w_1 d_1 \# w_2 d_2 \# \ldots \mapsto d_1 w_1 \# d_2 w_2 \ldots$



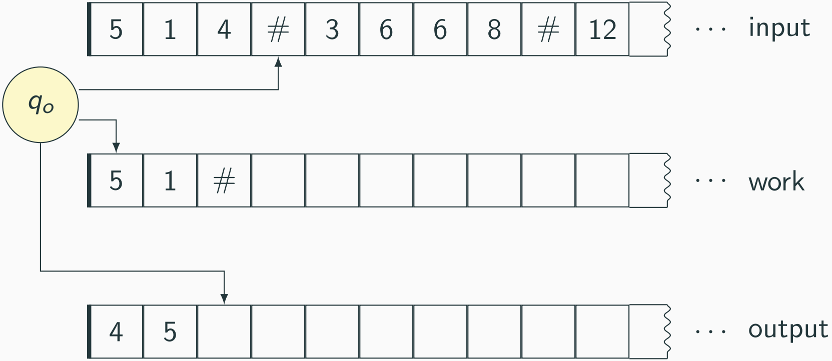
Example

$f_{\text{swap}} : w_1 d_1 \# w_2 d_2 \# \ldots \mapsto d_1 w_1 \# d_2 w_2 \ldots$



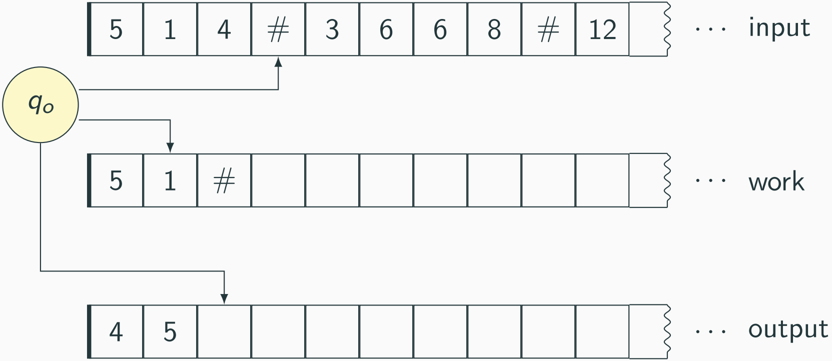
Example

$f_{\text{swap}} : w_1 d_1 \# w_2 d_2 \# \ldots \mapsto d_1 w_1 \# d_2 w_2 \ldots$



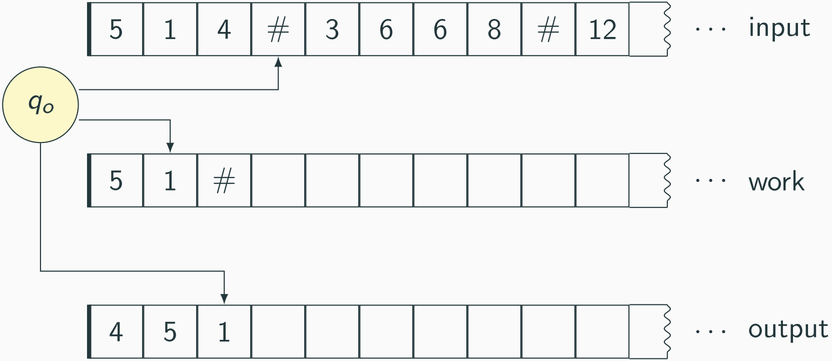
Example

$f_{\text{swap}} : w_1 d_1 \# w_2 d_2 \# \ldots \mapsto d_1 w_1 \# d_2 w_2 \ldots$



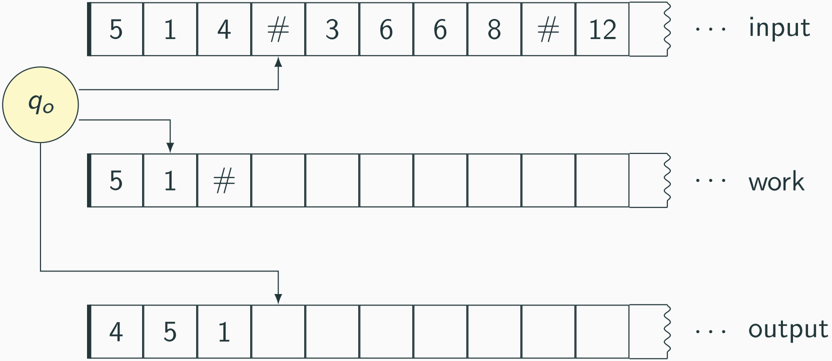
Example

$f_{\text{swap}} : w_1 d_1 \# w_2 d_2 \# \ldots \mapsto d_1 w_1 \# d_2 w_2 \ldots$



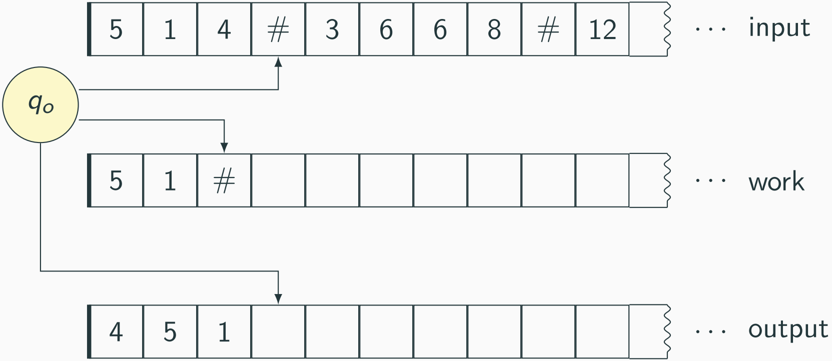
Example

$f_{\text{swap}} : w_1 d_1 \# w_2 d_2 \# \ldots \mapsto d_1 w_1 \# d_2 w_2 \ldots$



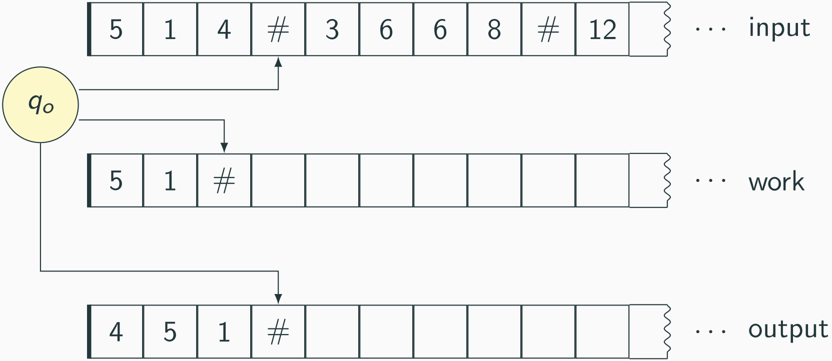
Example

$f_{\text{swap}} : w_1 d_1 \# w_2 d_2 \# \ldots \mapsto d_1 w_1 \# d_2 w_2 \ldots$



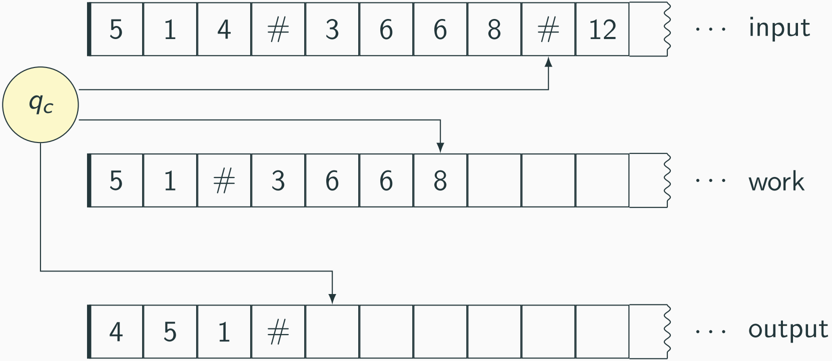
Example

$f_{\text{swap}} : w_1 d_1 \# w_2 d_2 \# \ldots \mapsto d_1 w_1 \# d_2 w_2 \ldots$



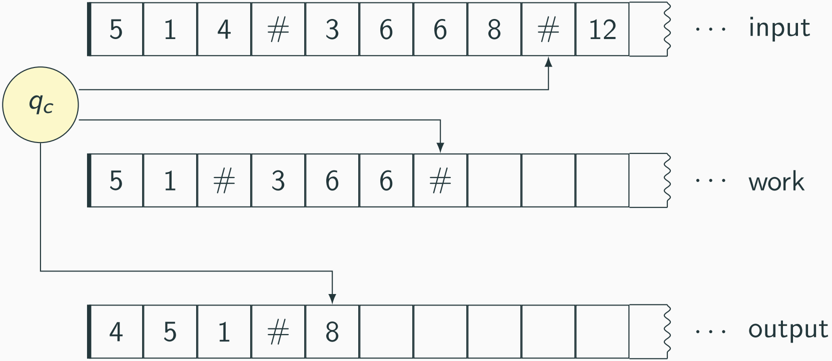
Example

$f_{\text{swap}} : w_1 d_1 \# w_2 d_2 \# \ldots \mapsto d_1 w_1 \# d_2 w_2 \ldots$



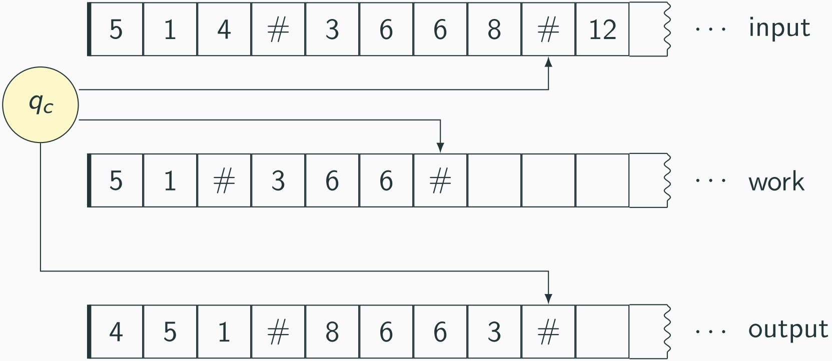
Example

$f_{\text{swap}} : w_1 d_1 \# w_2 d_2 \# \ldots \mapsto d_1 w_1 \# d_2 w_2 \ldots$



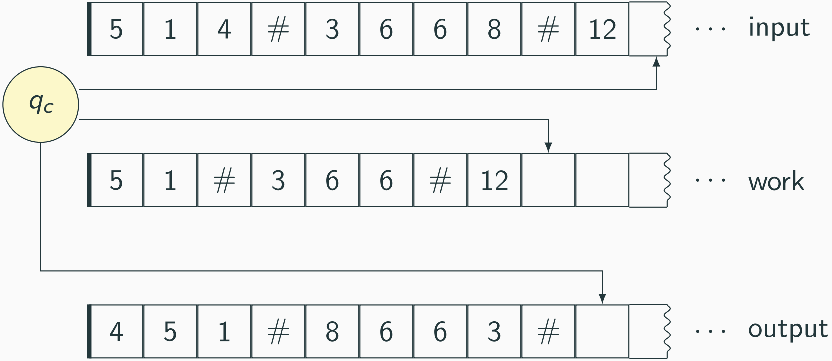
Example

$f_{\text{swap}} : w_1 d_1 \# w_2 d_2 \# \ldots \mapsto d_1 w_1 \# d_2 w_2 \ldots$



Example

$f_{\text{swap}} : w_1 d_1 \# w_2 d_2 \# \ldots \mapsto d_1 w_1 \# d_2 w_2 \ldots$



Computability

Three tape deterministic Turing machine

- Read-only one-way input tape
- Two-way working tape
- Write-only one-way output tape

M computes $f\colon \mathbb{D}^\omega \to \mathbb{D}^\omega$ if for all $x \in \mathrm{dom}(f)$,
M writes $f(x)$ in the limit

Theorem (Filiot and Winter 2021)

The synthesis problem of *computable functions* from non-deterministic asynchronous transducers over a *finite alphabet* is undecidable.

Computability

Three tape deterministic Turing machine

- Read-only one-way input tape
- Two-way working tape
- Write-only one-way output tape

M computes $f : \mathbb{D}^\omega \to \mathbb{D}^\omega$ if for all $x \in \mathrm{dom}(f)$,
M writes $f(x)$ in the limit

Theorem (Filiot and Winter 2021)

The synthesis problem of *computable functions* from non-deterministic asynchronous transducers over a *finite alphabet* is undecidable.

➔ Restrict to functional specifications, i.e. specifications that define functions.

Computability

Example

$f_{\mathrm{swap}} : w_1 d_1 \# w_2 d_2 \# \ldots \mapsto d_1 w_1 \# d_2 w_2 \ldots$

Computability

Example

$f_{\mathsf{swap}} : w_1 d_1 \# w_2 d_2 \# \ldots \mapsto d_1 w_1 \# d_2 w_2 \ldots$

- Definable by a non-deterministic register transducer
  (in the manuscript)
- Computable, not by a sequential transducer

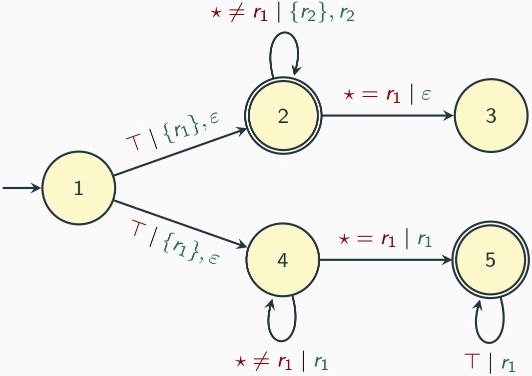
Computability

Example

$f_{\text{swap}} : w_1 d_1 \# w_2 d_2 \# \ldots \mapsto d_1 w_1 \# d_2 w_2 \ldots$

Co-example

$$f_{\text{again}} : dw \mapsto \begin{cases} w & \text{if } d \notin w \\ d^\omega & \text{otherwise} \end{cases}$$



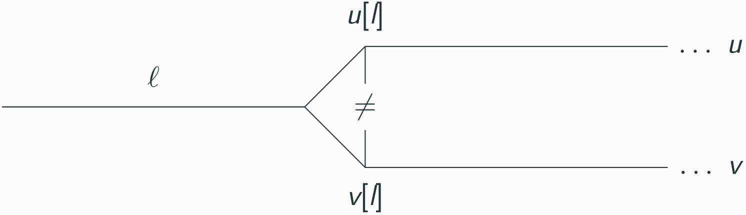
A register transducer defining f_{again}

Continuity

Cantor distance

For $u, v \in \mathbb{D}^\omega,\ d(u,v) = \begin{cases} 0 \text{ if } u = v \\ 2^{-|u \wedge v|}\text{otherwise} \end{cases}$

$u \wedge v$: longest common prefix ℓ of u and v



Continuous function

$f : \mathbb{D}^\omega \to \mathbb{D}^\omega$ is *continuous* if:

$$\lim_{n\infty} f(x_n) = f(\lim_{n\infty}(x_n))$$

Computability over finite alphabets

Theorem (Dave et al. 2019)

Let $f\colon \Sigma^\omega \to \Sigma^\omega$ be a function definable by a non-deterministic transducer over a *finite alphabet*. Then f is continuous iff it is computable.

Computability over finite alphabets

Theorem (Dave et al. 2019)

Let $f \colon \Sigma^\omega \to \Sigma^\omega$ be a function definable by a non-deterministic transducer over a *finite alphabet*. Then f is continuous iff it is computable.

Theorem (Dave et al. 2019)

Computability of functions defined by nondeterministic transducers is decidable in $\textsc{PTime}$.

Computability and Continuity

Computability

$f : \mathbb{D}^\omega \to \mathbb{D}^\omega$ computable: deterministic Turing machine that outputs $f(x)$ in the limit.

Continuity

$$\lim_{n\infty} f(x_n) = f(\lim_{n\infty}(x_n))$$

Computability $\Rightarrow$ Continuity

Deterministic machine: when *reading* head is at position k, the output only depends on the k first letters.

Computability and Continuity

Computability

$f : \mathbb{D}^\omega \to \mathbb{D}^\omega$ computable: deterministic Turing machine that outputs $f(x)$ in the limit.

Continuity

$$\lim_{n\infty} f(x_n) = f(\lim_{n\infty}(x_n))$$

Computability $\Rightarrow$ Continuity

Deterministic machine: when *reading* head is at position k, the output only depends on the k first letters.

- The other implication does not always hold.

Continuity and computability

Theorem

A function defined by a non-deterministic register transducer over oligomorphic domains or $(\mathbb{N}, <)$ *is computable iff it is continuous.*

Computability $\Rightarrow$ Continuity is proved as before.

Continuity $\Rightarrow$ Computability: requires to determine the next letter.

Next-letter problem

Input:	$u, v \in \mathbb{D}^*$
Output:	$d \in \mathbb{D}$ s.t. $\forall y \in \mathbb{D}^\omega$ s.t. $u \cdot y \in \mathrm{dom}(f)$,
	$v \cdot d \preceq f(u \cdot y)$ if it exists
	`No` otherwise

Continuity and computability

Theorem

A function defined by a non-deterministic register transducer over oligomorphic domains or $(\mathbb{N}, <)$ *is computable iff it is continuous.*

Computability $\Rightarrow$ Continuity is proved as before.

Continuity $\Rightarrow$ Computability: requires to determine the next letter.

Next-letter problem

Input:	$u, v \in \mathbb{D}^*$
Output:	$d \in \mathbb{D}$ s.t. $\forall y \in \mathbb{D}^\omega$ s.t. $u \cdot y \in \mathrm{dom}(f)$,
	$v \cdot d \preceq f(u \cdot y)$ if it exists
	`No` otherwise

Theorem

For functions defined by register transducers over oligomorphic domains or $(\mathbb{N}, <)$, deciding computability is $\mathrm{PS{\small PACE}}$-complete.

Summary

➔ Continuity $\equiv$ computability for functions defined by non-deterministic register transducers, over a large class of domains

➔ This is decidable.

Related publications

- E., Filiot and Reynier (FoSSaCS 2020). "On Computability of Data Word Functions Defined by Transducers"
- E., Filiot, *Lhote* and Reynier (submitted to LMCS). "Computability of Data-Word Transductions over Different Data Domains"

Perspectives

Reactive Synthesis

- Good-for-games register automata
- Register-bounded synthesis over $(\mathbb{N}, <, 0)$
- Synthesis from logical formalisms: $FO_2[<_p, \sim]$, $FO_2[<_p, <_d]$

Computability

- Generalise to other data domains and two-way models
- Lift the functionality requirement: automatic specifications

Going Further

- Explore other formalisms than register automata
- Minimisation and learning of non-deterministic transducers

Perspectives

Bibliography

Büchi, J. Richard and Lawrence H. Landweber (1969). "Solving Sequential Conditions by Finite-State Strategies". In: *Transactions of the American Mathematical Society* 138, pp. 295–311. ISSN: 00029947. DOI: 10.2307/1994916.

Carayol, Arnaud and Christof Löding (2015). " Uniformization in Automata Theory". In: *Logic, Methodology and Philosophy of Science - Proceedings of the 14th International Congress*.

Dave, Vrunda et al. (2019). "Deciding the Computability of Regular Functions over Infinite Words". In: *CoRR* abs/1906.04199. arXiv: 1906.04199.

D'Souza, Deepak and P. Madhusudan (2002). "Timed Control Synthesis for External Specifications". In: *STACS 2002, 19th Annual Symposium on Theoretical Aspects of Computer Science, Antibes - Juan les Pins, France, March 14-16, 2002, Proceedings*. Ed. by Helmut Alt and Afonso Ferreira. Vol. 2285. Lecture Notes in Computer Science. Springer, pp. 571–582. DOI: 10.1007/3-540-45841-7_47.

Ehlers, Rüdiger, Sanjit A. Seshia, and Hadas Kress-Gazit (2014). "Synthesis with Identifiers". In: *Verification, Model Checking, and Abstract Interpretation - 15th International Conference, VMCAI 2014, San Diego, CA, USA, January 19-21, 2014, Proceedings*. Ed. by Kenneth L. McMillan and Xavier Rival. Vol. 8318. Lecture Notes in Computer Science. Springer, pp. 415–433. DOI: 10.1007/978-3-642-54013-4_23.

Faran, Rachel and Orna Kupferman (2020). "On Synthesis of Specifications with Arithmetic". In: *SOFSEM 2020: Theory and Practice of Computer Science - 46th International Conference on Current Trends in Theory and Practice of Informatics, SOFSEM 2020, Limassol, Cyprus, January 20-24, 2020, Proceedings*. Ed. by Alexander Chatzigeorgiou et al. Vol. 12011. Lecture Notes in Computer Science. Springer, pp. 161–173. DOI: 10.1007/978-3-030-38919-2_14.

Figueira, Diego, Anirban Majumdar, and M. Praveen (2020). "Playing with Repetitions in Data Words Using Energy Games". In: *Log. Methods Comput. Sci.* 16.3.

Filiot, Emmanuel and Sarah Winter (2021). "Continuous Uniformization of Rational Relations and Synthesis of Computable Functions". In: *CoRR* abs/2103.05674. arXiv: 2103.05674.

Kaminski, Michael and Nissim Francez (1994). "Finite-memory automata". In: *Theoretical Computer Science* 134.2, pp. 329–363. ISSN: 0304-3975. DOI: https://doi.org/10.1016/0304-3975(94)90242-9.

Khalimov, Ayrat and Orna Kupferman (2019). “Register-Bounded Synthesis”. In: *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*. Ed. by Wan J. Fokkink and Rob van Glabbeek. Vol. 140. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 25:1–25:16. DOI: 10.4230/LIPIcs.CONCUR.2019.25.

Khalimov, Ayrat, Benedikt Maderbacher, and Roderick Bloem (2018). “Bounded Synthesis of Register Transducers”. In: *Automated Technology for Verification and Analysis, 16th International Symposium, ATVA 2018, Los Angeles, October 7-10, 2018. Proceedings*.

Pnueli, A. and R. Rosner (1989). “On the Synthesis of a Reactive Module”. In: *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL ’89. Austin, Texas, USA: ACM, pp. 179–190. ISBN: 0-89791-294-2. DOI: 10.1145/75277.75293.