Definability and synthesis of transductions

Nathan Lhote

Thèse en cotutelle présentée pour obtenir le grade de

Docteur de l'Université de Bordeaux et de l'Université Libre de Bruxelles

École doctorale mathématiques et informatique
spécialité informatique

Par Nathan Lhote

Définissabilité et Synthèse de Transductions

Definability and Synthesis of Transductions

Sous la direction d'Emmanuel Filiot et d'Olivier Gauwin

Soutenue le 12 octobre 2018

Membres du Jury:

Mikołaj Bojańczyk	Professeur, Uniwersytet Warszawski	Examinateur
Thomas Colcombet	Directeur de recherche CNRS, Université Paris-Diderot	Rapporteur
Emmanuel Filiot	Chercheur qualifié FNRS, Université Libre de Bruxelles	Directeur
Olivier Gauwin	Maître de conférences, Université de Bordeaux	Directeur
Christof Löding	Chercheur associé, RWTH Aachen	Rapporteur
Anca Muscholl	Professeure, Université de Bordeaux	Examinatrice
Jean-François Raskin	Professeur, Université Libre de Bruxelles	Examinateur
Helmut Seidl	Professeur, Technische Universität München	Président

Titre Définissabilité et Synthèse de Transductions

Résumé Dans la première partie de ce manuscrit nous étudions les *fonctions rationnelles*, c'est-à-dire définies par des transducteurs unidirectionnels. Notre objectif est d'étendre aux transductions les nombreuses correspondances logique-algèbre qui ont été établies concernant les langages, notamment le célèbre théorème de Schützenberger-McNaughton-Papert. Dans le cadre des fonctions rationnelles sur les mots finis, nous obtenons une caractérisation à la Myhill-Nerode en termes de congruences d'indice fini. Cette caractérisation nous permet d'obtenir un résultat de transfert, à partir d'équivalences logique-algèbre pour les langages vers des équivalences pour les transductions. En particulier nous montrons comment décider si une fonction rationnelle est définissable en logique du premier ordre. Sur les mots infinis, nous pouvons également décider la définissabilité en logique du premier ordre, mais avec des résultats moins généraux.

Dans la seconde partie nous introduisons une logique pour les transductions et nous résolvons le *problème de synthèse régulière*: étant donnée une formule de la logique, peut-on obtenir un transducteur bidirectionnel déterministe satisfaisant la formule ? Plus précisément nous fournissons un algorithme qui produit toujours une fonction régulière satisfaisant une spécification donnée en entrée. Nous exposons également un lien intéressant entre les transductions et les mots avec données. Par conséquent nous obtenons une logique expressive pour les mots avec données, pour laquelle le problème de satisfiabilité est décidable.

Mots-clefs Transductions, minimisation, congruence syntaxique, aperiodicité, logique du second ordre monadique, logique du premier ordre, origine, vérification, synthèse, data words

Title Definability and Synthesis of Transductions

Abstract In the first part of this manuscript we focus on the study of *rational functions*, functions defined by one-way transducers. Our goal is to extend to transductions the many logic-algebra correspondences that have been established for languages, such as the celebrated Schützenberger-McNaughton-Papert Theorem. In the case of rational functions over finite words, we obtain a Myhill-Nerode-like characterization in terms of congruences of finite index. This characterization allows us to obtain a transfer result from logic-algebra equivalences for languages to logic-algebra equivalences for transductions. In particular, we show that one can decide if a rational function can be defined in first-order logic. Over infinite words, we obtain weaker results but are still able to decide first-order definability.

In the second part we introduce a logic for transductions and solve the *regular synthesis problem*: given a formula in the logic, can we obtain a two-way deterministic transducer satisfying the formula? More precisely, we give an algorithm that always produces a regular function satisfying a given specification. We also exhibit an interesting link between transductions and words with ordered data. Thus we obtain as a side result an expressive logic for data words with decidable satisfiability.

Keywords Transductions, minimization, syntactic congruence, aperiodicity, monadic second-order logic, first-order logic, origin, verification, synthesis, data words

Laboratoires d'accueil Laboratoire Bordelais de Recherche en Informatique, 351 Cours de la Libération, 33405 Talence

Département d'informatique de l'Université Libre de Bruxelles, Bâtiment NO 8ème étage, Campus de la Plaine, ULB CP212, boulevard du Triomphe, 1050 Bruxelles

Remerciements

Ces années de thèse ont été une étape clef de ma vie pendant laquelle j'ai beaucoup appris et beaucoup changé. Une constante durant ces années est le soutient que j'ai reçu de la part des personnes qui m'entourent et qui ont contribué à rendre ces années heureuses. Je souhaite commencer par remercier mes encadrants de thèse, Manu, Olivier et Anca bien sûr. Je me sens extrêmement privilégié d'avoir eu trois encadrants d'une telle qualité et j'ai énormément appris à leur contact pendant ces trois virgule cinq années.

Merci aux membres de mon jury et en particulier à Thomas et Christof d'avoir accepté de relire mon manuscrit et merci pour vos remarques pertinentes.

Merci aux membres du LaBRI qui contribuent à une ambiance de travail très agréable. Merci à Mohammed pour des discussions endiablées sur le cours du mitigeur, à Louis-Marie pour ses calembours interstellaires, à Luis pour les parties de ping-pong acrobatique, à Filip pour son amour des cheese nan kebabs, à Nath pour ses leçons d'escalade. Merci aux nombreux membres du bureau 123 pour une bonne ambiance: Edon, Félix, Adithya, Varun, Jason, Paul, Karim, Kinda, Rohan, *etc.* Merci également aux membres de l'ULB, merci à Luc de m'avoir supporté dans son bureau, à Ismaël pour ses discussions matinales, à Guillermo pour la fondue, à Marie pour les sablés, à Nicolas pour ses tenues rayées, à Léo pour les soirées musicales. Je souhaite également remercier les gestionnaires du LaBRI et de l'ULB qui nous rendent la vie infiniment plus facile, en particulier Emmanuelle, Véronique, Pascaline et Maryka.

Merci aux matheux de cette annexe du LaBRI qu'est L'IMB. Merci à Thibaut 'Magic' Kritter pour les parties de basket, merci à Elsa, Nicolas, Jonathan et les autres pour les verres en terrasse et les nombreuses et inoubliables soirées `\lambda`.

Merci Elizabeth pour la dégustation de fruits de mer et merci Cannelle pour les verres lumineux. Merci aux potes de Rennes, Vincent meilleur perdant que gagnant, Sarah pour ces supers années coloc', Lucile pour le voyage au bout du monde, Robert pour les soirées Ludo & Robert , Ludo pour les soirées Ludo & Robert, Hélianthe pour avoir fait de moi la fashionista que je suis aujourd'hui. Un merci tout particulier à Amélie sans qui je n'aurais sans doute pas fait cette thèse.

Ensuite je souhaite remercier mes parents Dominique et Thierry que j'aime qui m'ont apporté un soutient inconditionnel pendant des études longues et pas toujours linéaires. Et puis pour le reste, merci aussi ! Merci à mes sœurs Rachel et Clara, mes deux personnes préférées ex æquo que j'aime et que j'admire. Merci à Loïc et Nicolas et merci aussi à leurs enfants (respectifs) Yaël, Marek, Côme et Talia, que j'adore voir grandir et apprendre à connaître. Merci à mes grand-parents, Jacqueline, Geneviève aka GM, Armand (et André que j'ai peu connu), pour l'amour et le soutient qu'ils m'ont apporté.

Enfin si j'ai beaucoup changé, beaucoup appris et si ces trois années ont été heureuses, c'est grâce à toi Cécilia. Merci.

Résumé en français

Des langages aux transductions

L'étude des langages formels est l'un des piliers de l'informatique théorique et a permis de développer de nombreux outils théoriques et pratiques dans différents domaines. Certaines classes de langages se démarquent des autres car elles bénéficient de plusieurs descriptions différentes, par exemple les langages récursivement énumérables peuvent être caractérisés en termes de machines de Turing ou en termes de grammaires de type 0 dans la hiérarchie de Chomsky. Un exemple qui nous intéresse particulièrement est celui de la classe des langages rationnels[1] qui se situe au plus bas niveau de la hiérarchie de Chomsky puisqu'elle est caractérisée par les grammaires régulières. Les langages rationnels sont également caractérisés par les expressions rationnelles, les formules de la logique monadique du second-ordre (MSO), les monoïdes finis (ou de manière équivalente les congruences d'indice fini), les automates finis (ainsi que toutes leurs variantes: déterministes, non-déterministes, alternants, unidirectionnels, bidirectionnels, *etc*), *etc.* Par exemple, le langage des mots de longueur paire sur l'alphabet $\{a\}$, est reconnu par l'automate donné en Fig. 1, et est défini par l'expression rationnelle $(aa)^*$.

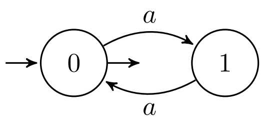


Figure 1: Automate déterministe reconnaissant le langage $(aa)^*$.

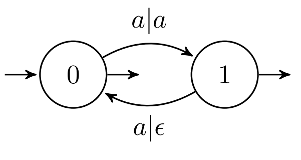
La théorie de la calculabilité peut aussi être abordée du point de vue des fonctions et là encore des classes particulières bénéficient de plusieurs descriptions différentes, par exemple, les fonctions calculables peuvent être caractérisées en termes de machines de Turing, de fonctions μ-récursives, ou d'expressions du lambda calcul. Dans le modèle de Turing, une machine (non-déterministe) avec une bande d'entrée et une bande de sortie définit une relation sur les mots appelée une *transduction.*

Une question importante à nos yeux est la suivante: quelle est la classe de transductions homologue aux langages rationnels ? Il n'y a pas de réponse évidente à cette question puisque la classe de transductions que l'on obtient dépend du modèle de calcul que l'on choisit de généraliser. Par exemple, les transductions *reconnaissables* par monoïdes finis forment une des classes de transductions les moins expressives et une telle transduction peut être décrite comme une union finie de produits de langages rationnels (voir [Ber79]).

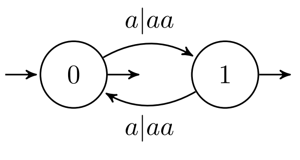
Un automate peut être promu en *transducteur*, c'est-à-dire un automate avec des sorties, et il réalise ainsi une transduction. En Fig. 2 nous donnons deux exemples de transducteurs.

[1]Souvent appelés langages réguliers.

Le premier copie la moitié des lettres de son entrée tandis que le second double chaque lettre. Tandis qu'un automate peut être vu comme une machine de Turing avec une bande de lecture seule, un transducteur est une machine de Turing avec une bande de lecture et une bande d'écriture. Les variantes d'automates finis reconnaissent toutes la même classe de langages, cependant ce n'est pas le cas pour les transductions. Premièrement, un transducteur déterministe doit réaliser une fonction puisqu'une certaine configuration de la bande d'entrée ne permet qu'un unique calcul. Un transducteur non-déterministe au contraire peut réaliser une relation non-fonctionnelle. Deuxièmement, les transducteurs unidirectionnels sont strictement moins expressifs que les transducteurs bidirectionnels, par exemple la fonction miroir, qui inverse l'ordre d'un mot, peut uniquement être réalisée par un transducteur bidirectionnel. Les transductions réalisées par des transducteurs unidirectionnels sont aussi définissables par des expressions rationnelles sur une produit de monoïdes libres (voir [Ber79]) et ont donc été nommées *transductions rationnelles.* Les fonctions réalisées par des transducteurs bidirectionnels ont également été caractérisées par différents modèles, notamment les transductions MSO à la Courcelle ainsi que les *streaming string transducers* (SST), des transducteurs unidirectionnels augmentés de registres. Ces équivalences ont été montrées respectivement dans [EH01, AC10], et les fonctions de cette classe sont souvent appelées *fonctions régulières.* Pour une vue d'ensemble de ces différents modèles voir [Fil15, FR16].



(a) $a^n \mapsto a^{\lceil \frac{n}{2} \rceil}$



(b) $a^n \mapsto a^{2n}$

Figure 2: Deux transducteurs séquentiels

Les problèmes

Les problèmes classiques de la théorie des langages formels peuvent être formulés en termes de transductions. Nous considérons des questions de définissabilité et de minimisation, et nous exposons quels résultats ont déjà été obtenus pour les langages rationnels et leurs extensions vers les transductions. De plus de nouveaux types de problèmes se posent, en particulier les questions de synthèse pour les transductions ont beaucoup été étudiées.

Problèmes de minimisation et formes canoniques Un problème de minimisation demande, étant donné un objet syntaxique M_2 (par exemple un automate) dans une classe $\mathcal{C}$, avec une interprétation sémantique $[\![M_2]\!]$, s'il existe un objet $M_1 \in \mathcal{C}$ tel que $[\![M_1]\!] = [\![M_2]\!]$ et M_1 est "plus petit" que M_2, pour une certaine définition de taille. Les problèmes de minimisation sont omniprésents en informatique pour des raisons évidentes d'efficacité. De plus, les objets minimaux sont souvent liés à des formes *canoniques* qui ont d'autres intérêts algorithmiques. Une forme canonique pour des objets d'une certaine classe est une procédure qui prend en entrée un objet $M \in \mathcal{C}$ et produit un objet M' qui ne dépend que de $[\![M]\!]$ et tel que $[\![M]\!] = [\![M']\!]$. En particulier, le fait d'avoir une procédure pour calculer une forme canonique permet de tester l'équivalence de deux objets. De plus une telle forme canonique permet souvent de tester d'autres propriétés sémantiques d'un objet.

En ce qui concerne les langages rationnels, la minimisation a été résolue pour différents modèles. Le *monoïde syntaxique* (de manière équivalente, la congruence syntaxique, voir [Ner63]) d'un langage rationnel est *le* plus petit monoïde reconnaissant le langage, et de plus ce monoïde est minimal au sens fort où il divise n'importe quel monoïde reconnaissant le langage. À partir de cela, on peut définir l'*automate déterministe minimal* d'un langage, qui est minimal en nombre d'états parmi les automates déterministes reconnaissant le même langage. Une fois encore, cette propriété de minimalité est en fait plus forte: l'automate minimal est un quotient de n'importe quel automate déterministe reconnaissant le même langage, et cette structure supplémentaire fournit des procédures de minimisation efficaces en PTime ([Moo56, Hop71]). Dans le cadre des transducteurs séquentiels (des transducteurs unidirectionnels avec un automate sous-jacent déterministe) un objet minimal similaire a été découvert par Choffrut avec la même propriété de minimalité forte que le transducteur séquentiel minimal est un quotient de n'importe quel transducteur séquentiel réalisant la même fonction. Cette caractérisation peut être trouvée dans [Cho03], où l'auteur fournit également plusieurs algorithmes de minimisation en PTime, similaires au cas des automates. L'existence de tels objets minimaux pour les automates et pour les transducteurs séquentiels a été traduite en termes de théorie des catégories par [CP17]. Par exemple le transducteur en Fig. 2(b) peut être minimisé en un transducteur avec un seul état.

Dans [BGMP16, BGMP17], les auteurs considèrent une approche différente de la minimisation: étant donné un transducteur bidirectionnel fonctionnel, ils montrent comment minimiser, uniformément sur toutes les entrées, le nombre de changements de direction de la tête de lecture. Une approche encore différente est étudiée dans [DRT16, DJRV17] où les auteurs considèrent des SST avec des restrictions sur les opérations de registres et parviennent à minimiser le nombre de registres en utilisant des propriétés de jumelage à la Choffrut.

Problèmes de définissabilité Le *problème de $\mathcal{C}_1$-definissabilité pour $\mathcal{C}_2$* demande, pour un objet syntaxique M_2 (par exemple un automate, une formule, *etc*) dans une classe $\mathcal{C}_2$ avec une interprétation sémantique $[\![M_2]\!]$, s'il existe un objet $M_1 \in \mathcal{C}_1$ tel que $[\![M_1]\!] = [\![M_2]\!]$. Premièrement, remarquons qu'un problème de minimisation peut être vu comme un cas particulier de problème de définissabilité en prenant pour $\mathcal{C}_1$ une classe de petits objets de $\mathcal{C}_2$. Des problèmes de définissabilité apparaissent dans de nombreux domaines de l'informatique lorsqu'on veut savoir si un objet peut être défini dans un certain modèle de calcul.

Un *fragment logique* de MSO est un sous-ensemble des formules MSO défini par une restriction syntaxique. Pour un tel fragment F, on a le problème de F-définissabilité pour les langages réguliers. Beaucoup de tels problèmes ont été résolus par la théorie des *variétiés* de monoïdes finis (parfois appelées pseudovariétés, voir [Str94]). En effet, les variétés de monoïdes sont closes par division ce qui implique qu'un langage est reconnaissable par un monoïde dans une variété $\mathbf{V}$ si et seulement si son monoïde syntaxique est dans $\mathbf{V}$. De plus la théorie d'Eilenberg des variétés, initiée par [Eil76], fournit une description équationnelle des variétés de monoïde ce qui donne dans de nombreux cas une procédure pour décider si un langage est reconnaissable par un monoïde dans une variété donnée.

Le premier exemple d'une telle équivalence est le théorème de Schützenberger/McNaughton-Papert ([Sch65, MP71]) qui dit qu'un langage est définissable en logique du premier ordre si et seulement si il est reconnaissable par un monoïde apériodique. Ceci fournit une procédure pour décider la FO[$\leq$]-définissabilité des langages rationnels: calculer le monoïde syntaxique (ou l'automate minimal) puis tester l'apériodicité. Depuis, d'autres équivalences ont été établies entre fragments logiques et variétés de monoïdes, voir [Str94, DK09] (et aussi Fig. 1.2) pour plus de détails. La notion de variété de monoïde a aussi été généralisée aux variétés de monoïdes ordonnés [Pin95] et aux variétés de timbres [CPS06].

En ce qui concerne les transductions séquentielles on a, grâce à l'algorithme de minimisation de Choffrut, une façon de décider si une fonction séquentielle peut être réalisée par un transducteur séquentiel avec un monoïde de transition dans une variété donnée. Par exemple, les deux transducteurs de Fig. 2 ne sont pas apériodiques puisqu'ils comptent modulo deux. Cependant, bien que le premier est minimal le second ne l'est pas, et il se trouve que la fonction $a^n \mapsto a^{2n}$ est apériodique car elle peut être réalisée par un transducteur avec un unique état.

Pour la logique, une équivalence a été établie entre les transducteurs bidirectionnels fonctionnels avec monoïde de transition apériodique et les $\mathsf{FO}[\leq]$-transductions dans [CD15]. Un résultat similaire a été obtenu prouvant l'équivalence entre les SST avec monoïde de transition apériodique et les $\mathsf{FO}[\leq]$-transductions dans [FKT14]. Ces équivalences ne fournissent cependant pas de procédure pour décider si une transduction peut être réaliser par un $\mathsf{FO}[\leq]$-transducteur. En effet pour les fonctions régulières on ne connaît pas encore d'objet minimal avec de bonnes propriétés tel que l'automate minimal ou le monoïde syntaxique dans le cas des langages rationnels.

Un point de vue différent a été considéré dans [CKLP15] où les auteurs fournissent une procédure pour décider si une transduction séquentielle peut être exprimée par un circuit dans AC^0. Leur approche, qui utilise une notion de continuité des transduction par rapport à une variété de monoïde a été étendue dans [CCP17].

Problèmes de synthèse Le *problème de* $\mathcal{C}_1, \mathcal{C}_2$*-synthèse* demande, étant donné un transducteur T_2 dans une classe $\mathcal{C}_2$ avec une interprétation sémantique $[\![T_2]\!]$ et un domaine $\mathrm{dom}([\![T_2]\!])$, s'il existe un transducteur $T_1 \in \mathcal{C}_1$ uniformisant T_2, c'est-à-dire tel que $[\![T_1]\!] \subseteq [\![T_2]\!]$ et $\mathrm{dom}([\![T_1]\!]) = \mathrm{dom}([\![T_2]\!])$. En termes de vérification, le transducteur T_2 est nommé la *spécification*, qui est censée capturer les comportements entrée/sortie acceptables, et T_1, normalement fonctionnel, peut être vu comme un *programme* qui satisfait la spécification. Les problèmes de synthèse sont centraux en informatique puisqu'il s'agit d'obtenir automatiquement à partir d'une spécification un programme qui la satisfait. Les problèmes de synthèse jouent également un rôle important en théorie des jeux, où l'on souhaite produire une stratégie (programme) qui assure (satisfait) une condition de victoire (spécification). Pour plus de détails sur l'histoire et les implications des problèmes de synthèse voir [Tho09].

Le problème de synthèse de Church, énoncé dans l'article fondateur [Chu63] demande, étant donné un circuit réalisant une transduction lettre-à-lettre si l'on peut synthétiser un transducteur séquentiel lettre-à-lettre l'uniformisant. Dans [BL69] les auteurs montrèrent que si la spécification est donnée comme une formule MSO réalisant une transduction lettre-à-lettre, alors le problème est décidable. En utilisant comme spécifications des formules LTL, [PR89] ont fourni une solution élémentaire au problème de synthèse ce qui insuffla beaucoup d'intérêt dans le domaine de la synthèse réactive, par exemple la compétition SYNTCOMP co-située avec la conférence CAV depuis 2014. Depuis, d'autres classes de transductions ont été considérées avec des résultats positifs et négatifs. Dans [CL15], les auteurs ont montré que lorsque la restriction lettre-à-lettre est retirée, le problème devient indécidable. Cependant, ils ont également montré que si seulement la spécification est lettre-à-lettre alors le problème reste décidable. Dans [FJLW16] les auteurs montrent que le problème de synthèse séquentielle est décidable lorsque la spécification est une transduction *finiment valuée* (il existe une borne uniforme sur le nombre de sorties produites par une entrée). Ils prouvent également que le problème est décidable si la spécification est donnée par un transducteur *déterministe* (un automate déterministe à deux bandes). Ce problème a également été étendu aux structures arborescentes, par exemple dans [LW17].

Contributions de la thèse

Ce manuscrit traite de certaines instances des questions mentionnées précédemment, minimisation, définissabilité et synthèse pour les transductions, et il est divisé en deux parties. Dans la première partie nous considérons les fonctions rationnelles sur les mots, c'est-à-dire les fonctions réalisées par des transducteurs unidirectionnels, sur des mots finis et infinis. Une restriction naturelle des transducteurs MSO, nommés transducteurs *préservant l'ordre*, capture exactement les fonctions rationnelles, voir [Boj14, Fil15]. Plusieurs questions de définissabilité peuvent donc découler de cette remarque: étant donné un fragment logique F de MSO, peut on décider la F-définissabilité des fonctions rationnelles ? Dans la première partie nous considérons de telles questions que nous résolvons pour certains fragments (en particulier $\mathsf{FO}[\leq]$) par des techniques de minimisation et de formes canoniques. Dans la seconde partie nous étudions des logiques expressives pour les transductions et nous résolvons le problème de synthèse régulière pour ces logiques. Nous établissons également un lien fort entre les transductions et les langages de mots avec données ordonnées.

Caractérisation des fonctions rationnelles

L'objectif et la situation Dans ces travaux notre but a été d'étendre le succès de l'étude des équivalences logique-algèbre des langages vers les transductions, afin de résoudre des problèmes de définissabilité pour des fragments logiques de MSO. Pour les langages rationnels, la caractérisation effective de fragments de MSO repose souvent sur une équivalence avec une variété de monoïdes. Ce qui veut dire, d'après les bonnes propriétés des variétés, que calculer l'automate minimal est suffisant pour décider si un langage est reconnu par un monoïde dans la variété et donc par une formule du fragment. En ce qui concerne les transductions séquentielles, comme nous l'avons vu, un objet minimal existe également cependant toutes les fonctions rationnelles ne sont pas séquentielles.

Dans [RS91], les auteurs donnent une procédure pour calculer une machine canonique pour n'importe quelle fonction rationnelle. Les machines utilisées sont des *bimachines*, qui furent introduites par [Sch61] puis nommées et étudiées par [Eil74]. Une bimachine peut être vue comme un transducteur séquentiel avec "information en avant", c'est-à-dire avec une information rationnelle sur le suffixe du mot, où l'information en avant est donnée par un automate co-déterministe. En particulier, la bimachine canonique de [RS91] a l'automate d'information en avant minimal parmi toutes les bimachines réalisant la fonction. Cependant cette bimachine n'est pas minimale en termes de monoïde de transition.

Les résultats Notre première contribution principale est de montrer que pour une transduction sur les mots finis donnée, il existe un nombre fini de bimachines minimales réalisant la fonction (minimales en termes de monoïde de transitions). Ainsi nous obtenons un algorithme pour décider si une transduction donnée en entrée peut être réalisée par un transducteur avec un monoïde de transitions dans une certaine variété (décidable).

De plus nous donnons un théorème de transfert d'équivalences logique-algèbre pour les langage vers des équivalences logique-algèbre pour les transductions. Ce qu'il faut retenir de ce résultat de transfert est que étant donné F un fragment logique de MSO équivalent à une variété de monoïdes, si F a accès au prédicat d'ordre alors l'équivalence peut être étendue aux transductions.

En particulier on peut décider si une transduction donnée peut être exprimée en $\mathsf{FO}[\leq]$ logique du premier ordre, $\mathsf{FO}^2[\leq]$ logique du premier ordre avec deux variables, et $\mathsf{B}\Sigma_1[\leq]$ la clôture booléenne du fragment existentiel de $\mathsf{FO}[\leq]$. De plus, dans le cas des transductions apériodiques nous montrons que *toutes* les bimachines canoniques sont apériodiques, ce qui nous permet de prouver que la décision de la FO[$\leq$]-définissabilité des fonctions rationnelles est un problème PSpace-complet.

Sur les mots infinis, le tableau est moins complet. Premièrement nous étendons le résultat de minimisation de Choffrut aux fonctions séquentielles sur les mots infinis. La difficulté est que pour les langages de mots infinis, il n'y a pas de bonne définition d'automate minimal. Nous contournons ce problème en ignorant le domaine de la transduction, et nous obtenons un unique transducteur séquentiel minimal qui prolonge la fonction initiale sur la clôture de son domaine.

Nous définissons un automate canonique pour l'information en avant et nous obtenons une bimachine canonique pour les fonctions rationnelles sur les mots infinis. Cependant cette bimachine n'a pas les même bonnes propriétés que celle de [RS91]: l'automate d'information en avant n'a pas de propriété de minimalité. Néanmoins nous montrons que n'importe quelle transduction apériodique a une bimachine canonique apériodique et nous obtenons donc une procédure de décision de la FO[$\leq$]-définissabilité pour les fonctions rationnelles sur les mots infinis.

Spécification et synthèse pour les transductions

L'objectif Le but de ce travail était de définir un "bon" langage de spécification pour les transductions. Notre objectif était d'obtenir un formalisme de haut niveau pour spécifier des propriétés de transductions, pour lequel des problèmes classiques de vérification pourraient être résolus: l'équivalence, la vérification de modèle et la synthèse par exemple. Nos trois exigences principales étaient 1) ce formalisme devrait être de haut niveau, c'est-à-dire proche du langage humain, en d'autres termes une logique. 2) il devrait être suffisamment expressif et au moins pouvoir exprimer les fonctions régulières, qui jouissent de nombreuses caractérisations différentes notamment par des modèles d'automates déterministes ce qui semble pertinent pour la vérification de modèle et la synthèse. 3) certains des problèmes mentionnés plus haut devraient être décidables.

Un premier candidat évident serait le modèle de transducteurs NMSO de Courcelle. Ce modèle rempli les points 2) et 3) mais nous arguons qu'il ne satisfait pas le point 1), bien qu'étant basé sur MSO. Pour un transducteur NMSO on *doit* spécifier exactement comment l'entrée produit la sortie, on n'a pas la possibilité de *sous-spécifier.* Par exemple la spécification *"L'entrée contient au moins une fois la lettre a et la sortie au plus deux fois la lettre b"* ne peut être exprimée par une transduction NMSO. En effet pour un transducteur NMSO un mot d'entrée ne peut produire qu'un nombre fini de mots de sortie. Dans un certain sens, un transducteur NMSO est plus proche d'un modèle de machine où les déplacements de la tête de lecture sont définis par des formules MSO.

L'approche Notre approche est basée sur une remarque faite dans [Boj14], que la sémantique de presque n'importe quel modèle de transducteurs peut être enrichie en *sémantique avec origines.* Au lieu de paires de mots (u, v) où u est l'entrée et v la sortie, la sémantique avec origine considère des paires de la forme $(u, (v, o))$ où u est l'entrée, v la sortie et $o : \mathrm{dom}(v) \to \mathrm{dom}(u)$ est la *fonction d'origine* qui, intuitivement, envoie une position de la sortie sur la position de l'entrée qui était traitée lorsque la position de sortie a été produite.

On considère des *graphes d'origine*: des structures relationnelles à deux composantes, une structure d'entrée, une structure de sortie ainsi qu'une fonction d'origine des positions de sortie vers les positions d'entrée. Nous considérons principalement les graphes d'origine de mots vers mots, c'est-à-dire quand les deux sous-structures sont des mots. Nous donnons en Fig. 3(a) un graphe d'origine produit par l'automate de Fig. 2(a) et en Fig. 3(b) un graphe d'origine produit par l'automate de Fig. 2(b).

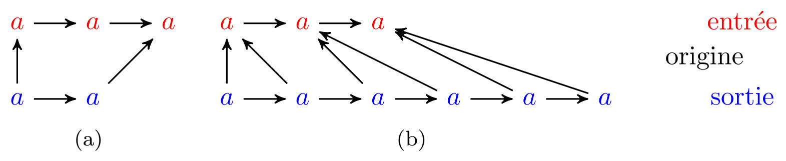


Figure 3: Deux graphes d'origine.

Nous considérons MSO sur les graphes d'origine et nous montrons que cette logique est indécidable (même pour des fragments faibles). Il faut cependant noter que dans [BDGP17] les auteurs montrent que la vérification de modèle d'une fonction régulière contre une formule MSO *est* décidable. Une façon d'obtenir ce résultat est d'observer que les graphes d'origine produits par un transducteur bidirectionnel déterministe ont une largeur de chemin bornée. En utilisant le théorème de Courcelle [Cou90] on obtient donc la décidabilité.

La logique Puisque la logique MSO entière est indécidable, notre approche a été de chercher un fragment suffisamment faible pour être décidable mais suffisamment expressif pour capturer *au moins* les fonctions régulières. Nous sommes parvenus à trouver une logique adéquate, après plusieurs essais, appelée Lo. Puisque nous voulions avoir toute l'expressivité de MSO sur l'entrée afin de capturer les fonctions régulière, nous avons défini une logique "asymétrique" dans le sens où elle est restreinte dans son expressivité en ce qui concerne les positions de sortie. La logique que nous définissons est la logique du premier ordre avec deux variables, avec l'ordre sur l'entrée et sur la sortie et une fonction d'origine. En plus de cela nous ajoutons n'importe quel prédicat MSO-définissable qui ne peut quantifier que sur l'entrée. Nous dénotons cette logique par Lo := $\mathsf{FO}^2[\leq_{\mathsf{out}}, \mathsf{o}, \mathsf{MSO}[\leq_{\mathsf{in}}]]$, où $\leq_{\mathsf{in}}$ et $\leq_{\mathsf{out}}$ sont les ordres respectivement sur l'entrée et sur la sortie et o est le symbole de fonction d'origine. Soit $\mathrm{even}(x, y)$ le prédicat binaire $\mathsf{MSO}[\leq_{\mathsf{in}}]$-définissable disant que la distance entre les positions d'entrée x et y est paire. Considérons la formule Lo suivante.

$$\forall x, y\ \mathsf{out}(x) \wedge \mathsf{out}(y) \wedge a(x) \wedge b(y) \to \langle \mathrm{even}(\mathsf{o}(x), \mathsf{o}(y)) \rangle$$

Les chevrons $\langle, \rangle$ sont seulement là pour séparer les prédicats $\mathsf{MSO}[\leq_{\mathsf{in}}]$, dans un soucis de clarté. Le prédicat $\mathsf{out}(x)$ est ajouté comme sucre syntaxique et peut être exprimé par exemple par $x \leq_{\mathsf{out}} x$. La formule dit que deux positions de sortie étiquetées par a et b respectivement doivent avoir des origines à distance paire.

Les résultats Nos techniques sont inspirées de [SZ12] où les auteurs obtiennent la décidabilité de la logique du premier ordre avec deux variables sur les mots avec données ordonnées, avec comme prédicats l'ordre sur les positions et l'ordre sur les données. Avant d'expliciter le lien entre graphes d'origine et mots avec données, discutons des résultats obtenus.

Nous montrons tout d'abord que le problème de satisfiabilité de Lo sur les graphes d'origine est décidable. En fait notre résultat est plus fort que cela puisque nous fournissons un algorithme pour calculer un automate reconnaissant le domaine d'une transduction Lo. Puisque la logique est close par opérations booléennes, nous obtenons la décidabilité de l'équivalence de formules. En raffinant nos techniques, nous obtenons alors le résultat principal de cette partie: un algorithme de synthèse régulière, c'est-à-dire une algorithme qui prend en entrée une formule Lo et qui produit une fonction régulière l'uniformisant.

Nous donnons également deux extensions de la logique pour lesquelles notre algorithme d'uniformisation fonctionne également. La première est $\exists$Lo qui étend Lo en ajoutant un bloc de quantificateurs monadiques existentiels. La seconde est $\mathsf{L}o^{\mathrm{so}}$ qui ajoute de nouveaux prédicats pouvant parler de la sortie produite par une unique position d'entrée. Alors que $\mathsf{L}o$ est incomparable en expressivité avec les transductions NMSO et les transductions rationnelles (unidirectionnelles), $\exists\mathsf{L}o$ et $\mathsf{L}o^{\mathrm{so}}$ englobent respectivement ces deux classes. Nous donnons les relations entre les classes de transductions connues en Fig. 4.2 et Fig. 4.10.

Finalement, nous exhibons un lien très étroit entre les transductions et les langages de mots avec données. Ce lien nous a guidé durant notre recherche d'une logique adéquate décidable: les résultats d'indécidabilité pour les logiques de mots avec données peuvent être traduits en résultats d'indécidabilité dans le contexte de logiques sur les graphes d'origine. Réciproquement, de la décidabilité de $\mathsf{L}o$, nous obtenons la décidabilité de $\mathsf{L}d := \mathsf{FO}^2[\leq, \mathsf{MSO}[\preccurlyeq]]$ sur les mots avec données où $\leq$ est l'ordre sur les positions, $\preccurlyeq$ est l'ordre sur les données et $\mathsf{MSO}[\preccurlyeq]$ dénote les prédicats MSO-définissables qui ne peuvent quantifier que sur les données et sur les ensembles de données. Par exemple on pourrait exprimer dans cette logique que le nombre de données différentes d'un mot est pair.

Contents

Introduction

From languages to transductions

The study of formal languages is one of the cornerstones of theoretical computer science and has been very fruitful in developing theoretical as well as practical tools in many domains. Some language classes stand out from the crowd because they enjoy several different descriptions, for instance the recursively enumerable languages which can be characterized both in terms of Turing machines or in terms of type-0 grammars in the Chomsky hierarchy. An example of particular interest to us is the class of rational languages[2] which sits on the lowest level of the Chomsky hierarchy as they are recognized by the regular grammars. The rational languages are also characterized by rational expressions, monadic second-order logic (MSO) formulas, finite monoids (or equivalently congruences of finite index), finite state automata (and all their variants: deterministic, non-deterministic, alternating, one-way, two-way, *etc*), *etc.* For instance the language of words of even length over the alphabet $\{a\}$ is recognized by the automaton of Fig. 4, and is defined by the rational expression $(aa)^*$.

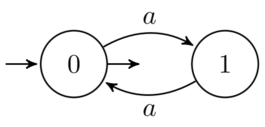


Figure 4: Deterministic one-way automaton recognizing $(aa)^*$.

The theory of computability can also be considered from the point of view of functions and there again particular classes enjoy many different descriptions, for instance the computable functions can be characterized in terms of Turing machines, μ-recursive functions or in the lambda calculus. In the model of Turing, a (non-deterministic) machine with an input tape and an output tape defines a relation on words over a finite alphabet, called a *transduction.*

An important question to us is: what is the class of transductions homologous to the rational languages? There is no straightforward answer to this question since the transduction classes we obtain depend on the computational model we choose to generalize. For instance the transductions *recognizable* by finite monoids are one of the lowest classes of transductions in terms of expressiveness and can be described as finite unions of products of rational languages (see *e.g.* [Ber79]).

Automata can be promoted to *transducers*, *i.e.* automata with outputs, to realize transductions. In Fig. 5 on the next page we give two examples of transducers. The first one copies half of the letters of its input while the second one doubles it. While an automaton can be thought

[2]Often called regular languages.

of as a Turing machine with a read-only tape, a transducer is a Turing machine with one read-only tape and one write-only tape. The different variants of automata all recognize the same class of languages, however this is not the case at all for transductions. Firstly, deterministic transducers can only realize (partial) functions since a particular configuration of the input tape only allows for one computation. Non-deterministic transducers on the other hand may realize non-functional relations. Secondly, one-way transducers are strictly less expressive than two-way transducers, for instance the mirror function, which reverses the order of a word, can be realized in the latter class but not in the former. The transductions realized by one-way transducers are also the transductions definable by rational expressions over a product of free monoids (see [Ber79]) and have thus been coined *rational transductions*. The functions realized by two-way transducers have also been characterized by different models, namely the MSO-transductions *à la* Courcelle as well as the so-called *streaming string transducers*, *i.e.* one way transducers with registers. The equivalences have been shown in [EH01, AC10], respectively, and the functions of this class are often called the *regular functions*. For an overview of the different models of transducers, see the surveys [Fil15, FR16].

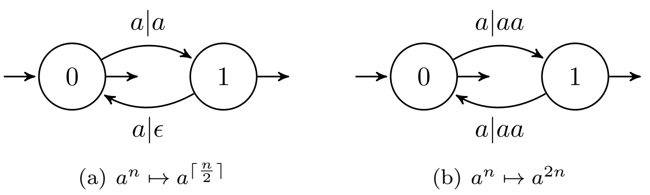


Figure 5: Two sequential transducers.

The problems

Classical problems in the theory of formal languages can be formulated in terms of transductions. We consider some minimization and definability questions and say what results have been obtained for rational languages and their extensions to transductions. Additionally new kinds of problems arise, and in particular synthesis questions for transductions have been extensively studied.

Minimization problems and canonical forms A minimization problem asks given a syntactic object M_2 (*e.g.* an automaton) in a class $\mathcal{C}$, with a semantic interpretation $[\![M_2]\!]$, whether there exists an object $M_1 \in \mathcal{C}$ such that $[\![M_1]\!] = [\![M_2]\!]$ and M_1 is "smaller" than M_2, for some definition of smaller. Minimization problems are pervasive in computer science, for obvious efficiency reasons. Moreover, minimal objects are often linked to *canonical* forms which have additional algorithmic interest. A canonical form for objects in a given class $\mathcal{C}$ is a procedure which takes as input an object $M \in \mathcal{C}$, and produces an object M' which only depends on $[\![M]\!]$ and such that $[\![M]\!] = [\![M']\!]$. In particular, having a procedure to compute a canonical form gives a way to test equivalence of two objects. Moreover such a canonical form may allow one to test other semantic properties of an object.

For rational languages, minimization has been solved for different models. The *syntactic monoid* (equivalently, syntactic congruence, see [Ner63]) of a rational language is *the* smallest monoid recognizing the language, and moreover this monoid is minimal in the strong sense that it divides any monoid recognizing the language. From this, one can define the *minimal deterministic* *automaton* of a language which is minimal in the number of states among other deterministic automata recognizing the same language. Again this minimality property is stronger than just having a minimal number of states: the minimal automaton is a quotient of any deterministic automaton recognizing the same language, and this extra structure yields efficient minimization procedures in PTime ([Moo56, Hop71]).

For sequential transducers (one-way transducers with a deterministic underlying automaton) a similar minimal object was discovered by Choffrut with the same strong minimality property that the minimal sequential transducer is a quotient of any sequential transducer realizing the function. Alternatively, one could say that the transition monoid of the minimal sequential transducer divides the transition monoid of any sequential transducer realizing the function. This can be found in the survey [Cho03] where the author also gives several PTime minimization procedures close to the automata case. The existence of such minimal objects for automata and sequential transducers has been translated in category theoretical terms in [CP17]. As an example, the transducer in Fig. 5(b) on the facing page can be minimized into a one state transducer.

A different take on minimization has been considered in [BGMP16, BGMP17] where the authors, given functional two-way transducer, are able to minimize the number of head reversals uniformly over all input words. Yet another approach in [DRT16, DJRV17] considers streaming string transducers, with some restrictions on the register operations, and is able to minimize the number of registers, using twinning properties *à la* Choffrut.

Definability problems The $\mathcal{C}_1$-*definability problem for* $\mathcal{C}_2$ asks, given a syntactic object M_2 (*e.g.* automaton, formula, *etc*) in a class $\mathcal{C}_2$, with a semantic interpretation $[\![M_2]\!]$, whether there exists an object $M_1 \in \mathcal{C}_1$ such that $[\![M_1]\!] = [\![M_2]\!]$. Let us first remark that minimization problems can be thought of as particular cases of definability problems, by taking $\mathcal{C}_1$ a class of small objects of $\mathcal{C}_2$. Definability problems occur in many areas of computer science when one wants to know if an object can be expressed in a given computational model.

A *logical fragment* of MSO is a subset of MSO-formulas defined by a syntactic restriction. For any such fragment F we have the associated F-definability problem for rational languages. Many of such problems have been solved through the theory of monoid *varieties* (sometimes called pseudovarieties, see [Str94]). Indeed many equivalences have been established between logical fragments and monoid varieties. Moreover, monoid varieties are crucially closed under division which means that a language is recognizable by a monoid in a pseudo-variety $\mathbf{V}$ if and only if its syntactic monoid is in $\mathbf{V}$. Furthermore, the Eilenberg theory of varieties, started in [Eil76], provides an equational description of monoid varieties, which gives in many cases a decision procedure to know if a language is recognizable by a monoid in a given variety.

The first example of such an equivalence is the Schützenberger/McNaughton-Papert Theorem ([Sch65, MP71]) which states that languages definable in first-order logic, $\mathsf{FO}[\leq]$, are exactly the languages recognized by aperiodic monoids. This provides a procedure to decide the $\mathsf{FO}[\leq]$-definability problem for rational languages: computing the syntactic monoid (or the minimal automaton) and then checking for aperiodicity. Since then, other equivalences have been established between logical fragments and monoid varieties, see [Str94, DK09] (and also Fig. 1.2 on page 17) for more on this. The notion of monoid varieties has also been generalized to varieties of ordered monoids [Pin95] and to *stamp varieties* [CPS06].

For sequential transductions, we also have, thanks to the Choffrut minimization algorithm, a way to decide if a given sequential function can be realized by a sequential transducer with a transition monoid in a given variety. For instance both transducers in Fig. 5 on the facing page are not aperiodic since they count modulo two. However, while the first one is minimal, the second one is not and in fact the function $a^n \mapsto a^{2n}$ is aperiodic because it can be realized by a one-state transducer (which is thus aperiodic).

Concerning logics, an equivalence has been established between functional two-way transducers with an aperiodic transition monoid and $\mathsf{FO}[\leq]$-transductions in [CD15]. A similar result was also obtained showing the equivalence between streaming string transducers with an aperiodic transition monoid and $\mathsf{FO}[\leq]$-transductions in [FKT14]. These equivalences however do *not* provide a procedure to decide if a given transduction can be realized by an $\mathsf{FO}[\leq]$-transducer. Indeed, regular functions lack a minimal object with good minimality properties such as the minimal automaton or the syntactic monoid for rational languages.

From another point of view, in [CKLP15] the authors provided a procedure to decide if a sequential transduction can be expressed as a circuit in AC^0. Their approach, which uses a notion of continuity of transductions with respect to a monoid variety, was extended in [CCP17].

Synthesis problems The $\mathcal{C}_1, \mathcal{C}_2$*-synthesis problem* asks, given a transducer T_2 in a class $\mathcal{C}_2$ with semantic interpretation $[\![T_2]\!]$ and domain $\mathrm{dom}([\![T_2]\!])$, if there exists a transducer $T_1 \in \mathcal{C}_1$ uniformizing T_2, *i.e.* such that $[\![T_1]\!] \subseteq [\![T_2]\!]$ and $\mathrm{dom}([\![T_1]\!]) = \mathrm{dom}([\![T_2]\!])$. In verification terms, the transducer T_2 is called the *specification*, which is supposed to capture acceptable input/output behaviors, and T_1, usually functional, can be thought of as a *program* that satisfies the specification. Synthesis problems are very central to computer science, since the goal is to obtain automatically from a specification, a program that satisfies it (or realizes it). Synthesis problems also play a huge role in game theory, where one often wants to produce a strategy (program) that ensures (satisfies) a winning condition (specification). For more on the history and implications of synthesis problems see [Tho09].

The Church synthesis problem, stated in the seminal article [Chu63], asks, given a circuit realizing a letter-to-letter transduction if one can synthesize a sequential letter-to-letter transducer uniformizing it. In [BL69] the authors showed that when the specification is given as a MSO-formula realizing a rational letter-to-letter transduction, then the problem is decidable. Using specifications in LTL instead, [PR89] provided an elementary solution to the synthesis problem which sparked a lot of interest in the domain of reactive synthesis, including the competition SYNTCOMP co-located with CAV since 2014. Since then many other classes have been considered with positive and negative results. In [CL15], the authors show that when the letter-to-letter restriction is removed, then the problem becomes undecidable. However they also show that if only the specification is required to be letter-to-letter, then the problem remains decidable. In [FJLW16] the authors show that the sequential synthesis problem is also decidable when the specification is a *finite-valued* rational transduction (there is a uniform bound on the number of different outputs an input word may produce). They also solve the problem when the specification is a given by a *deterministic* transducer (a deterministic two-tape automaton). This problem has also been extended to tree structures for instance in [LW17].

Contributions of the thesis

This manuscript deals with some instances of the questions mentioned above, minimization, definability and synthesis for transduction, and is divided into two main parts. In the first part we focus on rational word functions, that is functions realized by one-way transducers over words, finite and infinite. A natural restriction on MSO-transducers, called *order-preserving* MSO-transducers was shown to capture exactly the rational functions in [Boj14, Fil15]. From this fact many definability questions arise: for a logical fragment F of MSO, can one decide F-definability of rational functions? In the first part we consider such questions which we solve (for some fragments, including $\mathsf{FO}[\leq]$) using minimization and canonical forms for transducers.

In the second part we study expressive logics for transductions and solve the regular synthesis problem for them. We also establish a strong link between transductions and languages of data words with an ordered data domain.

Characterizations of rational functions

The goal and the setting In this work, our goal was to extend the successful study of logic-algebra equivalences from languages to transductions, to solve definability problems in logical fragments of MSO. For rational languages, the effective characterizations of logical fragments of MSO usually rely on an equivalence with a monoid variety. Then, by the good properties of varieties, computing the minimal automaton is enough to decide if a language is recognized by a monoid in the variety and thus by a formula in the fragment. For sequential transductions, as we have seen, a similar minimal device exists, but not all rational functions are sequential.

In [RS91], the authors give a procedure to compute a canonical machine for any rational function. The machines used in [RS91] are *bimachines*, which were introduced in [Sch61], and named and further studied in [Eil74]. Bimachines can be thought of as sequential transducers with look-ahead, where the look-ahead is given as a co-deterministic automaton. In particular, the canonical bimachine of [RS91] has *the* minimal look-ahead automaton among all bimachines realizing the function. However, this bimachine is not minimal in terms of transition monoid.

The results Our first main contribution is to show that for any given transduction over finite words, there exists a finite number of minimal bimachines realizing the function (minimal in terms of transition monoid). Thus we obtain an algorithm to decide if a given transduction can be expressed by a transducer with a transition monoid in a given (decidable) variety.

Furthermore we obtain a transfer theorem from logic-algebra equivalences over languages to logic-algebra equivalences over transductions. The main takeaway of this transfer theorem is that given F a logical fragment of MSO equivalent to some monoid variety[3], if F has access to the linear order predicate then the equivalence transfers to transductions.

In particular we are able to decide if a given transduction can be expressed in $\mathsf{FO}[\leq]$ first-order logic, $\mathsf{FO}^2[\leq]$ first-order logic with two variables, and $\mathsf{B}\Sigma_1[\leq]$ the boolean closure of the existential fragment of $\mathsf{FO}[\leq]$. Moreover, in the case of an aperiodic transduction, we are able to show that *all* its minimal bimachines are aperiodic, which allows us to show that deciding $\mathsf{FO}[\leq]$-definability of rational functions is PSpace-complete.

Over infinite words the picture is less complete. First we extend the minimization result of Choffrut to sequential functions over infinite words. The difficulty here is that for rational languages over infinite words, there is no known meaningful way to define a minimal automaton. We circumvent this issue by ignoring the domain of the transduction, and what we obtain is a unique minimal sequential transducer that extends the original one over the topological closure of its domain.

We define a canonical look-ahead automaton and from that we obtain a canonical bimachine for any rational function over infinite words. However this bimachine does *not* have the same nice property as the one in [RS91] over finite words: its look-ahead automaton has no minimality property. We nonetheless show that any aperiodic transduction has an aperiodic canonical bimachine and thus we obtain a decision procedure of $\mathsf{FO}[\leq]$-definability for rational functions over infinite words.

Specification and synthesis for transductions

[3]We actually consider more general classes of monoids

The goal The goal of this work was to define a "good" specification language for transductions. Our objective was to obtain a high-level formalism to specify properties of transductions, for which classical verification problems could be solved: equivalence, model-checking and synthesis for instance. Our three major requirements were 1) this formalism should be high-level, *i.e.* close to a human language, in other words a logic. 2) it should be expressive enough, and our yardstick for expressiveness was the class of regular functions, which enjoy many different characterizations including deterministic automata models which seem relevant for model-checking and synthesis. 3) some of the aforementioned problems should be decidable.

A first obvious candidate for this are the NMSO-transducers of Courcelle. This model definitely fulfills requirements 2) and 3), but let us argue why, even though it is based on MSO it does not fit the bill for 1), in our view. In an NMSO-transducer you *have* to specify exactly how the input produces the output, you do not have the possibility to *underspecify*. For instance for the specification *"The input contains at least once the letter a and the output at most twice the letter b"*, there are no NMSO-transducer realizing all pairs of input/output words that satisfy this specification. Indeed, in an NMSO-transducer each input word can only produce a finite number of output words. In some sense an NMSO-transducer is closer to a machine model where the moves are defined by MSO-formulas.

The approach Our approach is based on a remark made in [Boj14], that the semantics of almost any reasonable transducer model can be extended to a richer *origin semantics*. Instead of relating pairs of words (or other structures) (u, v) where u is the input structure and v the output structure, the origin semantics considers pairs of the form $(u,(v,o))$ where u is the input structure, v is the output structure and $o : \mathrm{dom}(v) \to \mathrm{dom}(u)$ is the *origin function* which intuitively maps an output position to the input position that was being processed when the output position was produced.

We consider *origin-graphs*: two-sorted relational structures with an input structure, an output structure and an additional origin function from output positions to input positions. We mainly focus on word-to-word origin-graphs: origin-graphs where the two substructures are words. We give in Fig. 6(a) an origin graph produced by the automaton from Fig. 5(a) on page 2 and in Fig. 6(b) an origin-graph produced by the automaton in Fig. 5(b) on page 2.

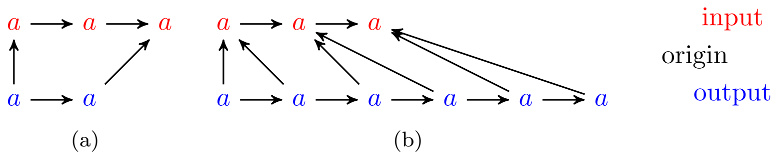


Figure 6: Two origin graphs.

We consider MSO over origin-graphs and we show that the logic is undecidable (even for weak fragments). Note however that in [BDGP17] the authors show that the model-checking of regular functions against MSO-formulas *is* decidable. One way to obtain the result is to see that the origin-graphs produced by a two-way deterministic transducer have bounded path-width, and thus, using Courcelle's theorem [Cou90] we obtain the decidability.

The logic Since the full MSO logic is undecidable, our approach was to find a fragment of MSO weak enough to be decidable, but still expressive enough to capture *at least* the regular functions. We managed to find a suitable logic, after some trial and error, called L*o*. Since we wanted the full power of MSO over the input, in order to capture the regular function, we defined an "asymmetric" logic in the sense that it is restricted in the way it can talk about output positions. The logic we came up with is first-order logic with two-variables, with the linear order predicate over the input and the output, and with an origin function symbol. To this we added *arbitrary* MSO-definable predicates that can only quantify over input positions. We denote the logic by $\mathsf{L}o := \mathsf{FO}^2[\leq_{\mathsf{out}}, \mathsf{o}, \mathsf{MSO}[\leq_{\mathsf{in}}]]$, where $\leq_{\mathsf{in}}$ and $\leq_{\mathsf{out}}$ are the linear orders over the input and output, respectively and o is the origin function symbol. Let $\mathrm{even}(x, y)$ be the binary $\mathsf{MSO}[\leq_{\mathsf{in}}]$-predicate stating that the distance between positions x and y is even. Let us consider the following $\mathsf{L}o$ formula.

$$\forall x, y\ \mathsf{out}(x) \wedge \mathsf{out}(y) \wedge a(x) \wedge b(y) \rightarrow \langle \mathrm{even}(\mathsf{o}(x), \mathsf{o}(y)) \rangle$$

The brackets $\langle, \rangle$ are just here to separate the $\mathsf{MSO}[\leq_{\mathsf{in}}]$-predicates, for readability. The predicate $\mathsf{out}(x)$ is added as syntactic sugar and can be expressed for instance by $x \leq_{\mathsf{out}} x$. The formula states that two output positions labeled by a and b respectively must have origins at even distance.

The results Our techniques are inspired by [SZ12], where the authors obtain decidability for first-order logic with two variables over words with linearly ordered data, with the linear order over positions and the order over data. Before explaining the link between origin-graphs and data words, let us talk about our results.

We first show that the satisfiablility of $\mathsf{L}o$ over origin-graphs is decidable. Actually our result is stronger than that since we show a way to compute, for any $\mathsf{L}o$-transduction, an automaton recognizing its domain. Since the logic is syntactically closed under boolean operations, we obtain that the equivalence of formulas is also decidable. By refining our techniques, we are even able to obtain the main result of this part: a regular synthesis algorithm, *i.e.* an algorithm that takes as input an $\mathsf{L}o$ formula and produces a regular function uniformizing it.

We also give two extensions of the logic for which our uniformization algorithm still works. The first one is $\exists\mathsf{L}o$ which extends $\mathsf{L}o$ by adding a block of existential monadic quantifiers. The second one is $\mathsf{L}o^{\mathrm{so}}$ which add new unary predicates that can talk about the output produced by a single input position. While $\mathsf{L}o$ is incomparable in expressiveness with both NMSO-transductions and one-way transductions, $\exists\mathsf{L}o$ and $\mathsf{L}o^{\mathrm{so}}$ subsume these two classes, respectively. We give the relations between known classes of transductions in Fig. 4.2 on page 99 and Fig. 4.10 on page 119.

Finally we discovered a very tight connection between transductions and languages over data words. This link helped us during our search of a suitable decidable logic for transductions: undecidability results of logics for words with ordered data can be translated into undecidability results in the context of logics over origin-graphs. Conversely, from the decidability of $\mathsf{L}o$, we obtain the decidability of $\mathsf{L}d := \mathsf{FO}^2[\leq, \mathsf{MSO}[\preccurlyeq]]$ over data words where $\leq$ is the linear order over positions, $\preccurlyeq$ is the order over data and $\mathsf{MSO}[\preccurlyeq]$ denotes any MSO-definable predicate that can only quantify over data and sets of data. For instance one could express that the number of data values between the data of two positions is even.

Part I

Computational, Logical and Algebraic Characterizations of Rational Word Functions

Chapter 1

Rational languages and rational functions

"In the beginning there was nothing, which exploded."

– Terry Pratchett, *Lords and ladies*

Rational languages enjoy many different characterizations including finite automata, rational expressions, monadic second-order logic and recognizability by congruences of finite index. In this chapter we present important results concerning some of these characterizations and the links between them, however we assume *some* knowledge of these notions and refer the reader to the textbooks [Str94, Sak09, Pin18] for a good overview of the subject. More precisely, we mention the tight connections existing between logical fragments of monadic second-order logic and varieties of congruences which are, basically, sets of congruences with good closure properties. These links, together with the existence of a syntactic congruence characterizing the algebraic properties of a language, have yielded many definability results. A paramount example is the Schützenberger/McNaughton-Papert theorem which gives a procedure to decide if a given language is definable in first-order logic.

Rational functions also enjoy different characterizations such as finite transducers, logical transducers *à la* Courcelle and rational expressions. In the case of functions however, much less is known of the links between logics and algebra. Our main goal, throughout Chap. 2 and 3 will be trying to establish such links.

1.1 Words and languages

1.1.1 Words

An *alphabet* is a finite set of symbols called *letters*. Let $\mathbb{N}$ denote the set of natural numbers and $\mathbb{N}_*$ denote the set of positive integers. Let $n \in \mathbb{N}_*$ and let $\mathbf{n}$ denote the set $\{1, \dots, n\}$ of positive integers less than or equal to n. A *finite word* w of *length* n over an alphabet A is a sequence of n letters of A. Formally, it is a function $w : \mathbf{n} \to A$. The length of w is denoted by $|w|$ and the set of words of length n is denoted by A^n. Furthermore we represent the word w by the sequence of its letters $w = w(1) \cdots w(n)$, and $w(i)$ is called the ith *position* of w. As a convention, the *empty word* is the unique word of length 0 and is written ϵ. An ω-*word* (or infinite word) is an infinite sequence of letters, *i.e.* a function $x : \mathbb{N}_* \to A$, and as for finite words we often write $x = x(1)x(2)\cdots$ and we set $|x| := \infty$. Given an infinite word x, we denote by $\mathrm{Inf}(x)$ the set of letters which appear infinitely often in x.

Concatenation The *concatenation* of a finite word u and a finite (or infinite word) w, is the word $u \cdot w$ (or simply uw) of length $|u|+|w|$ (with the convention that $\forall x \in \mathbb{N} \uplus \{\infty\}$ $x+\infty = \infty$) defined by $uw(i) := u(i)$ if $1 \leq i \leq |u|$, $uw(i) := w(i-|u|)$ if $|u| < i \leq |u|+|w|$. Note that the concatenation of two words u, v when u is infinite is undefined. The set of all finite words $A^* = \bigcup_{n\in\mathbb{N}} A^n$ together with the concatenation of words is a monoid whose identity element is the empty word ϵ. The set of non-empty words[1] is denoted by A^+, the set of infinite words is denoted by A^ω and the set of all words, finite or not, is denoted by A^∞. Given a finite word $u \in A^*$, we define the *ω-power* of u by $u^\omega = uuu\cdots$, with the convention that $\epsilon^\omega := \epsilon$. Given a sequence of finite words $u_1, \dots, u_n$ (u_n may even be infinite), we denote the concatenation (in the obvious order) of these words by $\prod_{i\in\mathbf{n}} u_i$. Similarly, given an infinite sequence of finite words $u_1, u_2, \dots$ we write $\prod_{1\leq i} u_i$ to denote the infinite concatenation of these words.

Rational words We say that a word w is *rational* if there exist finite words $u, v \in A^*$ such that $w = uv^\omega$. In particular any finite word u is rational since $u = u\epsilon^\omega$. Note that a given infinite word may have different decompositions, such as $(ba)^\omega = b(abab)^\omega$. We say that a word uv^ω is in *normal form* – with respect to some alphabetic order – if either $v = \epsilon$, which means that the word is finite, or v is minimal in the lexicographic order, among possible decompositions of the word, and v is not a suffix of u. For instance, the normal form of $(ba)^\omega$ is $b(ab)^\omega$.

Factors of a word Given two words $u \in A^\infty$, $v \in A^\infty$, u is a *strict prefix* of v if there exists a non-empty word $w \in A^\infty$ such that $uw = v$ which we denote by $u < v$, and we let $u^{-1}v$ denote this word w. Given two words $u, v \in A^\infty$, we say that u is a *prefix* of v, denoted by $u \leq v$, if either $u = v$ or $u < v$, and in the first case we set $u^{-1}v := \epsilon$. We extend this notation: for $K, L \subseteq A^\infty$ we set $K^{-1}L := \{w \mid \exists (u,v) \in K \times L,\ w = u^{-1}v\}$. Given a non-empty word $w \in A^\infty$, and integers $1 \leq i \leq j \leq |w|$ the prefix of w of length i is denoted by $w(:i)$, the *suffix* of w from position i is the word $w(i:) := \prod_{i\leq k\leq |w|} w(k)$ and the *infix* (or *factor*) of w from position i to position j is the word $w(i:j) := w(i)\cdots w(j)$. As a convention, for any word $w \in A^\infty$, $w(:0) := \epsilon$ and $w(|w|+1:) := \epsilon$. Two infinite words are called *ultimately equal* if they have a common infinite suffix.

1.1.2 Languages

A *language* (resp. ω-language, ∞-language) is a set of finite words (resp. infinite words, words) and for $L \subseteq A^\infty$ a non-empty language, $\bigwedge L$ denotes the *longest common prefix* (*i.e.* the infimum with respect to $\leq$) of L. We denote by $u \wedge v$ the longest common prefix of $\{u, v\}$, and the *delay* between u and v is the pair $\mathsf{del}(u,v) = \big((u\wedge v)^{-1}u, (u\wedge v)^{-1}v\big)$. For instance $aba \wedge abb = ab$ and $\mathsf{del}(aba, abb) = (a, b)$. We define the *prefix language* of a language $L \subseteq A^\infty$, by $\downarrow L := \{u \mid \exists v \in L,\ u \leq v\}$.

Metrics We define two different metrics over words, the delay distance which varies as the size of the delay, and the prefix distance which varies as the inverse of the size of the longest common prefix. We will mostly use the delay distance in Chap. 2 and the prefix distance in Chap. 3.

[1] A^* and A^+ are usually called the free monoid and the free semigroup generated by A, respectively.

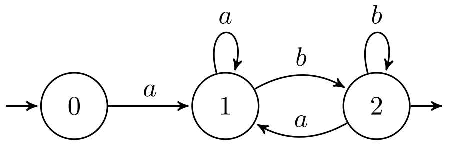


Figure 1.1: Left automaton recognizing aA^*b.

The *delay distance* between two words $u, v \in A^\infty$ is defined by $\|u,v\|_{\mathsf{del}} := |\mathsf{del}(u,v)|$ (where $|(x,y)| = |x| + |y|$). The *prefix distance* between two words $u, v \in A^\infty$ is defined by $\|u,v\|_\wedge := 0$ if $u = v$, and $\|u,v\|_\wedge := 2^{-|u \wedge v|}$ otherwise. The *topological closure* of a language $L \subseteq A^\omega$ with respect to the prefix distance can be alternatively defined as: $\overline{L} := \{u \mid \ \forall v < u, \ \exists x \ vx \in L\}$.

1.2 Finite automata

1.2.1 Automata

Pre-automata A *pre-automaton*, over an alphabet A is a tuple $\mathcal{A} = (Q, \Delta, I)$ where Q is the finite set of *states*, $\Delta \subseteq Q \times A \times Q$ is the *transition relation* and $I \subseteq Q$ is the set of *initial* states. A *run* of $\mathcal{A}$ over a word $w \in A^\infty$ is itself a word $r \in Q^\infty$ such that $|r| = |w| + 1$ and for all $1 \leq i < |r|$ we have $(r(i), w(i), r(i+1)) \in \Delta$. A run r is *initial* if $r(1) \in I$. We use the notation $p \xrightarrow{u}_{\mathcal{A}} q$ (or just $p \xrightarrow{u} q$) to denote that there is a finite run r of $\mathcal{A}$ over u such that $r(1) = p$ and $r(|r|) = q$. We write $p \xrightarrow{x}_{\mathcal{A}} P$ to denote that there is an infinite run r of $\mathcal{A}$ over x such that $r(1) = p$ and $\mathrm{Inf}(r) = P$.

Word automata We define three different models of automata: one over finite words, just called automata, and two over infinite words, namely Büchi automata and Muller automata. We use the generic term of *word automata* to refer to all three types at once, and ω-*automata* to refer to Büchi or Muller automata. An *automaton* (of finite words) over an alphabet A is a tuple $\mathcal{A} = (Q, \Delta, I, F)$ such that (Q, Δ, I) is a pre-automaton, and $F \subseteq Q$. A run r of an automaton is *final* if it is finite and $r(|r|) \in F$. A *Büchi automaton* is a tuple $\mathcal{A} = (Q, \Delta, I, F)$ such that (Q, Δ, I) is a pre-automaton, and $F \subseteq Q$. A run r of a Büchi automaton is *final* if it is infinite and $\mathrm{Inf}(r) \cap F \neq \varnothing$. A *Muller automaton* is a tuple $\mathcal{A} = (Q, \Delta, I, F)$ such that (Q, Δ, I) is a pre-automaton, and $F \subseteq \mathbf{2}^Q$ (the powerset of Q). A run r of a Muller automaton is *final* if it is infinite and $\mathrm{Inf}(r) \in F$. We will sometimes need a notion of word automata without initial states, which we will call *final* automata. A run of a word automaton is *accepting* if it is both initial and final and we denote by $[\![\mathcal{A}]\!]$ the language *recognized* by $\mathcal{A}$, *i.e.* the set of words over which $\mathcal{A}$ has an accepting run. A language (resp. ω-language) is *rational* if it is recognized by some automaton (resp. ω-automaton).

Example 1.2.1. The automaton $\mathcal{A}$ of Fig. 1.1 recognizes the language L of words which start with the letter a and end with b, *i.e.* $L = aA^*b$, and we have $\bigwedge L = a$. The initial state 0 is denoted by an ingoing arrow and the final state 2 by an outgoing arrow. $\mathcal{A}$ is trim and deterministic but is not co-deterministic since, for instance, the word b has two distinct final runs.

The same automaton seen as a Büchi automaton, recognizes the language $K = a(a^*b)^\omega$ of words which start with an a and have an infinite number of bs.

1.2.2 Subclasses of automata

A state q of a word automaton is *accessible* (resp. *co-accessible*) if there exists an initial (resp. final) run ending (resp. starting) in q. By extension, a word automaton is called *accessible* (resp. *co-accessible*) if all of its states are. A *trim* automaton is an automaton which is both accessible and co-accessible. An automaton is *deterministic*[2] (resp. co-deterministic) if it has at most one initial (resp. final) run per word. An automaton is *unambiguous* if it has at most one accepting run per word. An automaton over finite words (resp. infinite words) is *complete* if it has an initial run over any finite word (resp. infinite word). Symmetrically we define *co-completeness* by considering final runs instead. An automaton over finite words is called *left automaton* (resp. *right automaton*) if it is accessible (resp. co-accessible) and deterministic (resp. co-deterministic).

Determinization The famous *powerset construction*, due to [RS59], takes as input an automaton (over finite words) and produces a deterministic automaton recognizing the same language, at the cost of an exponential blow-up.

Theorem 1.2.2. *From an automaton recognizing a language L of finite words, one can obtain a deterministic automaton recognizing L.*

Büchi automata cannot be determinized in general since some rational languages can only be recognized but non-deterministic Büchi automata. However, one can transform any ω-automaton into a deterministic Muller automaton [McN66].

Theorem 1.2.3. *From an ω-automaton recognizing a language L, one can obtain a deterministic Muller automaton recognizing L.*

Although deterministic Büchi automata do not capture the whole class of rational languages over infinite words, it was shown in [CM03] that any rational ω-language can be recognized by a co-deterministic co-complete Büchi automaton, sometimes called *prophetic* or *backward deterministic* automata.

Theorem 1.2.4. *From an ω-automaton recognizing a language L, one can obtain a co-complete co-deterministic Büchi automaton recognizing L.*

1.3 Algebraic characterization of rational languages

In this section we present the algebraic aspect of rational languages. The *syntactic congruence* associated with a language characterizes it and, as its name suggests, it is not dependent on a particular description of the language. This yields an object which allows one to decide intrinsic properties of language, such as definability in a given logic (see Sec. 1.4).

1.3.1 Congruences

Let $\sim$ be an equivalence relation over a set S. Given $s \in S$, we denote by $[s]_\sim$ (or just $[s]$ when it is clear from context) the *equivalence class of* s, *i.e.* the set $\{t \mid \ t \sim s\}$. The *index* of a relation is the number of its equivalence classes. Given two equivalence relations $\sim_1$ and $\sim_2$ over S, we say that $\sim_1$ is *finer* than $\sim_2$ (or that $\sim_2$ is *coarser* than $\sim_1$), if any equivalence class of $\sim_2$ is a union of equivalence classes of $\sim_1$. In other words $\sim_1$ is included in $\sim_2$, seen as subsets of

[2]Here we choose, as in [CPP08], *global* definitions of determinism and co-determinism which coincide with the usual notions in the case of trim automata.

$S \times S$, which we denote by $\sim_1 \sqsubseteq \sim_2$. An equivalence relation $\sim$ over A^* is a *right congruence* if for any words $u, v \in A^*$ and any letter $a \in A$, $u \sim v \Rightarrow ua \sim va$. An equivalence relation $\approx$ over A^* (resp. A^ω) is a *left congruence* if for any words $u, v \in A^*$ (resp. A^ω) and any letter $a \in A$, $u \approx v \Rightarrow au \approx av$. A *congruence* over A^* is an equivalence relation which is both a left and a right congruence. The intersection of two right (resp. left) congruences $\sim_1, \sim_2$ is also a right (resp. left) congruence which we denote by $\sim_1 \sqcap \sim_2$. Let $\equiv$ be a congruence over A^*, the quotient $A^*/_{\equiv} = \{[u] \mid u \in A^*\}$ is naturally endowed with a monoid structure with a product defined by $[u] \cdot [v] = [uv]$ and identity element $[\epsilon]$. A congruence $\equiv$ of finite index over A^* can be seen as a surjective monoid morphism[3] from A^* to the finite monoid $A^*/_{\equiv}$. Hence a finite index congruence can be finitely described by a finite monoid M and a function $A \to M$. In the following we assume that finite index congruences are given that way.

Recognizability We say that a language over finite words $L \subseteq A^*$ is *recognized* by a right (resp. left) congruence $\sim$ if $L = \bigcup_{u \in L} [u]$, *i.e.* it is a union of equivalence classes of $\sim$.

The algebraic notion of recognizability for ω-languages is more intricate than in the finite case. There were several attempts, such as *weak recognizability* which lacked good properties (*e.g.* the existence of a minimal syntactic object), and *strong recognizability*, before reaching the more rewarding notions of ω-semigroups and Wilke algebras. In this manuscript we will only talk about strong recognizability (called simply recognizability), but we refer the interested reader to [CPP08] for more details concerning all these different notions. A language of infinite words $L \subseteq A^\omega$ is *recognized* by a congruence $\equiv$ if we have: $L = \bigcup_{uv^\omega \in L} [u]\,[v]^\omega$. Notice that if we have $[u]\,[v]^\omega \cap L \neq \varnothing$ then we must have $[u]\,[v]^\omega \subseteq L$ which is exactly the characterization of *saturation* given in [Arn85, Lemma 1.1].

Syntactic congruences Let L be a language of finite words over A, we define the *syntactic right congruence* $\sim_L$, the *syntactic left congruence* $\approx_L$ and the *syntactic congruence* $\equiv_L$ of L:

- $u \sim_L v$ if $\forall x \in A^*$, $ux \in L \Leftrightarrow vx \in L$
- $u \approx_L v$ if $\forall x \in A^*$, $xu \in L \Leftrightarrow xv \in L$
- $u \equiv_L v$ if $\forall x, y \in A^*$, $xuy \in L \Leftrightarrow xvy \in L$

One can easily show that each of these relations recognizes L and that they are the coarsest to do so among right congruences, left congruences and congruences, respectively.

Given a right congruence $\sim$, one can obtain the *associated* (complete deterministic accessible) pre-automaton defined by $\mathcal{A}_\sim := (A^*/_\sim, \{([u], a, [ua]) \mid a \in A, u \in A^*\}, \{[\epsilon]\})$. Given a language L recognized by $\sim$, then the automaton $(\mathcal{A}_\sim, \{[u] \mid u \in L\})$ recognizes L. Symmetrically, given a left congruence $\approx$, one can obtain the *associated* (co-complete co-deterministic co-accessible) final automaton defined by $\mathcal{A}_\approx := (A^*/_\approx, \{([au], a, [u]) \mid a \in A, u \in A^*\}, \{[\epsilon]\})$ (note that here $\{[\epsilon]\}$ is the set of *final* states).

The well-known Myhill-Nerode theorem [Ner63] states that a language L is rational if and only if its syntactic right congruence has finite index. Furthermore $\sim_L$ gives a description of *the minimal automaton* of a language L by $\mathcal{A}_L = (\mathcal{A}_{\sim_L}, \{[u] \mid u \in L\})$.

Theorem 1.3.1. *Given a language of finite words $L \subseteq A^*$, $\sim_L$ has finite index if and only if L is rational. Furthermore, the automaton $\mathcal{A}_L$ has the least possible number of states among complete deterministic automata recognizing L.*

[3]Such a morphism is called a *stamp* in [CPS06].

Of course the same result holds if we consider the syntactic left congruence and co-complete co-deterministic automata instead.

Remark 1.3.2. Given a left automaton $\mathcal{L}$ with state space Q, there is a bijection (up to adding a sink state) between Q and the equivalence classes of $\sim_{\mathcal{L}}$. For this reason we will often identify states of $\mathcal{L}$ with the equivalence classes of $\sim_{\mathcal{L}}$. Symmetrically we will identify the states of a right automaton $\mathcal{R}$ with the equivalence classes of $\approx_{\mathcal{R}}$. Abusing notations we will often write $\mathcal{L}_1 \sqsubseteq \mathcal{L}_2$ to denote that $\sim_{\mathcal{L}_1} \sqsubseteq \sim_{\mathcal{L}_2}$. Symmetrically, we will write $\mathcal{R}_1 \sqsubseteq \mathcal{R}_2$ instead of $\approx_{\mathcal{R}_1} \sqsubseteq \approx_{\mathcal{R}_2}$.

The minimal automaton of a language is actually minimal in a stronger sense than just having the least number of states among the deterministic complete automata recognizing it. Given a complete deterministic automaton recognizing a language, it (seen as a right congruence) *refines* the minimal automaton. Using this, a deterministic automaton can be minimized, in PTime, to obtain *the* minimal automaton (up to isomorphism) of the language [Moo56, Hop71].

Theorem 1.3.3. *Given a deterministic automaton recognizing a language of finite words L, one can obtain the minimal automaton (up to isomorphism) of L in* PTime.

The correspondence between automata and congruences over infinite words is less simple than in the case of finite words. Let $L \subseteq A^\omega$ be an ω-language, we define $\approx_L$ the *syntactic left congruence* of L, just like in the case of finite words, by $x \approx_L y$ if $\forall u \in A^*$, $ux \in L \Leftrightarrow uy \in L$. The definition of *syntactic congruence* of an ω-language is due to [Arn85] and is a bit more involved, but intuitive in the sense that for two words to be equivalent, they must behave the same whatever the context, and also when raised to the ω power.

$$u \equiv_L v \text{ if } \forall x, y, z \left[\begin{array}{l} x(uy)^\omega \in L \Leftrightarrow x(vy)^\omega \in L \text{ and} \\ xuyz^\omega \in L \Leftrightarrow xvyz^\omega \in L \end{array} \right.$$

Again one can easily see that $\equiv_L$ is the coarsest congruence recognizing L.

Transition congruences Let $\mathcal{A}$ be a word automaton with state space Q, we define the *right transition congruence* $\sim_{\mathcal{A}}$, the *left transition congruence* $\approx_{\mathcal{A}}$ and the *transition congruence* $\equiv_{\mathcal{A}}$ of $\mathcal{A}$:

- $u \sim_{\mathcal{A}} v$ if $\forall q \in Q$ there exists an initial run over u ending in $q \Leftrightarrow$ there is an initial run over v ending in q
- $u \approx_{\mathcal{A}} v$ if $\forall q \in Q$ there exists a final run over u beginning in $q \Leftrightarrow$ there is a final run over v beginning in q
- $u \equiv_{\mathcal{A}} v$ if $\forall p, q \in Q$ there exists a run over u from p to $q \Leftrightarrow$ there is one over v

The following proposition states that the transition congruences of an automaton recognize the language of the automaton.

Proposition 1.3.4. *Let L be a language of finite words recognized by $\mathcal{A}$, then $\sim_{\mathcal{A}}$, $\approx_{\mathcal{A}}$, $\equiv_{\mathcal{A}}$ all recognize L and are thus finer than their respective syntactic relations $\sim_L$, $\approx_L$ and $\equiv_L$.*

Example 1.3.5. Let $A = \{a, b\}$, let $L = aA^*b$ and let $\mathcal{A}$ be the automaton recognizing L of Fig. 1.1 on page 13. We have that $\sim_L = \sim_{\mathcal{A}}$, $\approx_L = \approx_{\mathcal{A}}$ and $\equiv_L = \equiv_{\mathcal{A}}$. Let us describe these equivalence relations by their equivalence classes: $A^*/_{\sim_L} = \{\epsilon, a + aA^*a, bA^*, aA^*b\}$, $A^*/_{\approx_L} = \{\epsilon, A^*a, b + bA^*b, aA^*b\}$ and $A^*/_{\equiv_L} = \{\epsilon, a + aA^*a, b + bA^*b, aA^*b, bA^*a\}$.

Languages	Logic	Variety	Equation
Rational	$\mathsf{MSO}[\leq]$	**All**	-
Star-free	$\mathsf{FO}[\leq]$	**A**	$x^\pi = x^{\pi+1}$
Unambiguous polynomials	$\mathsf{FO}^2[\leq]$	**DA**	$(xy)^\pi x(xy)^\pi = (xy)^\pi$
Piecewise testable	$\mathsf{B}\Sigma_1[\leq]$	**J**	$y(xy)^\pi = (xy)^\pi = (xy)^\pi x$
Idempotent		**I**	$x = x^2$

Figure 1.2: Well-known correspondences for languages of finite words.

Seen as a Büchi automaton $\mathcal{A}$ recognizes $K = a(a^*b)^\omega$. The syntactic congruence of K is given by $A^*/_{\equiv_K} := \{\epsilon, a^+, bA^*, aA^*bA^*\}$. Notice that $\equiv_K \neq \equiv_{\mathcal{A}}$ and furthermore, $\equiv_{\mathcal{A}}$ does not even recognize[4] K. For instance the word $a \cdot (aba)^\omega \in K$, however $[aba]_{\mathcal{A}} = aA^*a = [aa]_{\mathcal{A}}$ but $[a]_{\mathcal{A}} ([aa]_{\mathcal{A}})^\omega \not\subseteq K$.

1.3.2 Classes of congruences

A *class of congruences* (class for short) $\mathbf{C}$ associates with any finite alphabet A a set of congruences of finite index $\mathbf{C}(A)$ which is 1) closed under intersection of congruences 2) closed under taking coarser congruences[5]. Note that we will often abuse notations and write $\mathbf{C}$ instead of $\mathbf{C}(A)$. A class $\mathbf{C}$ is *decidable* if given a finite index congruence one can decide if it belongs to $\mathbf{C}$. In that case given a language, one can decide if it is a $\mathbf{C}$-language

Let A be an alphabet and let $\equiv$ be a congruence of finite index over A^*. We define π the *idempotent power* of $\equiv$ as the smallest positive integer such that for any word $u \in A^*$, $u^\pi \equiv u^{2\pi}$. The study of *varieties of finite monoids* (which can be seen as a particular case of classes of congruences, with more closure properties, and are sometimes called *pseudovarieties*), see *e.g.* [Str94], has been very successful in the algebraic characterization of rational languages and has yielded many links between logics and algebra. These varieties enjoy an equational characterization through Eilenberg variety theory [Eil76]. The point of such equations is that varieties for which the explicit equations are known are decidable since one only has to check a finite number of equalities. For instance the variety of idempotent congruences is characterized by the equation $x = x^2$ meaning that a congruence $\equiv$ is idempotent if it satisfies $u \equiv u^2$ for any word u. We give in the table of Fig. 1.2 some well-known congruence classes with their corresponding equations, and logics and we refer the reader to [DGK08] for a more precise description of these classes of languages. Actually the symbol π in the equations represents something more subtle than the idempotent power of a finite monoid in the theory of *profinite topology*, but this first approximation is good enough for our purposes and we refer the reader to [Pin18] for details. This table is *not* supposed to be exhaustive and just presents the main classes we will encounter in the next chapter. The definitions of the logics as well as the references for the results are given in Sec. 1.4. A class of particular interest to us is the class $\mathbf{A}$ of *aperiodic* congruences.

A language is called a ***C**-language* (sometimes we abuse definitions and say that the language is *in* $\mathbf{C}$) if it is recognized by a $\mathbf{C}$-congruence. If $\mathbf{C}$ is a decidable class, since the syntactic congruence of a rational language L is computable and is coarser than all congruences recognizing L, and since classes are closed under taking coarser congruences, then one can decide if a language is in $\mathbf{C}$.

[4]In fact $\equiv_{\mathcal{A}}$ only *weakly* recognizes K, since $K = [a]\,[b]^\omega$ (see [CPP08]).

[5]In other words, $\mathbf{C}(A)$ is a meet-semilattice with order $\sqsubseteq$ and meet operation $\sqcap$. It is furthermore bounded and its greatest element is the trivial congruence: $\forall u, v \in A^*,\ u \sim v$.

Theorem 1.3.6. *Let $\boldsymbol{C}$ be a decidable class of congruences. Given a word automaton, one can decide if the language it recognizes is a $\boldsymbol{C}$-language.*

An automaton whose transition congruence is a $\mathbf{C}$-congruence is called a *$\boldsymbol{C}$-automaton.* Note that according to Prop. 1.3.4 and since classes of congruences are closed under coarser congruences, a language of finite words is a $\mathbf{C}$-language if and only if it is recognized by a $\mathbf{C}$-automaton. Concerning languages of infinite words, as we have seen in Ex. 1.3.5 the transition congruence of an automaton is not necessarily finer than the syntactic congruence of the language recognized by the automaton.

Example 1.3.7. Let us consider $\equiv$, the syntactic congruence of $L = aA^*b$ given in Ex. 1.3.5, and let us determine to which of the congruence varieties given above it belongs. First it is idempotent, we have indeed, $a \equiv aa$, $b \equiv bb$, $ab \equiv abab$, $ba \equiv baba$ and of course $\epsilon \equiv \epsilon \cdot \epsilon = \epsilon$. From this we have for any word u, that $u^\pi = u$, hence we can show that $\equiv$ is in $\mathbf{DA}$ and $\mathbf{A}$ (indeed we have $\mathbf{I} \subseteq \mathbf{DA} \subseteq \mathbf{A}$). However $\equiv$ is not in $\mathbf{J}$, since $b(ab)^\pi = bab \not\equiv (ab)^\pi = ab$.

If we now consider the syntactic congruence of $K = a(a^*b)^\omega$ given in Ex. 1.3.5, we can show that it is also in $\mathbf{I}$, $\mathbf{DA}$ and thus $\mathbf{A}$ but again not in $\mathbf{J}$.

1.3.3 Aperiodicity

Aperiodic and counter-free automata In the particular case of $\mathbf{A}$ we do have that considering the transition congruence of an ω-automaton is enough to decide if a language is aperiodic:

Theorem 1.3.8 ([DG08])**.** *A language of infinite words is aperiodic if and only it is recognized by an aperiodic Büchi automaton.*

A different characterization of aperiodic languages relies on the notion of *counter-free* automata. A word automaton $\mathcal{A}$ is counter-free if for any state p, any finite word u, and any positive integer n we have $p \xrightarrow{u^n} p \Rightarrow p \xrightarrow{u} p$. There is a close relationship between aperiodic automata and counter-free automata. First, one can easily show that a counter-free automaton is necessarily aperiodic. The converse does not hold in general, unless the automaton is not *too ambiguous.* An automaton is *polynomially ambiguous* if the number of runs of the automaton over a finite word is bounded by a polynomial in the size of the word.

Proposition 1.3.9. *If an automaton is aperiodic and polynomially ambiguous then it is counter-free.*

Proof. Let $\mathcal{A}$ be an aperiodic and polynomially ambiguous automaton with state space Q. Let p be a state of $\mathcal{A}$, and let n be a positive integer larger than π, the idempotent power of $\equiv_\mathcal{A}$. Let us assume that $p \xrightarrow{u^n} p$, we want to show that $p \xrightarrow{u} p$. By aperiodicity we know that $p \xrightarrow{u^{n+1}} p$. If we consider the runs $p \xrightarrow{u^n} p \dots p \xrightarrow{u^n} p$ and $p \xrightarrow{u^{n+1}} p \dots p \xrightarrow{u^{n+1}} p$ over $u^{n(n+1)}$ and assume that they are different then we have a contradiction with the polynomial ambiguity of $\mathcal{A}$. Indeed, the word $u^{(n(n+1))k}$ has at least 2^k different runs from p to p. This means that the two previous runs are identical and we have in particular that $p \xrightarrow{u^n} p \xrightarrow{u} p$, which concludes the proof. □

Co-example 1.3.10. The automaton of Fig. 1.3 on the facing page has counters yet it is aperiodic. However, it is exponentially ambiguous.

The following theorem characterizing languages recognized by counter-free automata is due to [MP71] for finite words and to [DG08] for infinite words:

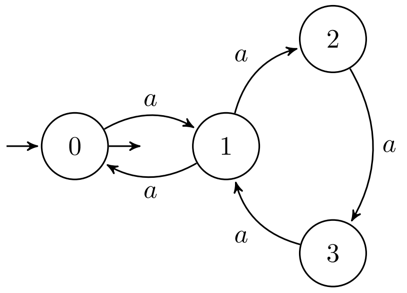


Figure 1.3: Aperiodic automaton with counters.

Theorem 1.3.11. *A language of finite (resp. infinite) words is aperiodic if and only if it is recognized by a counter-free automaton (resp. Büchi automaton).*

Aperiodic deterministic Muller automata also characterize aperiodic languages. In fact in [Tho81] the author shows that counter-free deterministic *Rabin* automata exactly characterize aperiodic languages. A proof of this result can also be found in [DKT16].

Theorem 1.3.12. *A language over infinite words is aperiodic if and only if it is recognized by an aperiodic Muller automaton.*

Similarly, we can show that the construction from [CM03] preserves aperiodicity.

Theorem 1.3.13. *Given an aperiodic congruence recognizing a language over infinite words L, one can obtain an aperiodic right Büchi automaton recognizing L in* EXPTIME.

Proof. Let us consider $\equiv$ an aperiodic congruence over A^*, and let $S := \{[u] \mid u \in A^+\}$ be the associated aperiodic semigroup. The construction from [CM03, Section 6.3.3] takes as input a semigroup and outputs a Büchi automaton. Our goal is to argue why this construction gives an aperiodic automaton, given an aperiodic semigroup. We use the same notations as in [CM03, Section 6.3], we only give the main arguments and we refer the reader to the original paper for more details on the construction. The states of the constructed automaton are of the form $([s,e], (s_1, \dots, s_n))$ where $[s,e]$ denotes a conjugacy class of linked pairs and $(s_1, \dots, s_n)$ is a chain of R-classes. Since S is aperiodic, the left action on conjugacy classes is obviously aperiodic. More precisely, if we have $[u^n] = \left[u^{n+1}\right]$, then in particular $[[u^n]\, s, e] = \left[\left[u^{n+1}\right] s, e\right]$. Similarly, one can see that there is an integer n large enough such that for any word u, $\rho(u^{n+1}) = \rho(u^n)$, and thus $\widehat{\varphi}(u^{n+1}w) = \widehat{\varphi}(u^n w)$.

Finally, the construction gives a Büchi automaton with a set of final *transitions* instead of final states. However one can easily go from a transition Büchi automaton to a Büchi automaton, while preserving aperiodicity, just by remembering in the states if the last transition seen is final or not. □

It was shown in [CH91] that deciding aperiodicity of language given as a deterministic automaton is PSPACE-hard. Since it is known that one can decide aperiodicity of a language in PSPACE, then the problem is PSPACE-complete, whether the input is a deterministic or non-deterministic automaton (see [DG08]).

Theorem 1.3.14. *The problem of deciding if a word automaton recognizes an aperiodic language is* PSPACE-*complete.*

1.4 Logics over words

In this section we present Monadic Second-Order (MSO) logic over words, a high-level formalism which is closer to natural language than say automata or congruences, and thus makes for a good specification language.

1.4.1 Monadic second-order logic

We give generic, although succinct, definitions of signatures, MSO-formulas, logical structures, *etc* and refer the reader to [EF95] for more details.

Signatures and logical structures A *signature* is a triple $\mathfrak{S} = (P, F, \mathsf{ar})$ with P being the set of *predicate symbols*, F the set of *function symbols* and $\mathsf{ar} : P \uplus F \to \mathbb{N}$ the *arity function.* A nullary function symbol is called a *constant symbol.* A *structure* M over $\mathfrak{S}$ is given as a set $\mathrm{dom}(M)$ called the *domain* of M, an *interpretation* for each predicate symbol p given by $p^M \subseteq \mathrm{dom}(M)^{\mathsf{ar}(p)}$ and for each function symbol f given by a *total* function $f^M : \mathrm{dom}(M)^{\mathsf{ar}(f)} \to \mathrm{dom}(M)$.

Monadic second-order formulas Monadic Second-Order formulas over a signature $\mathfrak{S}$, denoted by $\mathsf{MSO}[\mathfrak{S}]$, is the set of formulas given by the following grammar:

$$\phi ::= \exists X\ \phi \mid \exists x\ \phi \mid (\phi \wedge \phi) \mid \neg\phi \mid X(t) \mid p(t_1, \ldots, t_n) \mid \top$$

where X ranges over a countable set of *second-order variables* (often denoted by $X, Y, Z \ldots$), x ranges over a countable set of *first-order variables* (often denoted by $x, y, z \ldots$), p is a predicate of $\mathfrak{S}$ of arity n, $t_1, \ldots, t_n$ are terms build from function symbols and first-order variables and $\top$ is the true formula. We define universal quantifications and other boolean connectives as usual, $\forall X\ \phi := \neg\exists X\ \neg\phi$, $\forall x\ \phi := \neg\exists x\ \neg\phi$, $(\phi_1 \vee \phi_2) := \neg(\neg\phi_1 \wedge \neg\phi_2)$, $(\phi_1 \to \phi_2) := (\neg\phi_1 \vee \phi_2)$, $\bot := \neg\top$. Instead of $\exists x_1 \ldots \exists x_n$, we write $\exists x_1, \ldots, x_n$, and we do the same for universal quantification and second-order quantification.

Models A *free variable* is defined as usual as not being in the scope of a quantifier, and given a formula ϕ, we will often write $\phi(x_1, \ldots, x_m, X_1, \ldots, X_n)$ to denote that all free first-order variables of ϕ belong to $\{x_1, \ldots, x_m\}$ and free second-order variables to $\{X_1, \ldots, X_m\}$. A *sentence* is a formula without free variables. Let $\mathfrak{S}$ be a signature, M be a structure over $\mathfrak{S}$, let $\phi(x_1, \ldots, x_m, X_1, \ldots, X_n)$ be an $\mathsf{MSO}[\mathfrak{S}]$-formula and let $s_1, \ldots, s_m \in \mathrm{dom}(M)$ and $S_1, \ldots, S_n \subseteq \mathrm{dom}(M)$. In the following we write $\overline{x} := x_1, \ldots, x_m$ and $\overline{X} := X_1, \ldots, X_n$ and similarly for $\overline{s}$ and $\overline{S}$. We define a new signature $\mathfrak{S}_{\overline{x},\overline{X}}$ where each variable of $\overline{x}$ is a new constant symbols and each variable of $\overline{X}$ is a new unary predicate symbol. We can thus define a structure over $\mathfrak{S}_{\overline{x},\overline{X}}$, denoted by $M, \overline{s}, \overline{S}$ with the interpretation $x_i^M := s_i$ and $X_i^M := S_i$ for each *free* occurrence of the variables x_i of $\overline{x}$ and X_i of $\overline{X}$. We define by induction on formulas what it means for $M, \overline{s}, \overline{S}$ to *satisfy* ϕ (or to be a *model* of ϕ), which we denote by $M, \overline{s}, \overline{S} \models \phi$. If ϕ is equal to the true formula $\top$ then any structure is a model of ϕ, and $M, \overline{s}, \overline{S} \models \phi$. If $\phi = p(t_1, \ldots, t_k)$ with first-order variables of the terms of $\overline{t}$ all belonging to variables of $\overline{x}$, then the interpretations of the variables and the function symbols extend to the terms of $\overline{t}$, and we set that $M, \overline{s}, \overline{S} \models \phi$ if and only if $\overline{t}^{M,\overline{s}} \in p^M$. If $\phi = X_i(t)$, for some X_i of $\overline{X}$ and with the variables of t all belonging to $\overline{x}$, then we set that $M, \overline{s}, \overline{S} \models \phi$ if and only if $t^{M,\overline{s}} \in S_i$. If $\phi = \neg\psi$ then we set $M, \overline{s}, \overline{S} \models \phi$ if and only if $M, \overline{s}, \overline{S} \not\models \psi$. If $\phi = (\phi_1 \wedge \phi_2)$ then we set $M, \overline{s}, \overline{S} \models \phi$ if and only if $M, \overline{s}, \overline{S} \models \phi_1$

and $M, \overline{s}, \overline{S} \models \phi_2$. If $\phi = \exists x\ \psi$, then we set that $M, \overline{s}, \overline{S} \models \phi$ if and only if there exists an element $s \in \mathrm{dom}(M)$ such that $M, \overline{s}, s, \overline{S} \models \psi$, where free occurrences of x are interpreted as s. If $\phi = \exists X\ \psi$, then we set that $M, \overline{s}, \overline{S} \models \phi$ if and only if there exists a subset $S \subseteq \mathrm{dom}(M)$ such that $M, \overline{s}, \overline{S}, S \models \psi$, where free occurrences of X are interpreted as S.

1.4.2 MSO for words

We now focus on MSO restricted to structures which are words.

Words as logical structures A word w over an alphabet A is seen as a logical structure $\widetilde{w}$ over the signature $\mathfrak{S}_A$ with unary predicates a for each $a \in A$, a binary predicate $\leq$, and no functions. The *domain* of $\widetilde{w}$ is the set of its positions, the unary predicate a is interpreted as the set of positions labeled by a and the binary predicate $\leq$ is interpreted as the linear order over positions of w. For the sake of simplicity we will often write w instead of $\widetilde{w}$. We rather write $\mathsf{MSO}[\leq]$ to denote formulas over $\mathfrak{S}_A$, the alphabet often being implicit.

Formulas over finite and infinite words Let ϕ be an $\mathsf{MSO}[\leq]$-sentence, we define the *finite words semantics of* ϕ by $[\![\phi]\!]_* := \{u \in A^* \mid \widetilde{u} \models \phi\}$. Similarly, the *infinite words semantics* of ϕ is defined by $[\![\phi]\!]_\omega := \{x \in A^\omega \mid \widetilde{x} \models \phi\}$. Note that we will often write $[\![\phi]\!]$, relying on the context, and we say that ϕ *defines*, or *recognizes* the language $[\![\phi]\!]$. We have the seminal theorem(s) characterizing $\mathsf{MSO}[\leq]$-definable languages, due to [Bü60, Elg61, Tra61] for the finite case and [Bü62, McN66] for the infinite case:

Theorem 1.4.1. *A language (resp. ω-language) L is rational if and only if it is defined by some $\mathsf{MSO}[\leq]$-formula.*

Fragments of MSO A *fragment* F of $\mathsf{MSO}[\mathfrak{S}]$ is a subset of $\mathsf{MSO}[\mathfrak{S}]$-formulas. Given a fragment F and a language L, of finite or infinite words, we say that L is F-*definable* (or is an F-language) if there exists an F-formula defining it. We say that F *characterizes* (or is *equivalent* to) a congruence class $\mathbf{C}$, if any language is an F-language if and only if it is a $\mathbf{C}$-language.

The fragment of *first-order* logic $\mathsf{FO}[\mathfrak{S}]$ is the set of formulas which do not use second-order variables. A famous theorem linking algebra and logics is due, in the finite case, to [Sch65] for the algebraic part and to [MP71] on the logic side. The result was extended to infinite words by [Per84] on the algebraic side and by [Lad77] and [Tho79] on the logic side. As a corollary, $\mathsf{FO}[\leq]$-definability of rational languages is decidable.

Theorem 1.4.2. *A language (resp. ω-language) L is aperiodic if and only if it is $\mathsf{FO}[\leq]$-definable. Furthermore, one can decide in* PSPACE *if a language given as a word automaton is $\mathsf{FO}[\leq]$-definable.*

First-order logic with two variables, $\mathsf{FO}^2[\mathfrak{S}]$, is the set of first-order formulas which only use (and possibly reuse) two first-order variables. The following theorem is due to [Sch76a], [PW95] and [TW98] in the case of finite words. It was extended to infinite words by [DK09].

Theorem 1.4.3. *A language (resp. ω-language) L is in **DA** if and only if it is $\mathsf{FO}^2[\leq]$-definable. As a consequence, one can decide if a language given as a word automaton is $\mathsf{FO}^2[\leq]$-definable.*

The existential fragment of first-order logic, denoted by $\Sigma_1[\mathfrak{S}]$, is the set of formulas of the form $\exists x_1, \ldots, x_n\ \phi$, for some positive integer n and ϕ a quantifier-free formula. The boolean closure of a fragment F is denoted BF. The following theorem is mainly due to a result of [Sim75] (see [DGK08]).

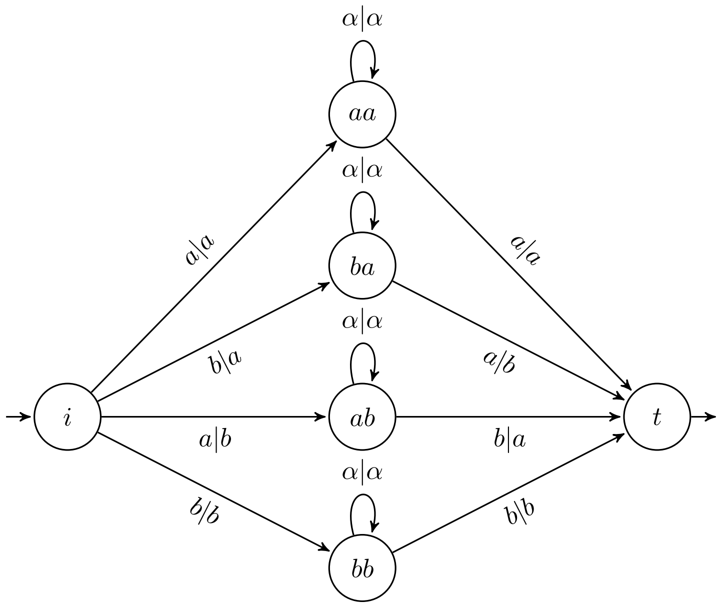


Figure 1.4: A transducer realizing f_{swap}.

Theorem 1.4.4. *A language L is in **J** if and only if it is $\mathsf{B}\Sigma_1[\leq]$-definable. As a consequence, one can decide if a language given as an automaton is $\mathsf{B}\Sigma_1[\leq]$-definable.*

Remark 1.4.5. The equivalence between $\mathsf{B}\Sigma_1[\leq]$ and **J** does not transfer to infinite words. An ω-language which is $\mathsf{B}\Sigma_1[\leq]$-definable is in particular in **J**, however the converse is not true. There is an equational characterization of $\mathsf{B}\Sigma_1[\leq]$-languages of infinite words but it relies on the notion of ω-semigroup which we choose not to introduce in this work. We refer the interested reader to [PP04].

Example 1.4.6. We define the minimum and maximum predicate by $\min(x) := \forall y\ x \leq y$ and $\max(x) := \forall y\ y \leq x$ which are definable in $\mathsf{FO}^2[\leq]$ (and thus in $\mathsf{FO}[\leq]$). The language $L = aA^*b$ is thus definable by the $\mathsf{FO}^2[\leq]$-formula: $\forall x\ (\min(x) \to a(x)) \wedge (\max(x) \to b(x))$. The language $K = a(a^*b)^\omega$ is definable by the $\mathsf{FO}^2[\leq]$-formula: $\forall x\ (\min(x) \to a(x)) \wedge (\exists y\ x < y \wedge b(y))$. Since we have seen in Ex. 1.3.7 that the syntactic congruence of L is not in **J**, we know from Th. 1.4.4 that L *cannot* be defined by a $\mathsf{B}\Sigma_1[\leq]$-formula.

1.5 Onward to transductions

Rational transductions have also been characterized by different models: functional transducers, (order-preserving) logical transducers and rational expressions, for instance. What is lacking is a good understanding of the links between these characterizations, and in particular a syntactic object which fully characterizes transducers. Such a syntactic object has been defined in the particular case of sequential transducers (see [Cho03]), and a first attempt for the general case of rational functions is given in [RS91]. In Chap. 2 and 3 we heavily rely on these two papers in our attempt to algebraically characterize rational functions.

1.5.1 Transductions and transducers

Transducers are automata which read pairs of words and thus recognize *relations* instead of languages. We refer the reader to [Ber79] for a detailed study of rational transductions.

Transducers A *transduction* (resp. ω-transduction) over alphabets A, B is a relation $R \subseteq A^* \times B^*$ (resp. $R \subseteq A^\omega \times B^\infty$). A *transducer*[6] (resp Büchi, Muller transducer) over alphabets A, B is a tuple $\mathcal{T} = (\mathcal{A}, i, o, t)$ (resp. $\mathcal{T} = (\mathcal{A}, i, o)$), with $\mathcal{A} = (Q, \Delta, I, F)$ an automaton (resp. Büchi, Muller automaton) over A called *the underlying automaton* of $\mathcal{T}$, $i : Q \to B^*$ the *initial function*, $o : \Delta \to B^*$ the *output function* and $t : Q \to B^*$ the *final function.* Let $u \in A^*$, let r be a run of $\mathcal{A}$ over u and let $\alpha := \prod_{1 \le i < |r|} o(r(i), u(i), r(i+1))$. We denote this by $r(1) \xrightarrow{u|\alpha}_{\mathcal{T}} r(|r|)$ if u is finite, and by $r(1) \xrightarrow{u|\alpha}_{\mathcal{T}} \mathrm{Inf}(|r|)$ otherwise. Furthermore, if r is accepting, let $\beta := i(r(1)) \cdot \alpha \cdot t(r(|r|))$ if r is finite and $\beta := i(r(1)) \cdot \alpha$ otherwise, then we say that the pair (u, β) is *realized* by $\mathcal{T}$ and we let $[\![\mathcal{T}]\!]$ denote the *transduction realized* by $\mathcal{T}$, that is the set of pairs realized by $\mathcal{T}$. We use the term ω-transducer to refer to a Büchi or Muller transducer. A transduction (resp. ω-transduction) is called *rational* if it is realized by a transducer (resp. ω-transducer).

Subclasses of transducers A transducer is *letter-to-letter* if for any transition $t \in \Delta$, its output $o(t)$ is a single letter of A, and the initial and final output functions are constant equal to ϵ. By extension a transduction realized by a letter-to-letter transducer is called *letter-to-letter.* A transduction f which is a partial function is called *functional* and we rather denote $(u, \alpha) \in f$ by $f(u) = \alpha$. All the transductions we consider in Chap. 2 and 3 are functional. It was shown in [Sch76b] that functionality of transducers over finite words is decidable. The result has been improved, both in the case of finite words (see [GI83, BCPS00]) and infinite words (see [Gir86, Pri02]), and functionality of transducers is actually decidable in PTime.

Theorem 1.5.1. *Functionality of transducers, over finite or infinite words, can be decided in* PTime.

A transducer is called *unambiguous* when its underlying automaton is unambiguous and in that case the transduction it realizes is necessarily functional. Moreover it has been shown in [Eil74] that any functional transduction over finite words can be realized by some unambiguous transducer. It is also the case over infinite words and this can be seen as a consequence of [Car10, Theorem 1]. A transducer is called *sequential*[7] if its underlying automaton is deterministic and we extend the terminology to the function it realizes. Sequentiality of a transduction is decidable, as it was shown in [Cho77] for finite words, and even in PTime, whether on finite words [WK94, BCPS00] or on infinite words [BC04].

Theorem 1.5.2. *Sequentiality of rational transductions, over finite or infinite words, can be decided in* PTime.

A *right-sequential* transducer over finite words is a transducer with a co-deterministic underlying automaton. For infinite words [Car10] proposed a definition of *right-sequential* transducers over infinite words based on so-called *prophetic* automata, *i.e.* co-deterministic and co-complete automata (see [CM03]). Intuitively a right-sequential transducer is deterministic but reads its input from right to left. We mentioned above than any rational function can be realized by an

[6]In the literature these are sometimes called *real-time* transducers, *i.e.* transducers without ϵ-transitions.
[7]These are sometimes called subsequential in the literature, to denote that final outputs are allowed.

unambiguous transducer, but a stronger result from [EM65] states that any rational function can be obtained as the composition (in both ways) of a sequential transducer with a right-sequential transducer. In [Car10] a similar result (but only in one way) was obtained for transducers over infinite words.

Theorem 1.5.3. *Let f be a rational function over finite words, then there exist g sequential (resp. right-sequential) and h letter-to-letter and right-sequential (resp. sequential) such that $f = g \circ h$.*

Let f be a rational function over infinite words, then there exist g sequential and h letter-to-letter and right-sequential such that $f = g \circ h$.

C-transducers The *transition congruence* of a transducer is the transition congruence of its underlying automaton. A transducer over finite words with a transition congruence in **C** is called a ***C**-transducer*[8]. A transduction over finite words is ***C**-rational* (resp. ***C**-sequential*) if is is realized by some functional (resp. sequential) **C**-transducer. Note that **C**-rational transductions are sometimes called **C**-transductions for short. Over infinite words, as we have seen in Ex. 1.3.7, considering only the transition congruence of an automaton is not justified in general to algebraically characterize a language. However according to Th. 1.3.8 it is justified for **A**, and we will see in Chap. 3 that it is also justified in the case of transducers. A transducer over infinite words is called *aperiodic* if its underlying automaton is aperiodic. An ω-transduction is called *aperiodic* (resp. *aperiodic-sequential*) if it is realized by an aperiodic transducer (resp. aperiodic and sequential transducer).

Example 1.5.4. Let $A := \{a, b\}$, we define the (partial) function $f_{\text{swap}} : A^* \to A^*$, for any $\alpha, \beta \in A, u \in A^*$ by $f(\alpha u \beta) := \beta u \alpha$. Figure 1.4 represents a transducer $\mathcal{T}$ realizing f_{swap}, where α denotes any letter of A.

We can compute $\equiv$ the transition congruence of $\mathcal{T}$ and we obtain the following equivalence classes: $A^*/_{\equiv} = \{\epsilon, a, b, aA^*a, aA^*b, bA^*a, bA^*b\}$. One can show that $\equiv$ is in **DA** (and hence **A**) but not in **I** nor **J**. Hence we can conclude that f_{swap} is a **DA**-transduction, however we cannot say if it is an **I** (resp. **J**) transduction, because there might exist some **I** (resp. **J**) transducer realizing it. Actually, the domain of f_{swap} is not an **I**-language since a is not in it whereas aa is. Because any transducer realizing f_{swap} must recognize its domain we can conclude that f_{swap} is not an **I**-transduction. In order to decide if f_{swap} is a **J**-transduction, we will need some algebraic characterization of transductions, which we provide in Sec. 2.2.

1.5.2 Logical transducers

MSO-transducers are a formalism of graph transformations introduced by Courcelle (see [Cou94]). In [EH01] the authors have shown that word-to-word MSO-transducers exactly capture functional two-way transducers. Our definition of logical transducers is equivalent to the notion of *order-preserving* MSO-transducers from [Fil15], a restriction which coincides with one-way transducers, *i.e.* rational functions [Fil15, Theorem 4].

Pointed structures Let $\mathfrak{S}$ be a signature and let F be a logical fragment. As we have seen before, a formula with a free variable can actually be seen as a sentence over an extended signature. We define the *pointed* version of $\mathfrak{S}$ by $\mathfrak{S}_{\mathsf{c}}$, which is $\mathfrak{S}$ with an additional constant symbol c. From this we can easily define the fragment F_{c} by all the F-formulas where

[8]Note that in [RS95] **C**-transductions are defined by *unambiguous* **C**-transducer. The two definitions turn out to be equivalent (see Cor. 2.2.16).

c is substituted for arbitrary occurrences of first-order variables. The finite (resp. infinite) pointed word semantics of an $\mathsf{F_c}$-formula is given by $[\![\phi]\!]_* := \{(u,i) \in A^* \times \{1,\dots,|u|\} \mid \widetilde{u}, i \models \phi\}$ (resp. $[\![\phi]\!]_\omega := \{(x,i) \in A^\omega \times |\mathbf{x}| \mid \widetilde{x}, i \models \phi\}$). Again we will often rely on the context and simply write $[\![\phi]\!]$.

Remark 1.5.5. Pointed formulas can be intuitively seen as formulas with an extra variable which cannot be quantified upon. We could have chosen a different definition of $\mathsf{F_c}$, namely substitute c for *free* occurrences of first-order variables. In fragments where the number of variables is not bounded, these two definitions are equivalent. However in the case of $\mathsf{FO}^2_\mathsf{c}[\leq]$, this alternative definition is strictly less expressive than the one we chose, as it was noticed by [Boj15]. As an example, the $\mathsf{FO}^2_\mathsf{c}[\leq]$-sentence $\exists x\ a(x) \wedge (x < c) \wedge \forall y\ (x < y) \to (c \leq y)$ which states that the position just before the pointed position is labeled by a cannot be expressed in the formalism where only free occurrences of variables are replaced by c.

Example 1.5.6. Let us define the pointed $\mathsf{FO}^2_\mathsf{c}[\leq]$-sentence: $\phi_{a,\text{middle}} := a(\mathsf{c}) \wedge \neg\min(\mathsf{c}) \wedge \neg\max(\mathsf{c})$, which states that the pointed position is labeled by a and is neither minimal nor maximal.

Logical transducers Let F be a fragment of $\mathsf{MSO}[\leq]$, an F-*transducer* over an alphabet A is a tuple $\mathcal{T} = \left(K, \phi_{\text{dom}}, (\phi_\alpha)_{\alpha\in K}\right)$ where K is a finite subset of A^*, ϕ_{dom} is an F-sentence, called the *domain formula*, and for each $\alpha \in K$, ϕ_α is an $\mathsf{F_c}$-sentence. Let u be a word, finite or infinite, satisfying ϕ_{dom}, and for each integer $1 \leq i \leq |u|$ let α_i be such that $(u,i) \models \phi_{\alpha_i}$, if it exists. In that case we set $\alpha := \prod_i \alpha_i$, and we say that $\mathcal{T}$ *realizes* the pair (u,α). The *finite words* (resp. *infinite words*) *semantics* of $\mathcal{T}$ is denoted by $[\![\mathcal{T}]\!]_* := \{(u,\alpha) \mid u \in A^*,\ \mathcal{T} \text{ realizes } (u,\alpha)\}$ (resp. $[\![\mathcal{T}]\!]_\omega := \{(x,\alpha) \mid x \in A^\omega,\ \mathcal{T} \text{ realizes } (x,\alpha)\}$). Just as before we will rely on the context and often write $[\![\mathcal{T}]\!]$. Note that this relation is not necessarily functional but in the following, we always assume that it is and that its domain is equal to $[\![\phi_{\text{dom}}]\!]$.

Example 1.5.7. We define the $\mathsf{FO}^2[\leq]$-transducer $\mathcal{T}_{\text{swap}} := (\{a,b\}, \phi_{\text{dom}}, (\phi_a, \phi_b))$ realizing the function of Fig. 1.4 on page 22.

- $\phi_{\text{dom}} := \exists x, y\ x < y$
- $\phi_a := (\min(\mathsf{c}) \wedge \exists x\ \ \max(x) \wedge a(x)) \vee (\max(\mathsf{c}) \wedge \exists x\ \ \min(x) \wedge a(x)) \vee \phi_{a,\text{middle}}$
- $\phi_b := (\min(\mathsf{c}) \wedge \exists x\ \ \max(x) \wedge b(x)) \vee (\max(\mathsf{c}) \wedge \exists x\ \ \min(x) \wedge b(x)) \vee \phi_{b,\text{middle}}$

where $\phi_{a,\text{middle}}$ is the formula given in Ex. 1.5.6

As mentioned above, the following result extends [Fil15, Theorem 4], where the author uses a slightly different but equivalent formalism for logical transducers. This result easily transfers to ω-transducers. It also generalizes to the equivalence between aperiodic transducers and $\mathsf{FO}[\leq]$-transducers, both over finite and infinite words. It is the natural extension of the result of Büchi from rational languages to rational functions.

Theorem 1.5.8. *A functional transduction (resp. ω-transduction) is realizable by an unambiguous transducer if and only if it is realizable by an* $\mathsf{MSO}[\leq]$*-transducer.*

Proof. We give the proof of the equivalence between ω-transducers and $\mathsf{MSO}[\leq]$-transducers over infinite words.

Let $\mathcal{T} = (\mathcal{A}, i, o)$ be an unambiguous transducer realizing a function $f : A^\omega \to B^\infty$, with underlying automaton $\mathcal{A} = (Q, \Delta, I, F)$. Let p, q be states of $\mathcal{A}$, let $L_{p,q} \subseteq A^*$ be the set of finite words which can go from p to q and let $L_q \subseteq A^\omega$ denote the set of infinite words having a final run from q. Since $L_{p,q}, L_q$ are rational, there are $\mathsf{MSO}[\leq]$-formulas $\phi_{p,q}, L_q$ recognizing them, respectively. Given an $\mathsf{MSO}[\leq]$-formula ϕ we define inductively the $\mathsf{MSO}_\mathsf{c}[\leq]$-formula $\phi^{<\mathsf{c}}$ by restricting the quantifications to the positions before c. Formally, if $\phi = \exists y\ \psi(y)$ then we set $\phi^{<\mathsf{c}} := \exists y\ (y < \mathsf{c}) \wedge \psi^{<\mathsf{c}}(y)$. Atomic formulas and boolean connectives are left unchanged.

Let us define $\mathcal{T}' = \big(K, \phi_{\mathrm{dom}}, (\phi_v)_{v\in K}\big)$ by:

- $K := (\{\epsilon\} \cup i(I)) \cdot o(\Delta)$
- $\phi_{\mathrm{dom}} := \bigvee_{p,q \in I\times F} \phi_{p,q}$
- $\phi_v(x) := \left[\begin{array}{l} \bigvee_{i(p_0)o(p_0,a,q)=v} \min(\mathsf{c}) \wedge a(\mathsf{c}) \wedge \phi_q^{>\mathsf{c}} \\ \bigvee_{o(p,a,q)=v, p_0\in I} \neg \min(\mathsf{c}) \wedge \phi_{p_0,p}^{<\mathsf{c}} \wedge a(\mathsf{c}) \wedge \phi_q^{>\mathsf{c}} \end{array} \right.$

By construction, and since $\mathcal{A}$ is unambiguous, we have that $\mathcal{T}'$ realizes f.

Conversely, let $\mathcal{T} = \big(K, \phi_{\mathrm{dom}}, (\phi_v)_{v\in K}\big)$ be a logical transducer realizing a function $f : A^\omega \to B^\infty$. From $\mathcal{T}$, we define an ω-language over the alphabet $A \times K$. Let ϕ be an $\mathsf{MSO}_\mathsf{c}[\leq]$-formula, we syntactically define $\phi'(c)$ an $\mathsf{MSO}[\leq]$-formula with one free variable. Each label predicate $a(x)$ is replaced syntactically by $\bigvee_{v\in K}(a,v)(x)$. Then each occurrence of the constant symbol c is replaced by a fresh free variable symbol c. For $v \in K$, we define a predicate $v(x) := \bigvee_{a\in A}(a,v)(x)$. We define the formula:

$$\phi_\mathcal{T} := \phi'_{dom} \wedge \forall c \bigvee_{v\in K} v(c) \wedge \phi'_v(c)$$

The formula $\phi_\mathcal{T}$ recognizes words such that the projection over A^ω is in $\mathrm{dom}(f)$. Furthermore each position is labeled by its output. We define a trim automaton $\mathcal{A}_\mathcal{T} = (Q_\mathcal{T}, \Delta_\mathcal{T}, I, F)$ recognizing the same language as $\phi_\mathcal{T}$. From this we naturally define a transducer $\mathcal{T}' := (\mathcal{A}, i, o)$ with $\mathcal{A} := (Q_\mathcal{T}, \Delta, I, F)$ by:

- $\Delta := \{(p,a,q) \mid\ (p,(a,v),q) \in \Delta_\mathcal{T}\}$
- $i : p \mapsto \epsilon$
- $o(p,a,q) := v$ such that $(p,(a,v),q) \in \Delta_\mathcal{T}$

Note that o is well-defined since f is a function. By construction $\mathcal{T}'$ realizes f. $\square$

Chapter 2

Characterizations of rational functions over finite words

"I love deadlines. I love the whooshing noise they make as they go by."

– Douglas Adams, *The Salmon of Doubt*

The existence of a syntactic object capturing all the algebraic properties of a language is a cornerstone of the study of rational languages. The many correspondences between algebraic varieties and logical fragments, together with the effective computation of the syntactic congruence, has yielded a generic tool to effectively decide if a language is expressible in some logical fragment.

The goal of this chapter is to try to obtain such an object for rational functions and see which logic-algebra correspondences can be carried over to functions. In the case of sequential functions, this has already been achieved: for any sequential function, one can compute a sequential transducer which is minimal in the strong sense that its right transition congruence is the coarsest among those of sequential transducers realizing the function (see [Cho03]). In the case of rational functions the result from [RS91] gives the construction of a canonical, *i.e.* not machine dependent, transducer realizing a function. However this canonical machine does not have the nice minimality properties which provide an algebraic characterization in the case of languages.

Refining the techniques of [RS91], we are able to show the existence, and give the constructions, of a *finite* number of minimal machines for any rational function. These machines are minimal in the sense that the transition congruence of any transducer realizing the function has to be finer than the transition congruence of one of the minimal machines. This gives us a procedure, for any decidable class of congruences **C**, to decide if a transduction is in **C**. On the logical side, we provide a transfer result which gives sufficient conditions on a logical fragment so that an equivalence with a class of congruences can be carried over to transductions. These conditions basically amount to having access to the linear order predicate. In these conditions we are thus able to decide if a transduction is definable in a given logical fragment.

We start this chapter by studying the case of sequential functions, reformulating the results of [Cho03] in our framework. We also show that the class of aperiodic transducers is *robust* to determinization, which means that a transduction is **A**-sequential if and only if it is sequential and in **A**, which is not true for an arbitrary class of congruences **C**. Then we define bimachines, a model of computation for rational functions introduced in [Sch61], and named as such and further studied by [Eil74]. A bimachine, as its name suggests, is given by two automata, one deterministic and one co-deterministic. Intuitively it can be seen as a sequential transducer with look-ahead where the look-ahead is given by a co-deterministic automaton. Of course the model is completely symmetrical and could be seen as a right-sequential transducer with a look-behind. We are able to minimize bimachines using a two-sided version of minimization *à la* Choffrut, and then we show that the number of minimal bimachines is always finite. In the particular case of aperiodic transductions, we show that the robustness of the class means that if a transduction is aperiodic then *all* its minimal bimachines are. This property provides a PSpace algorithm to decide aperiodicity of a transduction. Finally we consider the logical side of rational transductions and establish some logic-algebra correspondences.

The results of this chapter can be found in [FGL16a, FGL16b]. Throughout this chapter, we only consider finite words.

2.1 Sequential functions

Sequential transductions can be characterized by a right syntactic congruence which yields, as for automata, a unique (up to shifting outputs) minimal machine realizing the function (see [Cho03]). By the closure properties of classes of congruences, deciding **C**-sequentiality thus amounts to computing the minimal sequential transducer and deciding if its transition congruence is in **C**. In the case of a decidable class, this gives a decision procedure for **C**-sequentiality.

We describe this right syntactic congruence as well as a minimization procedure in PTime, which can both be found in [Cho03] but which we will generalize later in the case of rational functions. We then describe the determinization procedure of [BC02], which terminates for sequential functions, and show that it preserves aperiodicity of transducers. This means that deciding aperiodicity of a sequential function can be reduced to deciding **A**-sequentiality. In particular, as we will see in Sec. 2.3 this gives a procedure to decide if a sequential function is $\mathsf{FO}[\leq]$-definable.

2.1.1 Algebraic characterization of sequential functions

Here we describe the right syntactic congruence of a function and the minimal sequential transducer associated with it. This is very analogous to the Myhill-Nerode congruence of a language and the corresponding minimal deterministic automaton.

Infimum of a function Let A, B be alphabets and let $f : A^* \to B^*$ be a (partial) function. We define the *infimum* of f, $\widehat{f} : A^* \to B^*$ by $\widehat{f}(u) := \bigwedge\{f(v) \mid \ u \leq v,\ v \in \mathrm{dom}(f)\}$, which is defined over $\downarrow\!\mathrm{dom}(f)$. In other words $\widehat{f}(u)$ outputs the longest possible word that f can produce by reading a word starting with u.

Example 2.1.1. Consider the function f_{swap} given by the transducer of Fig. 1.4 on page 22. Then we have $\widehat{f_{\mathsf{swap}}}(u) = \epsilon$ for any word $u \in \{a, b\}^*$. Let us also consider the function f_{idem} given by the transducer of Fig. 2.1 on the next page. We have that $f_{\mathsf{idem}}(u) = \epsilon$ if $u \in \{\epsilon, a\}$ and $f_{\mathsf{idem}}(u) = a$ if $u \in aa^+$. One can easily see that $\widehat{f_{\mathsf{idem}}} = f_{\mathsf{idem}}$.

Right syntactic congruence

Definition 2.1.2. The *right syntactic congruence* of a function $f : A^* \to B^*$ is defined by $u \sim_f v$ if:

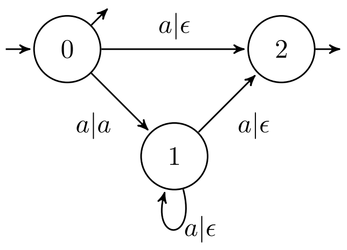


Figure 2.1: A transducer realizing f_{idem}.

- $u^{-1}\text{dom}(f) = v^{-1}\text{dom}(f)$
- for any $w \in u^{-1}\text{dom}(f)$, $\widehat{f}(u)^{-1}f(uw) = \widehat{f}(v)^{-1}f(vw)$

Proposition 2.1.3. *Let f be a function, then $\sim_f$ is a right congruence.*

Proof. Let $f : A \to B$ be a function, let $u \sim_f v$ and let $a \in A$. We want to show that $ua \sim_f va$. First $(ua)^{-1}\text{dom}(f) = a^{-1}(u^{-1}\text{dom}(f)) = (va)^{-1}\text{dom}(f)$. We know that for any word $w \in u^{-1}\text{dom}(f)$, $\widehat{f}(u)^{-1}f(uw) = \widehat{f}(v)^{-1}f(vw)$, and we let $g(w)$ denote this word. Let $aw \in u^{-1}\text{dom}(f)$.

$$\begin{array}{rcl}
\widehat{f}(ua)^{-1}f(uaw) & = & \left(\bigwedge_z f(uaz)\right)^{-1} f(uaw) \\
& = & \left(\bigwedge_z \widehat{f}(u)g(az)\right)^{-1} \widehat{f}(u)g(aw) \\
& = & \left(\widehat{f}(u)\bigwedge_z g(az)\right)^{-1} \widehat{f}(u)g(aw) \\
& = & \left(\bigwedge_z g(az)\right)^{-1} g(aw) \\
& = & \widehat{f}(va)^{-1}f(vaw)
\end{array}$$

Hence $ua \sim_f va$ which concludes the proof. □

Minimal sequential transducer The *minimal sequential transducer* of a transduction $f : A^* \to B^*$ is defined, using the right syntactic congruence, by $\mathcal{T}_f = (\mathcal{A}_f, i_f, o_f, t_f)$, with $\mathcal{A}_f = (Q_f, \Delta_f, I_f, F_f)$ where:

- $Q_f := \{[u] \mid u \in {\downarrow}\text{dom}(f)\}$
- $\Delta_f := \{([u], a, [ua]) \mid a \in A\}$
- $I_f := \{[\epsilon]\}$
- $F_f := \{[u] \mid u \in \text{dom}(f)\}$
- $i_f([\epsilon]) := \widehat{f}(\epsilon)$
- $o_f([u], a, [ua]) := \widehat{f}(u)^{-1}\widehat{f}(ua)$
- $t_f([u]) := \widehat{f}(u)^{-1}f(u)$

Intuitively, the minimal sequential transducer outputs over a word u the longest possible word, knowing that the input begins with u.

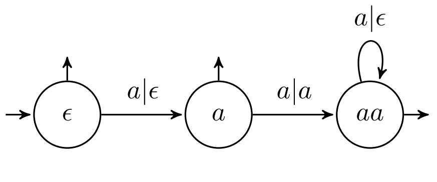


Figure 2.2: The minimal sequential transducer of f_{idem}.

Remark 2.1.4. Note that we have defined a transducer which has an infinite number of states, in general. Of course this definition applies to arbitrary functions, but will yield a finite machine when the function considered is a sequential function.

Notice that $\sim_{\mathcal{A}_f} = \sim_f$, by definition, and that the number of states of $\mathcal{A}_f$ is the number of equivalence classes of $\sim_f$ (up to adding a sink state).

Proposition 2.1.5. *The outputs of the minimal sequential transducer of a function are well-defined.*

Proof. We show that the output functions are well-defined. Let $f : A^* \to B^*$. The initial output function is obviously well-defined. Let $u \sim_f v$ and let $a \in A$ such that $ua \in {\downarrow}\text{dom}(f)$. We want to show that the output function o is well-defined, meaning that $\widehat{f}(u)^{-1}\widehat{f}(ua) = \widehat{f}(v)^{-1}\widehat{f}(va)$. As before we have that $\widehat{f}(u)^{-1}f(uw) = \widehat{f}(v)^{-1}f(vw)$, for any $w \in u^{-1}\text{dom}(f)$, and we denote this word by $g(w)$.

$$\begin{array}{rcl}\widehat{f}(u)^{-1}\widehat{f}(ua) & = & \left(\bigwedge_z f(uz)\right)^{-1} \bigwedge_z f(uaz) \\ & = & \left(\bigwedge_z \widehat{f}(u)g(z)\right)^{-1} \bigwedge_z \widehat{f}(u)g(az) \\ & = & \left(\bigwedge_z \widehat{g}(z)\right)^{-1} \bigwedge_z g(az) \\ & = & \widehat{f}(v)^{-1}\widehat{f}(va)\end{array}$$

The final output function is well-defined, since $\widehat{f}(u)^{-1}f(u) = \widehat{f}(v)^{-1}f(v)$ for $u, v \in \text{dom}(f)$, by definition of $\sim_f$. □

Example 2.1.6. Let us consider again the functions f_{swap} and f_{idem}, given by their respective transducers of Fig. 1.4 on page 22 and 2.1. As we have seen $\widehat{f_{\text{swap}}}$ is the constant function equal to ϵ. Hence, one can show that the right syntactic congruence of f_{swap} is the finest congruence, separating all words. The minimal sequential transducer of f_{swap} has an infinite number of states, and the final output function outputs $f_{\text{swap}}(u)$ for $u \in \text{dom}(f)$ since $\widehat{f_{\text{swap}}}$ is constant, equal to ϵ. This simply means that the function is *not* sequential.

The syntactic congruence of f_{idem} has three classes, namely ϵ, a and aa^+, and the minimal sequential transducer is given in Fig. 2.2.

Theorem 2.1.7. *The transducer $\mathcal{T}_f$ realizes f.*

Proof. Let $f : A^* \to B^*$ be a function, and let $u \in \text{dom}(f)$. By definition of the outputs of $\mathcal{T}_f$, $[\![\mathcal{T}_f]\!](u) = \widehat{f}(\epsilon) \cdot \left(\prod_{0 \leq i < |u|} \widehat{f}(u(:i))^{-1}\widehat{f}(u(:i+1))\right) \cdot \widehat{f}(u)^{-1}f(u) = f(u)$. □

We now give the theorem which justifies the name of *minimal* sequential transducer.

Theorem 2.1.8. *Let f be a sequential function realized by a sequential transducer with underlying automaton $\mathcal{A}$, then $\sim_{\mathcal{A}} \sqsubseteq \sim_f$.*

Proof. Let $\mathcal{T} = (\mathcal{A}, i, o, t)$ be a sequential transducer with $\mathcal{A} = (Q, \Delta, \{q_0\}, F)$ realizing f, let $u \sim_{\mathcal{A}} v$ and let p be the state of $\mathcal{A}$ reached from the initial state by u and v. Let $\mathcal{T}_p := (\mathcal{A}_p, p \mapsto \epsilon, o, t)$ with $\mathcal{A}_p := (Q, \Delta, \{p\}, F)$, and let $f_p := [\![\mathcal{T}_p]\!]$. We have $u^{-1}\mathrm{dom}(f) = v^{-1}\mathrm{dom}(f)$ since this set is the language of words which have a final run from p.

Let $w \in u^{-1}\mathrm{dom}(f)$, we want to show that $\widehat{f}(u)^{-1}f(uw) = \widehat{f}(v)^{-1}f(vw)$. Let us denote the initial runs of $\mathcal{T}$ over u and v by $q_0 \xrightarrow{u|\alpha} p$ and $q_0 \xrightarrow{v|\beta} p$, respectively. Let $\gamma := \bigwedge \{f_p(x) \mid x \in \mathrm{dom}(f_p)\}$, and let $p \xrightarrow{w|\delta} q_f$. Thus we have $\widehat{f}(u) = i(q_0)\alpha\gamma$, $\widehat{f}(v) = i(q_0)\beta\gamma$, $f(uw) = i(q_0)\alpha\delta t(q_f)$ and $f(vw) = i(q_0)\beta\delta t(q_f)$. Hence we obtain that $\widehat{f}(u)^{-1}f(uw) = \gamma^{-1}\delta t(q_f) = \widehat{f}(v)^{-1}f(vw)$. □

Corollary 2.1.9. *For any class of congruences* $\boldsymbol{C}$*, a transduction is* $\boldsymbol{C}$*-sequential if and only if its minimal sequential transducer is in* $\boldsymbol{C}$*.*

Proof. Let $\mathcal{T}$ be a sequential transducer with underlying automaton $\mathcal{A}$ realizing a function f, such that $\equiv_{\mathcal{A}} \in \mathbf{C}$. Let $\equiv$ be the transition congruence of $\mathcal{A}_f$, the underlying automaton of the minimal sequential transducer of f. According to Th. 2.1.8, we have that $\sim_{\mathcal{A}} \sqsubseteq \sim_f$. Let $u \equiv_{\mathcal{A}} v$, let $[w]_f$ be a state of $\mathcal{A}_f$, and let $[w]_f \xrightarrow{u}_{A_f} [wu]_f$ denote the run of A_f over u from $[w]_f$. Since $u \equiv_A v$, we have in particular that $wu \sim_{\mathcal{A}} wv$, and thus $wu \sim_f wv$. Hence we have a run $[w]_f \xrightarrow{v}_{A_f} [wu]_f$, and we obtain $\equiv_{\mathcal{A}} \sqsubseteq \equiv$. □

2.1.2 Minimization of sequential transducers

The minimization of sequential transducers is very similar to the minimization of deterministic automata. Here we describe a Moore-like minimization algorithm, which works by partitioning the states too coarsely and then refining the partition until a fixpoint is reached, which we show gives the syntactic congruence. The algorithm also relies on a preprocessing step to output words as early as possible [Cho03].

Let $\mathcal{T} = (\mathcal{A}, i, o, t)$ be a transducer, let p be a state and let the longest common prefix of outputs from p be $\alpha_p := \bigwedge \left\{\alpha \mid p \xrightarrow{u|\beta} q,\ \alpha = \beta t(q)\right\}$. Then $\mathcal{T}$ is called in *earliest form* if for any state p, $\alpha_p = \epsilon$.

Proposition 2.1.10. *Given a transducer* $\mathcal{T}$*, one can obtain in* PTime *a new transducer* $\widehat{\mathcal{T}}$ *in earliest form realizing the same function.*

Proof. We give the definition of $\widehat{\mathcal{T}}$, show that it realizes the same function and is indeed in earliest form, and finally we give a procedure to compute it. Let $\mathcal{T} = (\mathcal{A}, i, o, t)$ be a transducer realizing a function f, let p be a state of $\mathcal{A}$ and let α_p denote the longest common prefix of outputs from p. We define $\widehat{\mathcal{T}} = \left(\mathcal{A}, \widehat{i}, \widehat{o}, \widehat{t}\right)$ by:

- $\widehat{i}(p) := i(p)\alpha_p$
- $\widehat{o}(p, a, q) := \alpha_p^{-1} o(p, a, q)\alpha_q$
- $\widehat{t}(p) := \alpha_p^{-1} t(p)$

One can easily see that all the α_ps cancel out, and thus $\widehat{\mathcal{T}}$ realizes f as well.

Let $p \xrightarrow{u|\beta}_{\mathcal{T}} q$ and $p \xrightarrow{u|\beta'}_{\widehat{\mathcal{T}}} q$, we have that $\beta'\widehat{t}(q) = \alpha_p^{-1}\beta t(q)$, which means that the longest common prefix of outputs from p for $\widehat{\mathcal{T}}$ is equal to $\alpha_p^{-1}\alpha_p = \epsilon$.

To show that computing the α_ps can be done in PTime, we refer the reader to [Cho03, Section 5] which surveys different approaches as well as their precise complexities. $\square$

We now give the minimization theorem, which can be found in [Cho03].

Theorem 2.1.11. *Given a sequential transducer, one can compute (up to renaming states) the associated minimal sequential transducer in* PTime.

Proof. Let $\mathcal{T} = (\mathcal{A}, i, o, t)$, with $\mathcal{A} = (Q, \Delta, I, F)$ be a sequential transducer realizing a function f. According to Prop. 2.1.10 we can assume without loss of generality that $\mathcal{T}$ is in earliest form.

Now the idea is to compute an equivalence relation by successive refinements. The idea is very similar to Moore's deterministic automata minimization algorithm [Moo56], except that the initial relation also has to be compatible with the output functions.

In the following we identify the states of $\mathcal{A}$ with the equivalence classes of $\sim_{\mathcal{A}}$. The initial partition is given for all words u, v by $u \sim_0 v$ if: 1) $[u]_{\mathcal{A}} \in F \Leftrightarrow [v]_{\mathcal{A}} \in F$ and $t([u]_{\mathcal{A}}) = t([v]_{\mathcal{A}})$ and 2) $\forall a \in A$, $o([u]_{\mathcal{A}}, a, [ua]_{\mathcal{A}}) = o([v]_{\mathcal{A}}, a, [va]_{\mathcal{A}})$.

Let $i \in \mathbb{N}$, we define inductively for all words u, v, $u \sim_{i+1} v$ by: $u \sim_i v$ and $\forall a \in A$, $ua \sim_i va$. Of course we have $\forall i \in \mathbb{N}$ that $\sim_{\mathcal{A}} \sqsubseteq \sim_{i+1} \sqsubseteq \sim_i$, which means that we reach a fixpoint relation for some i, which we denote by $\sim_*$.

Let us first show that $\sim_*$ is fine enough to realize f, which means that $\sim_* \sqsubseteq \sim_f$ according to Th. 2.1.8. The relation $\sim_*$ is a right congruence since it is a fixpoint for right multiplication. Let $\mathcal{T}_* = (\mathcal{A}_*, i_*, o_*, t_*)$ with $\mathcal{A}_* = (Q_*, \Delta_*, I_*, F_*)$ be defined as:

- $Q_* := \{[u]_* \mid u \in {\downarrow}\mathrm{dom}(f)\}$
- $\Delta_* := \{([u]_*, a, [ua]_*) \mid a \in A\}$
- $I_* := \{[\epsilon]_*\}$
- $F_* := \{[u]_* \mid [u]_{\mathcal{A}} \in F\}$
- $i_*([\epsilon]_*) := i([\epsilon]_{\mathcal{A}})$
- $o_*([u]_*, a, [ua]_*) := o([u]_{\mathcal{A}}, a, [ua]_{\mathcal{A}})$
- $t_*([u]_*) := t([u]_{\mathcal{A}})$

The final states and the final outputs are well defined according to point 1) in the definition of $\sim_0$ given above. The output function is well-defined according to point 2). Given a word in the domain of f, its outputs for $\mathcal{T}_*$ will be exactly the same as for $\mathcal{T}$ which means that $\mathcal{T}_*$ realizes f and hence $\sim_* \sqsubseteq \sim_f$.

We only have left to show that $\sim_f \sqsubseteq \sim_*$, which we do by showing by induction on $i \geq 0$ that $\sim_f \sqsubseteq \sim_i$. First, let us remark that since $\mathcal{T}$ is in earliest form and is sequential, the function computed by $\mathcal{T}$ without final outputs is exactly $\widehat{f}$. Let $u \sim_f v$ we have 1) $u \in \mathrm{dom}(f) \Leftrightarrow v \in \mathrm{dom}(f)$, which means that $[u]_{\mathcal{A}} \in F \Leftrightarrow [v]_{\mathcal{A}} \in F$. According to the previous remark, $\widehat{f}(u)^{-1}f(u) = t([u]_{\mathcal{A}})$ and since $\widehat{f}(u)^{-1}f(u) = \widehat{f}(v)^{-1}f(v)$ we have $t([u]_{\mathcal{A}}) = t([v]_{\mathcal{A}})$. Similarly, according to the proof of Prop. 2.1.5, we have for any letter a that $\widehat{f}(u)^{-1}\widehat{f}(ua) = \widehat{f}(v)^{-1}\widehat{f}(va)$. Again according to the remark above, we obtain 2) $o([u]_{\mathcal{A}}, a, [ua]_{\mathcal{A}}) = o([v]_{\mathcal{A}}, a, [va]_{\mathcal{A}})$. Hence $\sim_f \sqsubseteq \sim_0$. Now let us assume that for some $i \in \mathbb{N}$ we have $\sim_f \sqsubseteq \sim_i$. Let $u \sim_f v$, we have by hypothesis that $u \sim_i v$. Since $\sim_f$ is a right congruence we also have for any $a \in A$ that $ua \sim_f va$ which means that $ua \sim_i va$, and hence $u \sim_{i+1} v$, which concludes the proof of correctness.

The fact that this procedure takes polynomial time can be argued just like for Moore's algorithm: each refinement step takes polynomial time and since each step merges at least two states, the number of steps is linear in the number of states of the transducer. $\square$

Corollary 2.1.12. *Let $\mathbf{C}$ be a decidable class of congruence, then $\mathbf{C}$-sequentiality of sequential transductions is decidable.*

Proof. Let $\mathcal{T}$ be a sequential transducer realizing f, according to Th. 2.1.11 we can compute the minimal sequential transducer of f. According to Cor. 2.1.9, f is $\mathbf{C}$-sequential if and only if $\mathcal{T}_f$ is a $\mathbf{C}$-transducer, which we can decide since $\mathbf{C}$ is decidable by assumption. □

2.1.3 Determinization preserves aperiodicity

Here we study the determinization of transducers. Determinization is not always possible since not all rational functions are sequential, such as f_{swap} of Ex. 1.4. There is however an algorithm which terminates in the case of sequential functions which we call *powerset construction with delays*. This algorithm is similar to the usual powerset construction, and the idea is to output on a transition the longest common prefix of all possible outputs. In order to obtain the original function, the macro-states of the construction thus have to contain not only states but pairs composed of a state and a remaining output (called a delay). Like for automata this construction, when it terminates, yields a deterministic machine at the cost of an exponential blow-up.

We describe the algorithm given in [BC02] and show that if the initial transducer is aperiodic then the resulting one also is. As a corollary this gives us, as we will see in Sec. 2.3, a procedure to decide if a sequential transduction is $\mathsf{FO}[\leq]$-definable.

C-sequentiality vs C-rationality Let $\mathbf{C}$ be a class of congruences. We start by stating that, in the case of sequential functions, $\mathbf{C}$-sequentiality and $\mathbf{C}$-rationality are not equivalent.

Theorem 2.1.13. *There exists a class of congruences $\mathbf{C}$, and a sequential $\mathbf{C}$-transduction which is not $\mathbf{C}$-sequential.*

Proof. Let $\mathbf{C} = \mathbf{I}$, the class of idempotent congruences. Let f_{idem} be the function given by the transducers $\mathcal{T}_1$ and $\mathcal{T}_2$ of Fig. 2.1 on page 29 and 2.2, respectively. The transducer $\mathcal{T}_1$ is idempotent and non-sequential whereas $\mathcal{T}_2$ is sequential and not in $\mathbf{I}$. Furthermore $\mathcal{T}_2$ is actually *the* minimal sequential transducer of f_{idem} which means, according to Cor. 2.1.9, that f_{idem} is not $\mathbf{I}$-sequential. □

Determinization algorithm We describe the powerset construction with delays which builds a sequential transducer realizing the same function as the input transducer. Since not all functions are sequential, the algorithm may not terminate in general. When it does terminate, one can show that the lengths of the delays are polynomial in the input transducer, which means that the overall worst-case complexity is, as for automata determinization, exponential.

Let $\mathcal{T} = (\mathcal{A}, i, o, t)$ be a transducer, with underlying automaton $\mathcal{A} = (Q, \Delta, I, F)$, realizing a function f. We describe $\mathcal{S} := (\mathcal{D}, i', o', t')$, with $\mathcal{D} := (Q', \Delta', S_0, F')$, which also realizes f. Note that $\mathcal{S}$ is sequential by definition but may have an infinite number of states when f is not sequential. We assume that $\mathcal{T}$ is trim and in earliest form, without loss of generality.

Let $\alpha := \bigwedge_{q\in I} i(q)$ and let $S_0 := \{(q, \alpha^{-1}i(q)) \mid q \in I\}$, with $i'(S_0) := \alpha$. We build Q' and Δ' up from S_0 using the following steps. Let P be a state already constructed in Q' and let $a \in A$. We define $R := \left\{(q, \beta\gamma) \mid (p, \beta) \in P, \text{ and } p \xrightarrow{a|\gamma} q\right\}$. Let $\alpha := \bigwedge_{(q,\beta)\in R} \beta$, then we add the state $S := \{(q, \alpha^{-1}\beta) \mid (q, \beta) \in R\}$ to Q' and the transition $P \xrightarrow{a|\alpha} S$ to Δ'. We only have left to define F' and t'. $F' := \{P \mid \exists(p, \alpha) \in P, p \in F\}$ and $t'(P) := \alpha t(p)$ such that $(p, \alpha) \in P$ and $p \in F$. Note that the definition of t' may seem ambiguous but it is not due to the functionality of f ($\mathcal{T}$ is assumed trim).

The previous construction as well as the following theorem are taken from [BC02].

Theorem 2.1.14. *Let $\mathcal{T}$ be a transducer realizing a sequential function f and let $\mathcal{S}$ be the transducer obtained by powerset construction with delays. Then $\mathcal{S}$ is finite and realizes f.*

Aperiodicity As we have seen determinization does not preserve transition congruence of a machine. However determinization of an aperiodic transducer always produces an aperiodic transducer.

Lemma 2.1.15. *Let $\mathcal{T}$ be an aperiodic transducer realizing a sequential function f and let $\mathcal{S}$ be the transducer obtained by powerset construction with delays. Then $\mathcal{S}$ is aperiodic.*

Proof. Let $\mathcal{T} = (\mathcal{A}, i, o, t)$, with $\mathcal{A} = (Q, \Delta, I, F)$, be an **A**-transducer realizing a sequential function $f : A^* \to B^*$. Let $\mathcal{S} = (\mathcal{D}, i', o', t')$, with $\mathcal{D} = (Q', \Delta', S_0, F')$, be the sequential transducer obtained from $\mathcal{T}$ by powerset construction with delays. We want to show that $\mathcal{S}$ is aperiodic and actually we will show that $\mathcal{D}$ is counter-free, which is equivalent according to Prop. 1.3.9 since $\mathcal{S}$ is sequential. $\mathcal{A}$ is aperiodic so there is an integer n such that $\forall u \in A^*$, $u^n \equiv_{\mathcal{A}} u^{n+1}$.

Let $u \in A^+$ be a word, let k be a positive integer and let $R_0 \xrightarrow{u|\alpha_0}_{\mathcal{S}} R_1 \dots R_{k-1} \xrightarrow{u|\alpha_{k-1}}_{\mathcal{S}} R_0$ denote a counter in $\mathcal{S}$. Let us assume that k is the size of the smallest such counter, which means that all R_js are pairwise distinct, we want to show $k = 1$.

Let $\Gamma_0 := \alpha_0 \cdots \alpha_{k-1}$, for $1 \leq j < k$ let $\Gamma_j := \alpha_j \cdots \alpha_{k-1}\alpha_0 \cdots \alpha_{j-1}$ and let us note that $\Gamma_j \alpha_j = \alpha_j \Gamma_{j+1 \bmod k}$. Let $R = \{q_1, \dots, q_m\}$ denote the states appearing in R_0. For $0 \leq j < k$, the states of R_j are exactly the states which can be reached in $\mathcal{A}$ from some state of R_0 by reading $u^{kn+j} \equiv_{\mathcal{A}} u^{kn}$. This means that the states of R_j are the same as the states of R_0, namely $q_1, \dots, q_m$. Thus let $R_j = \{(q_1, \beta_{1,j}), \dots, (q_m, \beta_{m,j})\}$.

Let $i, i' \in \mathbf{m}$, and let $q_i \xrightarrow{u|\alpha_{i,i'}}_{\mathcal{T}} q_{i'}$ denote a run in $\mathcal{T}$ when it exists. By definition of $\mathcal{S}$, we have for any $0 \leq j < k$:

$$\beta_{i,j}\alpha_{i,i'} = \alpha_j \beta_{i',j+1 \bmod k}$$

Let $q_{i_0} \xrightarrow{u}_{\mathcal{A}} \dots q_{i_{t-1}} \xrightarrow{u}_{\mathcal{A}} q_{i_t}$ such that t is a multiple of k. Thus we obtain for any $0 \leq j, j' < k$:

$$\begin{array}{rcl} \beta_{i_0,j}\alpha_{i_0,i_1} \cdots \alpha_{i_{t-1},i_t} & = & \alpha_j \cdots \alpha_{j-1}\beta_{i_t,j} \\ \beta_{i_0,j'}\alpha_{i_0,i_1} \cdots \alpha_{i_{t-1},i_t} & = & \alpha_{j'} \cdots \alpha_{j'-1}\beta_{i_t,j'} \end{array}$$

Since Q is finite there must be a state $q_l \in R$, such that q_l loops by reading a power of u, meaning that there is an integer t such that $q_l \xrightarrow{u^t} q_l$. For a large enough t we can assume by aperiodicity that t is of the form $t = ks + 1$. Let $l, i_1, \dots, i_{t-1}, l$ denote the state indices of the previous run from q_l to q_l over u^t. Let $\Phi := \alpha_{l,i_1} \cdots \alpha_{i_{t-1},l}$. We have:

$$\begin{array}{rcll} \beta_{l,j}\Phi & = & \Gamma_j^s \alpha_j \beta_{l,j+1} & (1) \\ \beta_{l,j}\Phi^k & = & \Gamma_j^{ks+1}\beta_{l,j} & (2) \\ \beta_{l,j+1}\Phi^k & = & \Gamma_{j+1}^{ks+1}\beta_{l,j+1} & (3) \end{array}$$

From (2) we have $|\Phi| = (ks+1)\frac{|\Gamma_0|}{k}$. From (1) we thus obtain: $|\beta_{l,j+1}| - |\beta_{l,j}| = \frac{|\Gamma_0|}{k} - |\alpha_j|$. Notice that this holds for any looping state q_l but the difference does not depend on the state itself.

Let us now consider q_i, a state which is not necessarily a looping state. Any state must be reachable from some looping state, since all states in R can be reached from some state of R by an arbitrarily large power of u. Let q_l be a looping state which can reach q_i by a run over u^{ks}. Again let $l, i'_1, \dots, i'_{ks-1}, i$ denote the sequence of indices of such a run and let $\Psi := \alpha_{l,i'_1} \cdots \alpha_{i'_{ks-1},i}$. We have:

$$\begin{array}{rcll} \beta_{l,j}\Psi & = & \Gamma_j^{ks}\beta_{i,j} & (4) \\ \beta_{l,j+1}\Psi & = & \Gamma_{j+1}^{ks}\beta_{i,j+1} & (5) \end{array}$$

Taking the lengths of words of (4) and (5), and taking the difference between the two equalities we obtain: $|\beta_{i,j+1}| - |\beta_{i,j}| = |\beta_{l,j+1}| - |\beta_{l,j}| = \frac{|\Gamma_0|}{k} - |\alpha_j|$ which again does not depend on i. Thus we obtain that for any state $q_i \in R$, looping or otherwise, $|\beta_{i,j+1}| - |\beta_{i,j}| = \frac{|\Gamma_0|}{k} - |\alpha_j|$.

If we assume that $\Gamma_0 = \epsilon$, then in particular $|\beta_{i,j+1}| = |\beta_{i,j}|$ for any state q_i. From (1), we have that $\beta_{l,j} = \beta_{l,j+1}$ for any looping state q_l. Then combining (4) and (5) we obtain $\beta_{i,j} = \beta_{i,j+1}$ for any state q_i. Hence all R_js are identical which means that $k = 1$.

Let us now assume that $\Gamma_0 \neq \epsilon$. Since equations (4) and (5) can have an arbitrarily large common suffix, we know that for any state q_i, either $\beta_{i,j}$ is a suffix of $\beta_{i,j+1}$ or *vice versa.* Let $\{x, y\} = \{j, j+1\}$ such that $\beta_{i,x} = \gamma_i\beta_{i,y}$. Note that x, y do not depend on i since $|\beta_{i,j+1}| - |\beta_{i,j}| = \frac{|\Gamma_0|}{k} - |\alpha_j|$, and furthermore the size of γ_i does not depend on i either. If $x = j$, since (4) and (5) can have an arbitrarily large common suffix, we have that γ_i is a suffix of Γ_{j+1}^{ks} which does not depend on i. Hence γ_i is a common prefix of $\beta_{i',j}$ for all $i' \in \mathbf{m}$, which means that $\gamma_i = \epsilon$ by definition of $\mathcal{S}$. Thus all R_js are equal which means that $k = 1$. Similarly, if $x = j+1$, then γ_i is a suffix of Γ_j^{ks}, and with the same reasoning, we conclude that $k = 1$. □

Theorem 2.1.16. *A sequential function is* **A***-rational if and only if it is* **A***-sequential.*

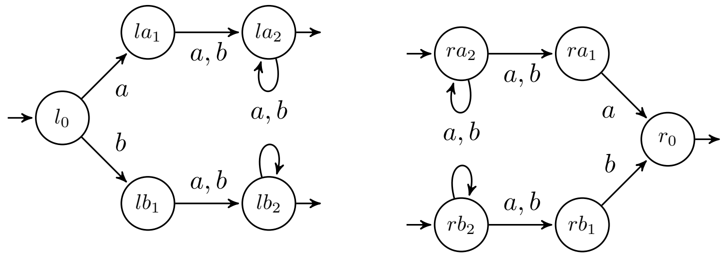
Proof. Let f be sequential function. If f is **A**-sequential then in particular it is **A**-rational. If f is **A**-rational, then there exists an aperiodic transducer realizing it. According to Lem. 2.1.15, the sequential transducer obtained by powerset construction with delays is also aperiodic which means that f is **A**-sequential. □

Remark 2.1.17. Note that all the results of this section can easily be extended symmetrically to right sequential function.

2.2 Algebraic characterization of rational functions

The goal of this section is to obtain an algorithm to decide **C**-rationality of transductions, for a decidable class of congruences **C**. The algebraic characterization in the case of rational languages is tied to the existence of a deterministic model, deterministic automata. This is the main reason why sequential functions can be characterized by a syntactic object. We thus consider a deterministic model of transductions called bimachines. A bimachine is a model of computation introduced in [Sch61] and named as such by [Eil74]. Bimachines are equivalent in expressive power to functional transducers and one of their main features is that they work in a deterministic fashion. As the name suggests, bimachines are made up of two automata: one deterministic, called the left automaton, and one co-deterministic, called the right automaton. Intuitively, a bimachine can be seen as a sequential transducer with look-ahead, where the look-ahead information is given by the right automaton. However we also use the fact that the roles of the two automata are completely symmetrical.

We define bimachines and show how one can go from a bimachine to an unambiguous transducer while preserving the transition congruence, which was already known [RS95]. This allows us to reduce the study of **C**-transductions to the study of **C**-bimachines, for any class **C**. Then we introduce the syntactic congruence of a function with respect to a fixed look-ahead, an object



l	α	r	$o(l, \alpha, r)$
l_0	α	$r\gamma_j$	γ
$l\beta_i$	α	r_0	β
$l\beta_i$	α	$r\gamma_j$	α

Figure 2.3: Left and right automata and output table of a bimachine realizing f_{swap}.

introduced in [RS91] and we use it to minimize bimachines through a two-sided Choffrut-like minimization. Finally we introduce the delay congruence, a syntactic object (also taken from [RS91]) which yields a minimal look-ahead, and thus a canonical bimachine. As it was noticed in [RS91] however, this canonical bimachine does not necessarily capture all the algebraic properties of a transduction. Refining the techniques of [RS91] we are able to show that any rational function has a finite number of minimal bimachines, which was conjectured in [RS91]. It turns out that we can bound the size of minimal bimachines, using the canonical bimachine, and thus we obtain the main result of this section: an algorithm to decide **C**-definability of rational functions, for any decidable class of congruences **C**.

2.2.1 Bimachines and transductions

We define bimachines, and the related notions. We also show how to go from a bimachine to a transducer while preserving transition congruences.

Bimachines A *bimachine* over alphabets A, B is given as a tuple $\mathcal{B} = (\mathcal{L}, \mathcal{R}, i, o, t)$ where $\mathcal{L} = (Q_{\mathcal{L}}, \Delta_L a, \{l_0\}, F_{\mathcal{L}})$ is a deterministic accessible automaton, called the *left automaton of* $\mathcal{B}$, $\mathcal{R} = (Q_{\mathcal{R}}, \Delta_R, I_{\mathcal{R}}, \{r_0\})$ is a co-deterministic co-accessible and co-complete automaton, called the *right automaton of* $\mathcal{B}$, $i : Q_{\mathcal{R}} \to B^*$ is the *initial function*, $o : Q_{\mathcal{L}} \times A \times Q_{\mathcal{R}} \to B^*$ is the *output function* and $t : Q_{\mathcal{L}} \to B^*$ is the *final function*. We also add the semantic restriction that $\mathcal{L}$ and $\mathcal{R}$ must recognize the same language[1].

Let u be a word and let l and r be runs of $\mathcal{L}$ and $\mathcal{R}$, respectively, over u. We extend o to $Q_{\mathcal{L}} \times A^* \times Q_{\mathcal{R}}$ by setting $o(l(1), u, r(|r|)) := \prod_{1 \le j \le |u|} o(l(j), u(j), r(j+1))$. Furthermore if l and r are accepting, let $\alpha := i(r(|r|)) \cdot o(l(1), u, r(|r|)) \cdot t(l(|l|))$. We say that the pair (u, α) is *realized* by $\mathcal{B}$ and we denote by $[\![\mathcal{B}]\!]$ the set of pairs realized by $\mathcal{B}$.

Example 2.2.1. Let us consider the bimachine $\mathcal{B}_{\text{swap}} = (\mathcal{L}, \mathcal{R}, i, o, t)$ with $\mathcal{L}$ and $\mathcal{R}$ the automata of Fig. 2.3. We set $i : r \mapsto \epsilon$, $t : l \mapsto \epsilon$. Let $\alpha, \beta, \gamma \in \{a, b\}$ and $i, j \in \mathbf{2}$, we define $o(l_0, \alpha, r\gamma_j) := \gamma$, $o(l\beta_i, \alpha, r_0) := \beta$ and in all other cases, $o(l, \alpha, r) := \alpha$. The function realized by $\mathcal{B}$ is f_{swap} (*cf.* Ex. 1.5.4), and in Fig. 2.4 on the facing page we give an example of a run of $\mathcal{B}_{\text{swap}}$ over the word *aabb*.

C-bimachines Let $i \in \mathbf{2}$, and let $\mathcal{B}_i$ be a bimachine with left and right automata $\mathcal{L}_i$ and $\mathcal{R}_i$, respectively. We say that $\mathcal{B}_1$ is *finer* than $\mathcal{B}_2$, which we denote by $\mathcal{B}_1 \sqsubseteq \mathcal{B}_2$, if we have both

[1]This can be decided in PTime by checking the emptiness of the product of $\mathcal{R}$ with the complement of $\mathcal{L}$ and *vice versa.*

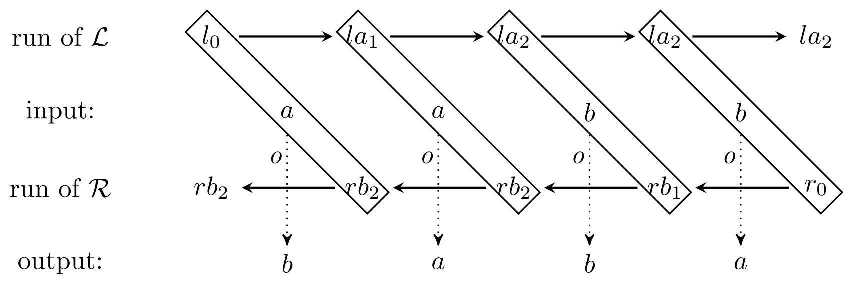


Figure 2.4: A run of $\mathcal{B}_{\text{swap}}$ over the word *aabb*.

$\mathcal{L}_1 \sqsubseteq \mathcal{L}_2$ and $\mathcal{R}_1 \sqsubseteq \mathcal{R}_2$. A bimachine realizing a function f is called *minimal* if there is no coarser bimachine realizing f. Two bimachines are called *equivalent* if they realize the same function and are both finer than each other. We will often abuse definitions and say that two bimachines are equal instead of equivalent. The *transition congruence* of a bimachine $\mathcal{B}$ with automata $\mathcal{L}$ and $\mathcal{R}$ is defined as $\equiv_{\mathcal{B}} := (\equiv_{\mathcal{L}} \sqcap \equiv_{\mathcal{R}})$. Let $\mathbf{C}$ be a class of congruences, a bimachine is called a *$\mathbf{C}$-bimachine* if both its automata are $\mathbf{C}$-automata. Equivalently, by closure under intersection and coarser congruences, a bimachine is a $\mathbf{C}$-bimachine if and only if its transition congruence is in $\mathbf{C}$.

Proposition 2.2.2. *There exists a function f for which there is more than one equivalence class of minimal bimachines.*

Proof. Let us consider f_{idem} the function given by the sequential transducer of Fig. 2.2 on page 30. Any sequential transducer realizing a total function can be seen as a bimachine with a trivial right automaton. Let $\mathcal{B}$ be such a bimachine with the underlying automaton of the minimal sequential transducer as left automaton and the trivial automaton as right automaton. There cannot be a bimachine realizing f_{idem} with coarser automata otherwise we would obtain a sequential transducer smaller than the minimal one. Hence $\mathcal{B}$ is a minimal bimachine. However, one can easily see that f_{idem} is also a right sequential function, and can be realized by a bimachine with a trivial left automaton. Hence f_{idem} has at least two incomparable minimal bimachines. □

From bimachines to transducers It is quite easy to show that from a $\mathbf{C}$-bimachine one can obtain an unambiguous $\mathbf{C}$-transducer realizing the same function, and this was already noticed in [RS95].

Proposition 2.2.3. *Let $\mathcal{B}$ be a bimachine realizing a function f with left and right automata $\mathcal{L}$ and $\mathcal{R}$, respectively. Then one can construct in* PTime *an unambiguous transducer realizing f with underlying automaton $\mathcal{L} \times \mathcal{R}$.*

Proof. Let $\mathcal{B} = (\mathcal{L}, \mathcal{R}, i, o, t)$ be a bimachine realizing a function f with left automaton $\mathcal{L} = (Q_{\mathcal{L}}, \Delta_L a, \{l_0\}, F_{\mathcal{L}})$ and right automaton $\mathcal{R} = (Q_{\mathcal{R}}, \Delta_R, I_{\mathcal{R}}, \{r_0\})$. The main idea is to consider the product of the two automata. The transducer will guess the run of the right automaton, and co-determinism ensures that there is only one possible run. We define a transducer $\mathcal{T} := (\mathcal{A}, i', o', t')$ with $\mathcal{A} := (Q, \Delta, I, F)$ by:

- $Q := Q_{\mathcal{L}} \times Q_{\mathcal{R}}$
- $\Delta := \{((l, r'), a, (l', r)) \mid \ a \in A,\ (l, a, l') \in \Delta_{\mathcal{L}},\ (r', a, r) \in \Delta_{\mathcal{R}}\}$

- $I := \{l_0\} \times I_{\mathcal{R}}$
- $F := F_{\mathcal{L}} \times \{r_0\}$
- $i'(l_0, r) := i(r)$
- $o'((l, r'), a, (l', r)) := o(l, a, r)$
- $t'(l, r_0) := t(l)$

By construction $\mathcal{T}$ realizes f and its underlying automaton is indeed $\mathcal{L} \times \mathcal{R}$. □

Corollary 2.2.4. *Let $\boldsymbol{C}$ be a class of congruences. A transduction realized by a $\boldsymbol{C}$-bimachine is a $\boldsymbol{C}$-transduction.*

Proof. Let $\mathcal{B}$ be a bimachine realizing a function f with $\mathbf{C}$-automata $\mathcal{L}$ and $\mathcal{R}$. Let $\mathcal{T}$ be the transducer obtained from Prop. 2.2.3, with underlying automaton $\mathcal{L} \times \mathcal{R}$. Let $u, v \in A^*$ be two words such that $u \equiv_{\mathcal{L}} v$ and $u \equiv_{\mathcal{R}} v$. This means that $u \equiv_{\mathcal{L}\times\mathcal{R}} v$, and thus $(\equiv_{\mathcal{L}} \sqcap \equiv_{\mathcal{R}}) \sqsubseteq \equiv_{\mathcal{L}\times\mathcal{R}}$. Since $\mathbf{C}$ is closed under intersection of congruences and coarser congruences, this means that $\equiv_{\mathcal{L}\times\mathcal{R}}$ is in $\mathbf{C}$, hence $\mathcal{T}$ is a $\mathbf{C}$-transducer and f is a $\mathbf{C}$-transduction. □

In [RS95] it was noticed that one can easily go from an unambiguous transducer to a bimachine while preserving the transition congruence. We show later in this section a stronger result, namely that the transducer needs *not* be unambiguous for the proposition to hold. The result however is not trivial and relies on the notions of canonical bimachine, and bimachine minimization.

2.2.2 Bimachines and minimization

Just like for sequential functions, the minimization of bimachines relies on a syntactic congruence. We define two different minimizations which are completely symmetrical: left minimization and right minimization. Left minimization consists in fixing the right automaton and then computing the minimal left automaton possible given the look-ahead information provided by the right automaton. Our main conceptual tool is the syntactic congruence of a function *with respect to* a fixed right automaton, taken from [RS91]. From this we are able to compute, just like for sequential functions, a minimal left automaton in PTime. We also show that composing left and right minimization (either way) always produces a minimal bimachine. Finally we obtain that one can go from a (possibly ambiguous) transducer to a bimachine while preserving the transition congruence.

$\mathcal{R}$-syntactic congruence Let $f : A^* \to B^*$ be a function and let $\mathcal{R}$ be a right automaton recognizing $\mathrm{dom}(f)$. We define for each state of $\mathcal{R}$, *i.e.* each equivalence class $[v]_{\mathcal{R}}$ of $\approx_{\mathcal{R}}$, a function $\widehat{f}_{[v]_{\mathcal{R}}} : u \mapsto \bigwedge \{f(uw) \mid w \approx_{\mathcal{R}} v\}$, which is defined over $\mathrm{dom}(f)v^{-1}$. Note that the right automaton will often be implicit and we will rather write $\widehat{f}_v$. Intuitively, this function outputs over a word u the longest possible word, with the look-ahead information that the suffix is in $[v]_{\mathcal{R}}$.

Definition 2.2.5. Let $f : A^* \to B^*$ be a function and let $\mathcal{R}$ be a right automaton recognizing $\mathrm{dom}(f)$. The *$\mathcal{R}$-syntactic congruence* of f is defined for $u, v \in A^*$ by: $u \sim_f^{\mathcal{R}} v$ if

- $u^{-1}\mathrm{dom}(f) = v^{-1}\mathrm{dom}(f)$ and
- $\forall w \in A^*$, $\widehat{f}_w(u)^{-1} f(uw) = \widehat{f}_w(v)^{-1} f(vw)$

Intuitively, u and v are equivalent if when we remove the contributions of u and v from $f(uw)$ and $f(vw)$, respectively, we obtain the same word left to write. Note here that the contributions of u and v are computed with the knowledge that the suffix is equivalent to w, with respect to $\mathcal{R}$.

Proposition 2.2.6. *Let f be a function and let $\mathcal{R}$ be a right automaton recognizing* $\mathrm{dom}(f)$*, then $\sim_f^{\mathcal{R}}$ is a right congruence.*

Proof. Let $f : A^* \to B^*$ be a function, let $\mathcal{R}$ be a right automaton recognizing $\mathrm{dom}(f)$, let $u \sim_f^{\mathcal{R}} v$ and let $a \in A$. We want to show that $ua \sim_f^{\mathcal{R}} va$. First $(ua)^{-1}\mathrm{dom}(f) = a^{-1}(u^{-1}\mathrm{dom}(f)) = (va)^{-1}\mathrm{dom}(f)$. We know that for any word $w \in u^{-1}\mathrm{dom}(f)$, $\widehat{f}_w(u)^{-1}f(uw) = \widehat{f}_w(v)^{-1}f(vw)$, and we let $g(w)$ denote this word. Let $aw \in u^{-1}\mathrm{dom}(f)$.

$$\begin{array}{rcl}
\widehat{f}_w(ua)^{-1}f(uaw) & = & \left(\bigwedge_{z\approx_{\mathcal{R}}w} f(uaz)\right)^{-1} f(uaw) \\
& = & \left(\bigwedge_{z\approx_{\mathcal{R}}w} \widehat{f}_{aw}(u)g(az)\right)^{-1} \widehat{f}_{aw}(u)g(aw) \\
& = & \left(\widehat{f}_{aw}(u)\bigwedge_{z\approx_{\mathcal{R}}w} g(az)\right)^{-1} \widehat{f}_{aw}(u)g(aw) \\
& = & \left(\bigwedge_{z\approx_{\mathcal{R}}w} g(az)\right)^{-1} g(aw) \\
& = & \widehat{f}_w(va)^{-1}f(vaw)
\end{array}$$

Hence $ua \sim_f^{\mathcal{R}} va$ which concludes the proof. □

Example 2.2.7. Let us consider again the function f_{swap} and $\mathcal{R}$ the right automaton of $\mathcal{B}_{\mathrm{swap}}$ given in Ex. 2.2.1. We give the classes of the congruence $\sim_{f_{\mathrm{swap}}}^{\mathcal{R}}$: $\{\epsilon, a, b, aA^+, bA^+\}$ which exactly coincides with the states of the left automaton of Fig. 2.3 on page 36.

Left minimal bimachine Let $f : A^* \to B^*$ be a function and let $\mathcal{R} := (Q_{\mathcal{R}}, \Delta_{\mathcal{R}}, I_{\mathcal{R}}, \{r_0\})$ be a right automaton recognizing $\mathrm{dom}(f)$. Using the $\mathcal{R}$-syntactic congruence of f we define the *left minimal bimachine* of f with respect to $\mathcal{R}$. We set $\mathcal{B}_f(\mathcal{R}) := (\mathcal{L}_f(\mathcal{R}), \mathcal{R}, i, o, t)$ with $\mathcal{L}_f(\mathcal{R}) := (Q_{\mathcal{L}}, \Delta_{\mathcal{L}}, l_0, F_{\mathcal{L}})$ defined by:

- $Q_{\mathcal{L}} := \left\{[u]_f^{\mathcal{R}} \mid u \in {\downarrow}\mathrm{dom}(f)\right\}$
- $\Delta_{\mathcal{L}} := \left\{([u]_f^{\mathcal{R}}, a, [ua]_f^{\mathcal{R}})\right\}$
- $l_0 := [\epsilon]_f^{\mathcal{R}}$
- $F_{\mathcal{L}} := \left\{[u]_f^{\mathcal{R}} \mid u \in \mathrm{dom}(f)\right\}$
- $i([u]_{\mathcal{R}}) := \widehat{f}_u(\epsilon)$
- $o([u]_f^{\mathcal{R}}, a, [v]_{\mathcal{R}}) := \widehat{f}_{av}(u)^{-1}\widehat{f}_v(ua)$
- $t([u]_f^{\mathcal{R}}) := \widehat{f}_\epsilon(u)^{-1}f(u)$

The definition of the left minimal bimachine of a function f with respect to a right automaton $\mathcal{R}$ is very similar to the definition of the minimal sequential transducer of a function. Intuitively, it outputs over a word u, and given a class $[w]_{\mathcal{R}}$, the longest possible word knowing that the input begins with u and ends with a word equivalent to w. Again this bimachine may have an infinite number of states, in general.

Remark 2.2.8. When f is a total function, and $\mathcal{R}$ is the trivial right automaton with only one state, then $\sim_f^{\mathcal{R}} = \sim_f$ and since $\mathcal{B}_f(\mathcal{R})$ does not make use of $\mathcal{R}$, it is basically the same as $\mathcal{T}_f$.

Note that, as for sequential functions, $\sim_{\mathcal{L}_f(\mathcal{R})} = \sim_f^{\mathcal{R}}$. In particular, when $\sim_f^{\mathcal{R}}$ has infinite index then the "bimachine" $\mathcal{B}_f(\mathcal{R})$ is actually infinite.

We show, as for the minimal sequential transducer of Section 2.1, that the outputs of the machine we obtain from $\sim_f^{\mathcal{R}}$ are well-defined.

Proposition 2.2.9. *Let f be a function and let $\mathcal{R}$ be a right automaton recognizing* $\mathrm{dom}(f)$*. The outputs of $\mathcal{B}_f(\mathcal{R})$ are $\mathcal{R}$ is well-defined.*

Proof. Let f be a function and let $\mathcal{R}$ be a right automaton recognizing $\mathrm{dom}(f)$. We show that the output functions of $\mathcal{B}_f(\mathcal{R})$ are well-defined. The initial output function is obviously well-defined. Let $u \sim_f^{\mathcal{R}} v$ and let $a \in A$ such that $ua \in {\downarrow}\mathrm{dom}(f)$. We want to show that the output function o is well-defined, meaning that $\forall w,\ \widehat{f}_{aw}(u)^{-1}\widehat{f}_w(ua) = \widehat{f}_{aw}(v)^{-1}\widehat{f}_w(va)$. As before we have that $\widehat{f}_{aw}(u)^{-1}f(uaw) = \widehat{f}_{aw}(v)^{-1}f(vaw)$, for any $a \in A$, $aw \in u^{-1}\mathrm{dom}(f)$, and we denote this word by $g(aw)$.

$$\begin{aligned}
\widehat{f}_{aw}(u)^{-1}\widehat{f}_w(ua) &= \left(\bigwedge_{z\approx_{\mathcal{R}}aw} f(uz)\right)^{-1} \bigwedge_{z\approx_{\mathcal{R}}w} f(uaz)\\
&= \left(\bigwedge_{z\approx_{\mathcal{R}}aw} \widehat{f}_{aw}(u)g(z)\right)^{-1} \bigwedge_{z\approx_{\mathcal{R}}w} \widehat{f}_{aw}(u)g(az)\\
&= \left(\bigwedge_{z\approx_{\mathcal{R}}aw} g(z)\right)^{-1} \bigwedge_{z\approx_{\mathcal{R}}w} g(az)\\
&= \widehat{f}_{aw}(v)^{-1}\widehat{f}_w(va)
\end{aligned}$$

The final output function is well-defined, since $\widehat{f}_\epsilon(u)^{-1}f(u) = \widehat{f}_\epsilon(v)^{-1}f(v)$ for $u, v \in \mathrm{dom}(f)$, by definition of $\sim_f^{\mathcal{R}}$. □

We now can show that $\mathcal{B}_f(\mathcal{R})$ realizes f. However, $\mathcal{B}_f(\mathcal{R})$ may be infinite if $\mathcal{R}$ does not give enough look-ahead information to realize f sequentially.

Theorem 2.2.10. *Let f be a function and let $\mathcal{R}$ be a right automaton recognizing* $\mathrm{dom}(f)$*. The bimachine $\mathcal{B}_f(\mathcal{R})$ realizes f.*

Proof. Let $f : A^* \to B^*$ be a function, let $\mathcal{R}$ be a right automaton recognizing $\mathrm{dom}(f)$ and let $u \in \mathrm{dom}(f)$. By definition of the outputs of the left minimal bimachine $\mathcal{B}_f(\mathcal{R})$, $[\![\mathcal{B}_f(\mathcal{R})]\!](u) = \widehat{f}_u(\epsilon) \cdot \left(\prod_{0\leq i<|u|} \widehat{f}_{u(i+1:)}(u(:i))^{-1}\widehat{f}_{u(i+2:)}(u(:i+1))\right) \cdot \widehat{f}_\epsilon(u)^{-1}f(u) = f(u)$. □

We now show that the left minimal bimachine with respect to some right automaton is indeed minimal among bimachines realizing the same function with the same right automaton. This was already shown in [RS91], but only in the case of total functions.

Theorem 2.2.11. *Let f be a transduction realized by a bimachine with left and right automata $\mathcal{L}$ and $\mathcal{R}$, respectively. Then $\sim_{\mathcal{L}} \sqsubseteq \sim_f^{\mathcal{R}}$.*

Proof. Let $\mathcal{B} = (\mathcal{L}, \mathcal{R}, i, o, t)$ be a bimachine with automata $\mathcal{L} = (Q_{\mathcal{L}}, \Delta_{\mathcal{L}}, \{l_0\}, F_{\mathcal{L}})$ and $\mathcal{R} = (Q_{\mathcal{R}}, \Delta_{\mathcal{R}}, I_{\mathcal{R}}, \{r_0\})$ realizing f and let $u \sim_{\mathcal{L}} v$. We have $u^{-1}\mathrm{dom}(f) = v^{-1}\mathrm{dom}(f)$ since $\mathcal{L}$ recognizes $\mathrm{dom}(f)$.

Let $w \in u^{-1}\mathrm{dom}(f)$, we want to show that $\widehat{f}_w(u)^{-1}f(uw) = \widehat{f}_w(v)^{-1}f(vw)$. Let $\alpha := i([uw]_{\mathcal{R}})o(l_0, u, [w]_{\mathcal{R}})$ and $\beta := i([vw]_{\mathcal{R}})o(l_0, v, [w]_{\mathcal{R}})$ denote the outputs before reading w of the runs of $\mathcal{B}$ over uw and vw respectively. Let $\gamma := \bigwedge\{o([u]_{\mathcal{L}}, z, r_0)t([uz]_{\mathcal{L}}) \mid z \approx_{\mathcal{R}} w\}$ be the longest common prefix of outputs from state $[u]_{\mathcal{L}}$ with look-ahead information $[w]_{\mathcal{R}}$.

Then we have $\widehat{f}_w(u) = \alpha\gamma$, $\widehat{f}_w(v) = \beta\gamma$, $f(uw) = \alpha o([u]_{\mathcal{L}}, w, r_0)t([uw]_{\mathcal{L}})$ and also $f(vw) = \beta o([u]_{\mathcal{L}}, w, r_0)t([uw]_{\mathcal{L}})$. Finally we obtain that $\widehat{f}_w(u)^{-1}f(uw) = \gamma^{-1}o([u]_{\mathcal{L}}, w, r_0)t([uw]_{\mathcal{L}}) = \widehat{f}_w(v)^{-1}f(vw)$. □

Corollary 2.2.12. *Let f be a function and let $\mathcal{R}$ be a right automaton recognizing* $\mathrm{dom}(f)$*. Then there exists a bimachine realizing f with right automaton $\mathcal{R}$ if and only if $\sim_f^{\mathcal{R}}$ has finite index.*

Proof. If $\mathcal{B}$ is a bimachine realizing f with left and right automata $\mathcal{L}$ and $\mathcal{R}$, respectively, then according to Th. 2.2.11, $\sim_{\mathcal{L}} \sqsubseteq \sim_f^{\mathcal{R}}$ which means that $\sim_f^{\mathcal{R}}$ has finite index. Conversely, let us assume that $\sim_f^{\mathcal{R}}$ has finite index, then $\mathcal{B}_f(\mathcal{R})$ is finite and realizes f, from Prop. 2.2.10. □

Left minimization algorithm Just like for sequential transducers, we show how to minimize the left automaton of a bimachine by successive refinements. In this context we require a notion of earliest bimachine to apply the refinement technique.

Let $\mathcal{B} = (\mathcal{L}, \mathcal{R}, i, o, t)$ be a bimachine, let $u, v \in A^*$ such that $uv \in \mathrm{dom}(f)$. Let $\alpha_{u,v} := \bigwedge \{o([u]_{\mathcal{L}}, w, r_0)t([uw]_{\mathcal{L}}) \mid w \approx_{\mathcal{R}} v\}$ be the longest common prefix of outputs from $[u]_{\mathcal{L}}$ with look-ahead $[v]_{\mathcal{R}}$. Then $\mathcal{B}$ is called in *earliest form* if for any $u, v \in A^*$, we have $\alpha_{u,v} = \epsilon$ (note that we safely write $\alpha_{u,v}$ instead of $\alpha_{[u]_{\mathcal{L}},[v]_{\mathcal{R}}}$ to simplify the notations).

Proposition 2.2.13. *Given a bimachine $\mathcal{B}$, one can obtain in* PTime *a new bimachine $\widehat{\mathcal{B}}$ in earliest form realizing the same function.*

Proof. We give the definition of $\widehat{\mathcal{B}}$, show that it realizes the same function and is indeed in earliest form, and finally we give a procedure to compute it. Let $\mathcal{B} = (\mathcal{L}, \mathcal{R}, i, o, t)$ be a bimachine realizing a function f and let $u, v \in A^*$. Let $\alpha_{u,v}$ denote the longest common prefix of outputs from $[u]_{\mathcal{L}}$ with look-ahead $[v]_{\mathcal{R}}$. We define $\widehat{\mathcal{B}} = \left(\mathcal{L}, \mathcal{R}, \widehat{i}, \widehat{o}, \widehat{t}\right)$ by:

- $\widehat{i}([v]_{\mathcal{R}}) := i([v]_{\mathcal{R}})\alpha_{\epsilon,[v]_{\mathcal{R}}}$
- $\widehat{o}([u]_{\mathcal{L}}, a, [v]_{\mathcal{R}}) := \alpha_{u,av}^{-1} o([u]_{\mathcal{L}}, a, [v]_{\mathcal{R}})\alpha_{ua,v}$
- $\widehat{t}([u]_{\mathcal{L}}) := \alpha_{u,\epsilon}^{-1} t([u]_{\mathcal{L}})$

One can easily see that all the $\alpha_{u,v}$s cancel out, and thus $\widehat{\mathcal{B}}$ realizes f as well. By construction, $\widehat{\mathcal{B}}$ is in earliest form.

To show that computing the $\alpha_{u,v}$s can be done in PTime, one can consider the transducer obtained from Prop. 2.2.3 and put it in earliest form, which can be done using the result from [BC00] for instance. □

We now give the minimization theorem:

Theorem 2.2.14. *Given a bimachine with right automaton $\mathcal{R}$ realizing a function f, one can compute in* PTime *$\mathcal{B}_f(\mathcal{R})$, the left minimal bimachine with respect to $\mathcal{R}$.*

Proof. Let $\mathcal{B} = (\mathcal{L}, \mathcal{R}, i, o, t)$ be a bimachine realizing a function f with left and right automata $\mathcal{L} = (Q_{\mathcal{L}}, \Delta_{\mathcal{L}}, \{l_0\}, F_{\mathcal{L}})$ and $\mathcal{R} = (Q_{\mathcal{R}}, \Delta_{\mathcal{R}}, I_{\mathcal{R}}, \{r_0\})$, respectively. According to Prop. 2.2.13 we can assume without loss of generality that $\mathcal{B}$ is in earliest form.

We use the same ideas as in the sequential case, but the original partition depends on the outputs for all letters and all states of $\mathcal{R}$.

As usual, we identify the states of $\mathcal{L}$ with the classes of $\sim_{\mathcal{L}}$. The initial partition is given for all words u, v by $u \sim_0 v$ if: 1) $[u]_{\mathcal{L}} \in F_{\mathcal{L}} \Leftrightarrow [v]_{\mathcal{L}} \in F_{\mathcal{L}}$ and $t([u]_{\mathcal{L}}) = t([v]_{\mathcal{L}})$ and 2) $\forall a \in A,\ w \in A^*$, $o([u]_{\mathcal{L}}, a, [w]_{\mathcal{R}}) = o([v]_{\mathcal{L}}, a, [w]_{\mathcal{R}})$.

Let $i \in \mathbb{N}$, we define inductively for all words u, v, $u \sim_{i+1} v$ by: $u \sim_i v$ and $\forall a \in A$, $ua \sim_i va$. Of course we have $\forall i \in \mathbb{N}$ that $\sim_{\mathcal{L}} \sqsubseteq \sim_{i+1} \sqsubseteq \sim_i$, which means that we reach a fixpoint relation for some i, which we denote by $\sim_*$.

Let us first show that $\sim_*$ is fine enough to realize f with $\mathcal{R}$ as right automaton, which means that $\sim_* \sqsubseteq \sim_f^{\mathcal{R}}$ according to Th. 2.2.11. The relation $\sim_*$ is a right congruence since it is a fixpoint for right multiplication. Let $\mathcal{B}_* = (\mathcal{L}_*, \mathcal{R}, i_*, o_*, t_*)$ with $\mathcal{L}_* = (Q_*, \Delta_*, I_*, F_*)$ be defined as:

- $Q_* := \{[u]_* \mid\ u \in {\downarrow}\mathrm{dom}(f)\}$
- $\Delta_* := \{([u]_*, a, [ua]_*) \mid\ a \in A\}$
- $I_* := \{[\epsilon]_*\}$
- $F_* := \{[u]_* \mid\ [u]_{\mathcal{L}} \in F_{\mathcal{L}}\}$
- $i_*([u]_{\mathcal{R}}) := i\,[u]_{\mathcal{R}}$
- $o_*([u]_*, a, [v]_{\mathcal{R}}) := o([u]_{\mathcal{L}}, a, [v]_{\mathcal{R}})$
- $t_*([u]_*) := t([u]_{\mathcal{L}})$

The final states and the final outputs are well defined according to point 1) in the definition of $\sim_0$ given above. The output function is well-defined according to point 2). Given a word in the domain of f, its outputs for $\mathcal{B}_*$ will be exactly the same as for $\mathcal{B}$ which means that $\mathcal{B}_*$ realizes f and hence $\sim_* \sqsubseteq \sim_f^{\mathcal{R}}$, according to Th. 2.2.11.

We only have left to show that $\sim_f^{\mathcal{R}} \sqsubseteq \sim_*$, which we do by showing by induction on $i \geq 0$ that $\sim_f^{\mathcal{R}} \sqsubseteq \sim_i$. First, let us remark that since $\mathcal{B}$ is in earliest form, the function computed by $\mathcal{B}$ without final outputs and with $[v]_{\mathcal{R}}$ as final state is exactly $\widehat{f}_v$. Let $u \sim_f^{\mathcal{R}} v$ we have 1) $u \in \mathrm{dom}(f) \Leftrightarrow v \in \mathrm{dom}(f)$, which means that $[u]_{\mathcal{L}} \in F \Leftrightarrow [v]_{\mathcal{L}} \in F$. According to the previous remark, $\widehat{f}_\epsilon(u)^{-1} f(u) = t([u]_{\mathcal{L}})$ and since $\widehat{f}_\epsilon(u)^{-1} f(u) = \widehat{f}_\epsilon(v)^{-1} f(v)$ we have $t([u]_{\mathcal{L}}) = t([v]_{\mathcal{L}})$. Similarly, according to the proof of Prop. 2.2.9, we have for any letter a and for any word w that $\widehat{f}_{aw}(u)^{-1} \widehat{f}_w(ua) = \widehat{f}_{aw}(v)^{-1} \widehat{f}_w(va)$. Again according to the remark above, we obtain 2) $o([u]_{\mathcal{L}}, a, [w]_{\mathcal{R}}) = o([v]_{\mathcal{L}}, a, [w]_{\mathcal{R}})$. Hence $\sim_f^{\mathcal{R}} \sqsubseteq \sim_0$. Now let us assume that for some $i \in \mathbb{N}$ we have $\sim_f^{\mathcal{R}} \sqsubseteq \sim_i$. Let $u \sim_f v$, we have by hypothesis that $u \sim_i v$. Since $\sim_f^{\mathcal{R}}$ is a right congruence we also have for any $a \in A$ that $ua \sim_f^{\mathcal{R}} va$ which means that $ua \sim_i va$, and hence $u \sim_{i+1} v$, which concludes the proof. $\square$

From transducers to bimachines In [RS95] the authors observe that one can go from bimachines to unambiguous transducers and back while preserving the transition congruence. We now have the tools to extend this result to arbitrary transducers, and show that one can go from a (possibly ambiguous) transducer to a bimachine while preserving the transition congruence. Unsurprisingly this construction causes an exponential blow-up of size.

Theorem 2.2.15. *Let $\mathcal{T}$ be a transducer with underlying automaton $\mathcal{A}$ realizing a function f. One can obtain a bimachine realizing f with left and right automata coarser than $\sim_{\mathcal{A}}$ and $\approx_{\mathcal{A}}$, respectively.*

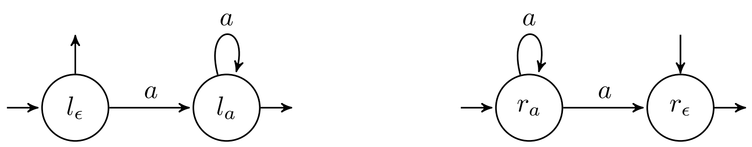


Figure 2.5: Automata of a bimachine realizing f_{idem}.

Proof. Let $\mathcal{T} = (\mathcal{A}, i, o, t)$ be a transducer with underlying automaton $\mathcal{A} = (Q, \Delta, I, F)$ realizing f. Let $\mathcal{R}$ be the right automaton associated with $\approx_{\mathcal{A}}$ recognizing $\text{dom}(f)$. We want to show that $\sim_{\mathcal{A}} \sqsubseteq \sim_f^{\mathcal{R}}$ which will conclude the proof, since $\mathcal{B}_f(\mathcal{R})$ realizes f according to Th. 2.2.10. Let $u \sim_{\mathcal{A}} v$, our goal is to show that $u \sim_f^{\mathcal{R}} v$. We have of course that $u^{-1}\text{dom}(f) = v^{-1}\text{dom}(f)$. Let $w \in u^{-1}\text{dom}(f)$, and let $r_1, \ldots, r_n \in Q$ denote the states of $\mathcal{A}$ which can be reached from I by reading u (or v) and from which there is a final run by reading w. Let $i \in \mathbf{n}$, let $p_i \xrightarrow{u|\alpha_i} r_i$ and $q_i \xrightarrow{v|\beta_i} r_i$ denote initial runs over u and v respectively to r_i. Let $r_i \xrightarrow{w|\gamma_i} s_i$ denote a final run over w from r_i, and let $\delta_i := \bigwedge \left\{ \delta t(s) \mid r_i \xrightarrow{z|\delta} s,\ z \approx_{\mathcal{A}} w \right\}$. Then for any $i \in \mathbf{n}$ we have $\widehat{f_w}(u) = i(p_i)\alpha_i\delta_i$, $f(uw) = i(p_i)\alpha_i\gamma_i t(s_i)$, $\widehat{f_w}(v) = i(q_i)\beta_i\delta_i$ and $f(vw) = i(q_i)\beta_i\gamma_i t(s_i)$. Taking for instance $i = 1$, we obtain $\widehat{f_w}(u)^{-1}f(uw) = \delta_1^{-1}\gamma_1 t(s_1) = \widehat{f_w}(v)^{-1}f(vw)$, hence $u \sim_f^{\mathcal{R}} v$ which concludes the proof. □

As an important corollary, we obtain for any class of congruences $\mathbf{C}$ that one only needs to study $\mathbf{C}$-bimachines in order to study $\mathbf{C}$-rationality.

Corollary 2.2.16. *Let $\mathbf{C}$ be a class of congruences and let f be a function. The function f is $\mathbf{C}$-rational if and only if it is realized by a $\mathbf{C}$-bimachine.*

Proof. This is a simple consequence of both Cor. 2.2.4 and Th. 2.2.15. □

Example 2.2.17. We give in Fig. 2.5 the automata of the bimachine obtained from the transducer of Fig. 2.1 on page 29 using the construction of Th. 2.2.15. The outputs of the bimachine are defined by $o(l_\epsilon, a, r_a) := a$ and all other outputs are set to ϵ. Notice that this bimachine is in $\mathbf{I}$ while the minimal sequential transducer of the function f_{idem}, given in Fig. 2.2 on page 30 is not. We can observe that f_{idem} has three different minimal bimachines: one where all the information is given by the left automaton (see Fig. 2.2 on page 30), *i.e.* the right automaton is trivial, one where the left automaton is trivial and one where the information is split between the two, as in Fig. 2.5.

Remark 2.2.18. Symmetrically, we define $\approx_f^{\mathcal{L}}$, the $\mathcal{L}$-syntactic congruence of a function f, by considering the longest common suffix instead of the longest common prefix. Our minimization results and algorithm obviously extend to the symmetric case, and we naturally define $\mathcal{B}_f(\mathcal{L})$ and $\mathcal{R}_f(\mathcal{L})$, the minimal bimachine with left automaton $\mathcal{L}$ and its right automaton. We continue to only state our results in one direction, for readability.

Left-right minimization We show that composing left and right minimization yields a minimal bimachine but before we show two useful lemmas.

The intuition behind this first lemma is very simple: if an automaton is fine enough to realize a function, then a finer automaton will also do, *a fortiori*.

Lemma 2.2.19. *Let $\mathcal{L}_1 \sqsubseteq \mathcal{L}_2$ be two left automata and let $\mathcal{R}_1 \sqsubseteq \mathcal{R}_2$ be two right automata all recognizing the same language. Let $\mathcal{B}_2$ be a bimachine realizing a function f with left and right automata $\mathcal{L}_2$ and $\mathcal{R}_2$, respectively. One can obtain a bimachine $\mathcal{B}_1$ realizing f with left and right automata $\mathcal{L}_1$ and $\mathcal{R}_1$, respectively.*

Proof. Intuitively, since $\mathcal{L}_1$ and $\mathcal{R}_1$ keep more information than $\mathcal{L}_2$ and $\mathcal{R}_2$ respectively, $\mathcal{B}_1$ can simulate the behavior of $\mathcal{B}_2$. Let $\mathcal{B}_2 = (\mathcal{L}_2, \mathcal{R}_2, i, o, t)$. We define $\mathcal{B}_1 := (\mathcal{L}_1, \mathcal{R}_1, i_1, o_1, t_1)$ by: $i_1([v]_{\mathcal{R}_1}) := i([v]_{\mathcal{R}_2})$, $o_1([u]_{\mathcal{L}_1}, a, [v]_{\mathcal{R}_1}) := o_2([u]_{\mathcal{L}_2}, a, [v]_{\mathcal{R}_2})$ and $t_1([u]_{\mathcal{L}_1}) := t([u]_{\mathcal{L}_2})$. Since the automata of $\mathcal{B}_1$ are finer than those of $\mathcal{B}_2$, then $\mathcal{B}_1$ is well defined and simulates $\mathcal{B}_2$, hence they realize the same function. □

The following lemmas explicits a trade-off between the two automata of a bimachine: the finer an automaton is, the coarser the other automaton can be.

Lemma 2.2.20. *Let $\mathcal{B}_1$ and $\mathcal{B}_2$ be bimachines realizing f with their respective right automata verifying $\mathcal{R}_1 \sqsubseteq \mathcal{R}_2$. Then $\mathcal{L}_f(\mathcal{R}_1) \sqsupseteq \mathcal{L}_f(\mathcal{R}_2)$.*

Proof. Let $\mathcal{B}_1$ and $\mathcal{B}_2$ be bimachines realizing f with their respective right automata verifying $\mathcal{R}_1 \sqsubseteq \mathcal{R}_2$. Let us consider $B_f(\mathcal{R}_2)$ which has automata $\mathcal{L}_f(\mathcal{R}_2)$ and $\mathcal{R}_2$. Since $\mathcal{R}_1 \sqsubseteq \mathcal{R}_2$, according to Lem. 2.2.19, we can obtain a bimachine realizing f with automata $\mathcal{L}_f(\mathcal{R}_2)$ and $\mathcal{R}_1$. By minimality of $\mathcal{L}_f(\mathcal{R}_1)$, we have according to Th. 2.2.11 that $\mathcal{L}_f(\mathcal{R}_2) \sqsubseteq \mathcal{L}_f(\mathcal{R}_1)$. □

Remark 2.2.21. The previous lemma explicitly tells us that for a given function, there is a trade-off between the coarseness of the left and the right automata. The finer the left automaton is, the coarser the right automaton can be, and *vice versa.* Intuitively this means that the more information the one automaton gives the less the other is required to give. In fancy terms we have, for any given function, a Galois connection between the set of minimal left automata and the set of minimal right automata (with partial orders $\sqsubseteq$).

Theorem 2.2.22. *Let $\mathcal{B}$ be a bimachine with right automaton $\mathcal{R}$ realizing a function f. Then $\mathcal{B}_f(\mathcal{L}_f(\mathcal{R}))$ is minimal.*

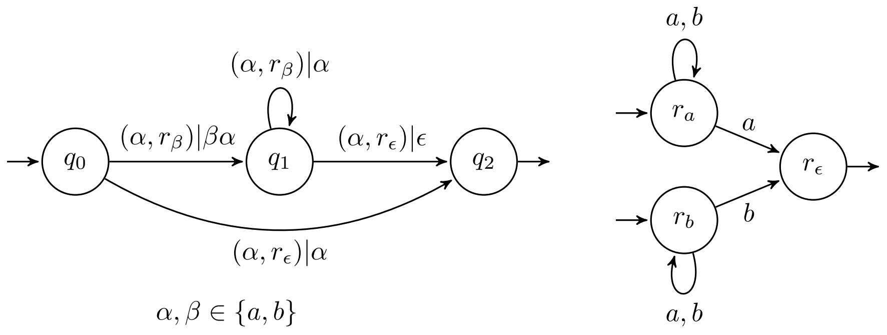
Proof. Let us first remark that minimal bimachines are exactly the bimachines whose automata are unchanged (up to isomorphism) by both left minimization and right minimization. Since left and right minimization are idempotent operators, we only have to show that $\mathcal{B}_f(\mathcal{L}_f(\mathcal{R}))$ is a fixpoint for left minimization. Let $\mathcal{L} = \mathcal{L}_f(\mathcal{R}_f(\mathcal{L}_f(\mathcal{R})))$, we want to show that $\mathcal{L} = \mathcal{L}_f(\mathcal{R})$. By Th. 2.2.11 we have $\mathcal{L}_f(\mathcal{R}) \sqsubseteq \mathcal{L}$. Considering $\mathcal{B}$ and $\mathcal{B}_f(\mathcal{L}_f(\mathcal{R}))$, with $\mathcal{R} \sqsubseteq \mathcal{R}_f(\mathcal{L}_f(\mathcal{R}))$ we obtain from Lem. 2.2.20 that $\mathcal{L} \sqsubseteq \mathcal{L}_f(\mathcal{R})$. □

2.2.3 Look-ahead versus labeling

Here we link the notion of bimachines with Th. 1.5.3 (from [EM65]), which says that any rational function is the composition of a left and a right sequential function.

Labeling Let $f : A^* \to B^*$ be a function and let $\mathcal{R}$ be a right automaton recognizing $\mathrm{dom}(f)$. We define the *labeling function associated with* $\mathcal{R}$ by the right sequential transducer $\ell(\mathcal{R}) := (\mathcal{R}, i, o, t)$ with $i : r \mapsto \epsilon$, $t : [\epsilon]_{\mathcal{R}} \mapsto \epsilon$ and $o([au]_{\mathcal{R}}, a, [u]_{\mathcal{R}}) := (a, [u]_{\mathcal{R}})$. We define the transduction $f_{\mathcal{R}} := f \circ [\![\ell(\mathcal{R})]\!]^{-1}$. Note that $f_{\mathcal{R}}$ is a function, since $\ell(\mathcal{R})$ is injective (by unambiguity of $\mathcal{R}$). Intuitively, $f_{\mathcal{R}}$ produces the same output as f over an input annotated by the look-ahead information given by $\mathcal{R}$.

Our goal is to link the sequentiality of $f_{\mathcal{R}}$ with the fact that $\mathcal{R}$ gives enough look-ahead to have a bimachine with $\mathcal{R}$ as a right automaton. We begin with the easy direction.



$(\alpha, r_\beta)|\alpha$
$(\alpha, r_\beta)|\beta\alpha$
$(\alpha, r_\epsilon)|\epsilon$
$(\alpha, r_\epsilon)|\alpha$
q_0
q_1
q_2
$\alpha, \beta \in \{a, b\}$
a, b
r_a
a
r_ϵ
r_b
b
a, b

Figure 2.6: Transducer realizing $f_{\mathrm{last},\mathcal{R}}$ and right automaton $\mathcal{R}$.

Proposition 2.2.23. *Let f be a transduction and let $\mathcal{R}$ be a right automaton recognizing* $\mathrm{dom}(f)$*. If $\sim_f^{\mathcal{R}}$ has finite index then $f_{\mathcal{R}}$ is sequential.*

Proof. Let f be such that $\sim_f^{\mathcal{R}}$ has finite index. Then according to Cor. 2.2.12 there exists a bimachine $\mathcal{B} = (\mathcal{L}, \mathcal{R}, i, o, t)$ with $\mathcal{L} = (Q_{\mathcal{L}}, \Delta_{\mathcal{L}}, l_0, F_{\mathcal{L}})$ realizing f. Let us consider the sequential transducer $\mathcal{T} := (\mathcal{A}, i', o', t')$ with $\mathcal{A} := (Q, \Delta, I, F)$ defined by:

- $Q := \{q_0\} \uplus Q_{\mathcal{L}} \times Q_{\mathcal{R}}$
- $\Delta := \begin{array}{l} \{((l, r'), (a, r), (l', r)) \mid \ (l, a, l') \in \Delta_{\mathcal{L}},\ (r', a, r) \in \Delta_{\mathcal{R}}\} \\ \cup \{(q_0, (a, [u]_{\mathcal{R}}), (l', [u]_{\mathcal{R}})) \mid \ (l_0, a, l') \in \Delta_{\mathcal{L}},\ au \in \mathrm{dom}(f)\} \end{array}$
- $I := \{q_0\}$
- $F := F_{\mathcal{L}} \times \{r_0\}$ (or $F_{\mathcal{L}} \times \{r_0\} \cup \{q_0\}$ if $\epsilon \in \mathrm{dom}(f)$)
- $i'(q_0) = \epsilon$
- $o'((l, r'), (a, r), (l', r)) := o(l, a, r)$ and $o'(q_0, (a, [u]_{\mathcal{R}}), (l', [u]_{\mathcal{R}})) := i([au]_{\mathcal{R}})o(l_0, a, [u]_{\mathcal{R}})$
- $t'(l) := t(l)$ (and $t'(q_0) := i(r_0)t(l_0)$ if $\epsilon \in \mathrm{dom}(f)$)

□

Example 2.2.24. Let f_{last} be the function defined by $f(u\alpha) = \alpha u$, for $\alpha \in A = \{a, b\}$ and $u \in A^*$. We consider the right automaton $\mathcal{R}$ given in Fig. 2.6 and give a transducer realizing $f_{\mathrm{last},\mathcal{R}}$ in the same figure. Notice that f_{last} is not sequential while $f_{\mathrm{last},\mathcal{R}}$ is.

We can now consider the converse result, which is harder to show. We give a constructive proof of the result by exhibiting when possible a bimachine with $\mathcal{R}$ as a right automaton. As a side result, we show that this construction preserves aperiodicity.

Lemma 2.2.25. *Let $\mathcal{T}$ be a transducer realizing a function f and let $\mathcal{R}$ be a right automaton recognizing* $\mathrm{dom}(f)$*. If $f_{\mathcal{R}}$ is sequential then $\sim_f^{\mathcal{R}}$ has finite index. In that case one can compute $\mathcal{B}_f(\mathcal{R})$ in 2*ExpTime*. Furthermore if $\mathcal{T}$ and $\mathcal{R}$ are aperiodic then $\mathcal{B}_f(\mathcal{R})$ also is.*

Proof. Let $\mathcal{T} = (\mathcal{A}, i, o, t)$ with $\mathcal{A} = (Q, \Delta, I, F)$ be a transducer realizing a function f, let $\mathcal{R} = (Q_\mathcal{R}, \Delta_\mathcal{R}, I_\mathcal{R}, \{r_0\})$ be a right automaton recognizing $\mathrm{dom}(f)$ and let us assume that $f_\mathcal{R}$ is sequential. Let us describe the steps of the proof. Step 1) we construct a transducer realizing $f_\mathcal{R}$ by taking the product of $\mathcal{T}$ and $\mathcal{R}$. Step 2) we determinize this transducer using the powerset construction with delays of [BC02], in ExpTime. We thus obtain $\mathcal{S}$ a sequential transducer realizing $f_\mathcal{R}$, of exponential size with respect to $\mathcal{T}$. Step 3) we project the input alphabet of $\mathcal{S}$, which is $A \times Q_\mathcal{R}$, to A, to obtain a transducer realizing f. Step 4) we consider the product automaton of the underlying automaton of this transducer with $\mathcal{R}$, and determinize it, using the usual powerset construction, again with an ExpTime complexity to obtain a deterministic automaton $\mathcal{D}$. We are then able to exhibit a bimachine with $\mathcal{D}$ and $\mathcal{R}$ as automata realizing f. Finally we left minimize this bimachine to obtain $\mathcal{B}_f(\mathcal{R})$ (recall that $\sim_f^\mathcal{R} = \sim_{\mathcal{L}_f}$). We also show that each of the four steps preserves aperiodicity.

1) From $\mathcal{T}$ and $\mathcal{R}$ we can obtain a transducer realizing $f_\mathcal{R}$, by just taking $\mathcal{T}$ and using the states of $\mathcal{R}$ to label the input. Let $\mathcal{T}' := (\mathcal{A}', i', o', t')$ with $\mathcal{A}' := (Q', \Delta', I', F')$ be defined by:

- $Q' := Q \times Q_\mathcal{R}$
- $\Delta' := \{((p,s),(a,r),(q,r)) \mid \ (p,a,q) \in \Delta,\ (s,a,r) \in \Delta_\mathcal{R}\}$
- $I' := I \times I_\mathcal{R}$
- $F' := F \times \{r_0\}$
- $i'(q,r) := i(q)$
- $o'((p,s),(a,r),(q,r)) := o(p,a,q)$
- $t'(q,r) := t(q)$

By construction $\mathcal{T}'$ realizes $f_\mathcal{R}$. Let us assume that $\mathcal{A}$ and $\mathcal{R}$ are aperiodic, and let us show that $\mathcal{A}'$ then must be aperiodic also. Let $u \in (A \times Q_\mathcal{R})^*$ be a word and let $\overline{u}$ denote the projection of u onto the alphabet A. Let p, q be states of $\mathcal{A}$ and let $v \in A^*$. We assume that we have for some integer n, $(p, [\overline{u}^n v]_\mathcal{R}) \xrightarrow{u^n} (q, [v]_\mathcal{R})$. By construction of $\mathcal{A}'$, we know that the last letter of u is $(a, [v]_\mathcal{R})$ for some $a \in A$. If $n > 1$, then from the definition of $\mathcal{A}'$, we know that we must have $[\overline{u}x] = [x]$. By aperiodicity of $\mathcal{A}$ we know that $p = p_0 \xrightarrow{\overline{u}} p_1 \dots p_n \xrightarrow{\overline{u}} p_{n+1} = q$, for n large enough. We have for any $0 \leq k \leq n$ that $(p_k, [v]_\mathcal{R}) \xrightarrow{u} (p_{k+1}, [v]_\mathcal{R})$. Hence we can conclude that $(p, [v]_\mathcal{R}) \xrightarrow{u^{n+1}} (q, [v]_\mathcal{R})$ which means that $\mathcal{A}'$ is aperiodic.

2) We use the powerset construction of [BC02] on $\mathcal{T}'$, presented in Sec. 2.1, to obtain a sequential transducer $\mathcal{S}$ realizing $f_\mathcal{R}$ in ExpTime, since $f_\mathcal{R}$ is sequential by assumption. According to Lem. 2.1.15, this procedure preserves aperiodicity.

3) We project away the labels of the input letters of $\mathcal{S}$ to obtain a transducer realizing f. Furthermore this transducer is unambiguous, otherwise some word would have two distinct labelings. Let us show that this procedure preserves aperiodicity. Let $A_\mathcal{S}$ be the deterministic underlying automaton of $\mathcal{S}$, which we assume to be aperiodic. We consider $\overline{\mathcal{A}_\mathcal{S}}$, the same automaton but with the input labels projected onto A. Let $u \in A^*$, let p, q be states of $\overline{\mathcal{A}_\mathcal{S}}$, let n be a positive integer such that $p \xrightarrow{u^n}_{\overline{\mathcal{A}_\mathcal{S}}} q$. Let $v \in A^*$ be a word such that there is a final run of $\overline{\mathcal{A}_\mathcal{S}}$ over v from q. This means that there is a labeling z of u^n, *i.e.* $\overline{z} = u^n$, ending in $(a, [v]_\mathcal{R})$

for some $a \in A$, such that: $p \xrightarrow{z}_{\mathcal{A_S}} q$. Since $\mathcal{R}$ is aperiodic, we know that for m large enough, $u^m v \approx_{\mathcal{R}} u^{m+1}v$. Let y be the labeling of u with last letter $(a, [u^m v]_{\mathcal{R}})$, then $z = y^k z'$ with z' a labeling of u^m. By aperiodicity of $\mathcal{A_S}$, if k is large enough we have a run $p \xrightarrow{y^{k+1}z'}_{\mathcal{A_S}} q$. Since $\overline{y^{k+1}z'} = u^{n+1}$, we finally have $p \xrightarrow{u^{n+1}}_{\overline{\mathcal{A_S}}} q$, which concludes the proof.

4) Let us consider the product of automata $\overline{\mathcal{A}}_{\mathcal{S}}$ and $\mathcal{R}$, and let $\mathcal{D}$ be the automaton obtained by the usual powerset construction of automata. This construction preserves aperiodicity, and is in ExpTime. We only have left to show that we can obtain a bimachine realizing f with $\mathcal{D}$ and $\mathcal{R}$ as automata. Indeed, if it is the case and $\mathcal{D}$ is aperiodic, we have according to Th. 2.2.11 that $\mathcal{D} \sqsubseteq \mathcal{L}_f(\mathcal{R})$ which means that $\mathcal{B}$ is aperiodic *a fortiori*.

Let $\mathcal{S} = (\mathcal{A_S}, i_{\mathcal{S}}, o_{\mathcal{S}}, t_{\mathcal{S}})$ with $\mathcal{A_S} = (Q_{\mathcal{S}}, \Delta_{\mathcal{S}}, \{q_0\}, F_{\mathcal{S}})$. Let us define the output functions of $\mathcal{B} := (\mathcal{D}, \mathcal{R}, i_{\mathcal{B}}, o_{\mathcal{B}}, t_{\mathcal{B}})$. Let $i([u]_{\mathcal{R}}) := i_{\mathcal{S}}(q_0)$. Let $P := \{(p_1, [v_1]_{\mathcal{R}}), \dots, (p_n, [v_n]_{\mathcal{R}})\}$ be a state of $\mathcal{D}$, let $a \in A$ and let $v \in A^*$, we define $o_{\mathcal{B}}(P, a, [v]_{\mathcal{R}}) := o_{\mathcal{S}}(p_i, (a, [v]), q)$ such that $v_i \approx_{\mathcal{R}} av$ and $(p_i, (a, [v]_{\mathcal{R}}), q) \in \delta_{\mathcal{S}}$. Let us show that $o_{\mathcal{B}}$ is well-defined. Let $i, j \in \mathbf{n}$, such that $v_i = v_j = av$. This means that there exists a word which can reach both p_i and p_j in $\overline{\mathcal{A}}_{\mathcal{S}}$ and that there is a final run over v from both p_i and p_j. Since $\mathcal{A_S}$ is unambiguous, we have $p_i = p_j$. Furthermore, since $\mathcal{A_S}$ is deterministic, the state q is uniquely defined. Let $t(P) := t_{\mathcal{S}}(p_i)$ such that p_i is final. Again, by the unambiguous nature of $\overline{\mathcal{A}}_{\mathcal{S}}$, this state is uniquely defined. One can easily see that the outputs of $\mathcal{B}$ exactly match those of $\mathcal{S}$, which means that $\mathcal{B}$ realizes f. □

We now obtain the result linking the existence of a bimachine realizing a function f with right automaton $\mathcal{R}$, with the sequentiality of $f_{\mathcal{R}}$.

Theorem 2.2.26. *Let f be a transduction and let $\mathcal{R}$ be a right automaton recognizing* $\mathrm{dom}(f)$*. There exists a bimachine realizing f with right automaton $\mathcal{R}$ if and only if $f_{\mathcal{R}}$ is sequential.*

Proof. This is a consequence of Prop. 2.2.23 and Lem. 2.2.25. □

2.2.4 Canonical bimachine and characterization

Here we define the second key notion from [RS91], the delay congruence of a function. This congruence yields the coarsest look-ahead possible to realize the function sequentially. Combining this with the previously introduced minimization, the authors of [RS91] obtained a canonical bimachine for rational functions. Refining their method, we show that the number of minimal bimachines of a rational function is finite, which was conjectured in [RS91]. Since we bound the size of all minimal bimachines, we obtain an algorithm to decide **C**-rationality (for a decidable **C**).

Delay congruence

Definition 2.2.27. Let $f : A^* \to B^*$ be a function, the *left delay congruence* of f is defined for $u, v \in A^*$ by $u \overset{\vartriangle}{\approx}_f v$ if:

- $\mathrm{dom}(f)u^{-1} = \mathrm{dom}(f)v^{-1}$
- $\sup\{\|f(wu), f(wv)\|_{\mathsf{del}} \mid wu \in \mathrm{dom}(f)\} < \infty$ (recall that $\|x, y\|_{\mathsf{del}} = |x| + |y| - 2|x \wedge y|$)

Furthermore, let $\mathcal{R}_f$ denote the right automaton with left transition congruence $\overset{\vartriangle}{\approx}_f$ and recognizing $\mathrm{dom}(f)$.

First we show that this left congruence is coarser than any left transition congruence of a machine realizing a given function.

Theorem 2.2.28. *Let $\mathcal{T}$ (resp. $\mathcal{B}$) be a transducer (resp. bimachine) with underlying automaton $\mathcal{A}$ (resp. right automaton $\mathcal{R}$) realizing a function f. Then $\approx_{\mathcal{A}} \sqsubseteq \overset{\vartriangle}{\approx}_f$ (resp. $\approx_{\mathcal{R}} \sqsubseteq \overset{\vartriangle}{\approx}_f$).*

Proof. We give the proof for a bimachine, the proof for a transducer can be obtained as a consequence of Th. 2.2.15. Let $\mathcal{B} = (\mathcal{L}, \mathcal{R}, i, o, t)$ be a bimachine and let $u \approx_{\mathcal{R}} v$, we want to show that $u \overset{\vartriangle}{\approx}_f v$. Since $\approx_{\mathcal{R}}$ recognizes $\mathrm{dom}(f)$ we have $\mathrm{dom}(f)u^{-1} = \mathrm{dom}(f)v^{-1}$.

Let $w \in \mathrm{dom}(f)u^{-1}$, let us bound the size of $del(f(wu), f(wv))$. We have that $f(wu) = i([wu]_{\mathcal{R}})o([\epsilon]_{\mathcal{L}}, w, [u]_{\mathcal{R}})o([w]_{\mathcal{L}}, u, [\epsilon]_{\mathcal{R}})t([wu]_{\mathcal{L}})$ and since $u \approx_{\mathcal{R}} v$ we also have that $f(wv) = i([wu]_{\mathcal{R}})o([\epsilon]_{\mathcal{L}}, w, [u]_{\mathcal{R}})o([w]_{\mathcal{L}}, v, [\epsilon]_{\mathcal{R}})t([wv]_{\mathcal{L}})$. If we remove the longest common prefix, we obtain that $del(f(wu), f(wv)) \leq |o([w]_{\mathcal{L}}, u, [\epsilon]_{\mathcal{R}})t([wu]_{\mathcal{L}})| + |o([w]_{\mathcal{L}}, v, [\epsilon]_{\mathcal{R}})t([wv]_{\mathcal{L}})| \leq k(|u| + |v| + 2)$ where k is the maximum size of words in the range of i, o, t. Finally we obtain $\sup_{w \in A^*} \|f(wu), f(wv)\|_{\mathsf{del}} \leq k(|u| + |v| + 2) < \infty$ which means that $u \overset{\vartriangle}{\approx}_f v$. $\square$

We now show how to compute the right automaton $\mathcal{R}_f$.

Theorem 2.2.29. *Let $\mathcal{B}$ be a bimachine realizing a function f. One can compute $\mathcal{R}_f$ in* PTime.

Proof. Let $\mathcal{B} =$ be a bimachine with left and right automata $\mathcal{L}$ and $\mathcal{R}$, respectively. According to Th. 2.2.28, we know that $\mathcal{R} \sqsubseteq \mathcal{R}_f$. Let $[u_1]_{\mathcal{R}}, [u_2]_{\mathcal{R}}$ be two states of $\mathcal{R}$, we can choose u_1, u_2 of size linear in $Q_{\mathcal{R}}$. Thus we only have to show how to decide if $u_1 \overset{\vartriangle}{\approx}_f u_2$ in PTime. First we can easily check that $\mathrm{dom}(f)u_1^{-1} = \mathrm{dom}(f)u_2^{-1}$. Now we want to check whether we have $\sup_{w \in A^*} \|f(wu_1), f(wu_2)\|_{\mathsf{del}} < \infty$. From $\mathcal{B}$, for any word $x \in A^*$ we can easily construct a transducer $\mathcal{T}_x$ which realizes the function $f_x : w \mapsto f(wx)$. We can take the transducer $\mathcal{T}$ obtained from $\mathcal{B}$ by the construction of Prop. 2.2.3, we set as final states the states from which $\mathcal{T}$ has a final run over x, and we define the final output function to recover the missing output of x. Thus deciding whether $u_1 \overset{\vartriangle}{\approx}_f u_2$ amounts to deciding if $\|f_{u_1}, f_{u_2}\|_{\mathsf{del}} := \sup_{w \in A^*} \|f_{u_1}(w), f_{u_2}(w)\|_{\mathsf{del}} < \infty$.

We define a twinning-like property which will be equivalent to $\|f_{u_1}, f_{u_2}\|_{\mathsf{del}} < \infty$. This notion is equivalent to that of *adjacent functions* defined in [RS91]. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two transducers with the same domain. We say that $\mathcal{T}_1$ and $\mathcal{T}_2$ satisfy the *adjacency twinning property* if for any pair of initial runs $p_1 \xrightarrow{x|\alpha_1}_{\mathcal{T}_1} q_1 \xrightarrow{y|\beta_1}_{\mathcal{T}_1} q_1$ and $p_2 \xrightarrow{x|\alpha_2}_{\mathcal{T}_2} q_2 \xrightarrow{y|\beta_2}_{\mathcal{T}_2} q_2$ such that there is a word z with a final run from both q_1 and q_2 we have: $\mathsf{del}(i(p_1)\alpha_1, i(p_2)\alpha_2) = \mathsf{del}(i(p_1)\alpha_1\beta_1, i(p_2)\alpha_2\beta_2)$

Claim: $\|[\![\mathcal{T}_1]\!], [\![\mathcal{T}_2]\!]\|_{\mathsf{del}} < \infty$ if and only if $\mathcal{T}_1$ and $\mathcal{T}_2$ satisfy the adjacency twinning property.

Let us show this claim. Assume that $\mathcal{T}_1$ and $\mathcal{T}_2$ do not satisfy the adjacency twinning property. Then let $p_1 \xrightarrow{x|\alpha_1}_{\mathcal{T}_1} q_1 \xrightarrow{y|\beta_1}_{\mathcal{T}_1} q_1$ and $p_2 \xrightarrow{x|\alpha_2}_{\mathcal{T}_2} q_2 \xrightarrow{y|\beta_2}_{\mathcal{T}_2} q_2$ be two initial runs such that there is a word z with a final run from both q_1 and q_2 and: $\mathsf{del}(i(p_1)\alpha_1, i(p_2)\alpha_2) \neq \mathsf{del}(i(p_1)\alpha_1\beta_1, i(p_2)\alpha_2\beta_2)$. Then we obtain that for any integer n, $\|f_{u_1}(xy^nz), f_{u_2}(xy^nz)\|_{\mathsf{del}} < \|f_{u_1}(xy^{n+1}z), f_{u_2}(xy^{n+1}z)\|_{\mathsf{del}}$ and hence $\|f_{u_1}, f_{u_2}\|_{\mathsf{del}} = \infty$. Conversely, let us assume that T_1 and T_2 satisfy the adjacency twinning property. This basically means that over synchronized loops, the delay between $\mathcal{T}_1$ and $\mathcal{T}_2$ cannot increase. For any word, one can decompose its runs over $\mathcal{T}_1$ and T_2 into small parts of bounded size and a bounded number of synchronized loops. Overall, the delay must stay bounded.

Now we only have left to show that one can decide the adjacency twinning property in PTime. To do this we use the result of [FMR18] which allows to state properties of transducer and check them in PTime (actually NLogSpace), for a fixed property. The formalism used is called *pattern logic* and allows to quantify over runs of a machine and express constraints on these runs, their input and their output. Using the syntax of the article and considering the union of the two transducers, the adjacency twinning property is expressed by (the negation of):

$$\begin{gathered}\exists \pi_1 : p_1 \xrightarrow{x|\alpha_1} q_1, \exists \pi_1' : q_1 \xrightarrow{y|\beta_1} q_1, \exists \pi_1'' : q_1 \xrightarrow{z|\gamma_1} r_1, \\ \exists \pi_2 : p_1 \xrightarrow{x|\alpha_2} q_2, \exists \pi_2' : q_2 \xrightarrow{y|\beta_2} q_2, \exists \pi_1'' : q_1 \xrightarrow{z|\gamma_2} r_2, \\ \mathsf{init}(p_1) \wedge \mathsf{init}(p_2) \wedge \mathsf{final}(r_1) \wedge \mathsf{final}(r_1) \wedge \mathrm{SDel}_{\neq}(\alpha_1, \beta_1, \alpha_2, \beta_2)\end{gathered}$$

Note that the only difference with the twinning property is that the word z has to be the same for the two runs. Thus we have a PTime algorithm, which concludes the proof. □

Canonical bimachine We now show that $\mathcal{R}_f$ always gives a sufficient look-ahead to realize the function f, by exhibiting the canonical bimachine of [RS91]. The bimachine $\mathcal{B}_f(\mathcal{R}_f)$ is called the *canonical bimachine* of f.

Theorem 2.2.30. *Let f be a rational transduction, then $\mathcal{B}_f(\mathcal{R}_f)$ is finite.*

Proof. Using Lem. 2.2.25, we only have to show that $f_{\mathcal{R}_f}$ is sequential. For this we use the twinning property of [Cho77]. A transducer $\mathcal{T}$ (assumed trim) is said to satisfy the *twinning property* if for any words $u, v \in A^*$, for any initial runs $p_1 \xrightarrow{u|\alpha_1} q_1 \xrightarrow{v|\beta_1} q_1$ and $p_2 \xrightarrow{u|\alpha_2} q_2 \xrightarrow{v|\beta_2} q_2$, we have $\mathsf{del}(i(p_1)\alpha_1, i(p_2)\alpha_2) = \mathsf{del}(i(p_1)\alpha_1\beta_1, i(p_2)\alpha_2\beta_2)$. According to [Cho77, Proposition 3.4], a transducer realizes a sequential sequential function if and only if it satisfies the twinning property. Let $\mathcal{T}$ be a trim transducer realizing $f_{\mathcal{R}_f}$, and let us assume that $\mathcal{T}$ does not satisfy the twinning property. Let $u, v \in (A \times Q_{\mathcal{R}})^*$ be annotated words and let $p_1 \xrightarrow{u|\alpha_1} q_1 \xrightarrow{v|\beta_1} q_1$ and $p_2 \xrightarrow{u|\alpha_2} q_2 \xrightarrow{v|\beta_2} q_2$ be initial runs such that we have $\mathsf{del}(i(p_1)\alpha_1, i(p_2)\alpha_2) \neq \mathsf{del}(i(p_1)\alpha_1\beta_1, i(p_2)\alpha_2\beta_2)$. Let $(a, [w]_{\mathcal{R}_f}) \in A \times Q_{\mathcal{R}_f}$ be the last letter of v. Let x_1, x_2 be two words over which there is a final run from q_1 and q_2, respectively. Then, since $uvx_1, uvx_2 \in \mathrm{dom}(f_{\mathcal{R}_f})$ we must have $\overline{x_1} \approx_{\mathcal{R}_f} \overline{w} \approx_{\mathcal{R}_f} \overline{x_2}$. Hence we obtain that for any n, $\|f_{\mathcal{R}_f}(uv^n x_1), f_{\mathcal{R}_f}(uv^n x_2)\|_{\mathsf{del}} < \|f_{\mathcal{R}_f}(uv^{n+1}x_1), f_{\mathcal{R}_f}(uv^{n+1}x_2)\|_{\mathsf{del}}$. Finally we have $\sup_{n\in\mathbb{N}} \|f(\overline{uv^n x_1}), f(\overline{uv^n x_2})\|_{\mathsf{del}} = \infty$ which is in contradiction with $\overline{x_1} \approx_{\mathcal{R}_f} \overline{x_2}$, and $f_{\mathcal{R}_f}$ is sequential. □

From this theorem we are now able to show that $\mathcal{R}_f$ is *the* minimal look-ahead automaton needed to realize the function f.

Theorem 2.2.31. *Let f be a function and let $\mathcal{R}$ be a right automaton recognizing* $\mathrm{dom}(f)$. *There exists a bimachine with $\mathcal{R}$ as right automaton if and only if $\mathcal{R} \sqsubseteq \mathcal{R}_f$.*

Proof. Let f be a transduction realized by a bimachine with right automaton $\mathcal{R}$. According to Th. 2.2.28, we have $\mathcal{R} \sqsubseteq \mathcal{R}_f$. Let $\mathcal{R}$ be a right automaton recognizing $\mathrm{dom}(f)$ such that $\mathcal{R} \sqsubseteq \mathcal{R}_f$. Then since $\mathcal{B}_f(\mathcal{R}_f)$ is a bimachine realizing f, we can obtain from Lem. 2.2.19 a bimachine realizing f with $\mathcal{R}$ as right automaton. □

However the canonical bimachine does not capture all the algebraic properties of a transduction in general.

Proposition 2.2.32. *There exists a class of congruences $\mathbf{C}$ and a function f such that f is a $\mathbf{C}$-transduction and the canonical bimachine of f is not a $\mathbf{C}$-bimachine.*

Proof. Let $\mathbf{C} = \mathbf{I}$. The function f_{idem} is sequential and has a total domain, thus its minimal right automaton is trivial. This means that the left automaton of its canonical bimachine is simply the underlying automaton of its minimal sequential transducer, given in Fig. 2.2 on page 30. This automaton is not idempotent, however the bimachine of Fig. 2.5 on page 43 is idempotent and realizes f_{idem}. $\square$

Example 2.2.33. If we consider the function f_{swap}, then its canonical bimachine is actually the bimachine given in Ex. 2.2.1. One can also show that it is the only minimal bimachine of the function since $\mathcal{L}_{f_{\text{swap}}} = \mathcal{L}_{f_{\text{swap}}}(\mathcal{R}_{f_{\text{swap}}})$. Since this bimachine is not a **J**-bimachine, this means that f_{swap} is not in **J**.

Bounding minimal bimachines As we have seen, looking at the canonical bimachine is not enough to decide **C**-rationality in general, and we might need to consider all minimal bimachines. Here we show that we can bound the size of all minimal bimachines realizing a given function f. We start by comparing the size of a minimal bimachine with respect to the canonical bimachine.

Theorem 2.2.34. *Let $\mathcal{B}$ be a minimal bimachine with left automaton $\mathcal{L}$ realizing a function f. Then $\mathcal{L}_f(\mathcal{R}_f) \sqsubseteq \mathcal{L}$.*

Proof. Intuitively, since $\mathcal{R}_f$ is coarser than any right automaton of a bimachine realizing f, then $\mathcal{L}_f(\mathcal{R}_f)$ is the finest minimal left automaton on could need to realize f. Let $\mathcal{B}$ be a minimal bimachine with automata $\mathcal{L}$ and $\mathcal{R}$ realizing a function f. By minimality of $\mathcal{B}$, we know that $\mathcal{L}_f(\mathcal{R}) = \mathcal{L}$. According to Th. 2.2.28, we have $\mathcal{R} \sqsubseteq \mathcal{R}_f$ and from Lem. 2.2.20 we thus obtain that $\mathcal{L} = \mathcal{L}_f(\mathcal{R}) \sqsupseteq \mathcal{L}_f(\mathcal{R}_f)$. $\square$

In particular, we obtain a bound on the size of the minimal bimachines.

Theorem 2.2.35. *Let $\mathcal{B}$ be a bimachine realizing a function f. Then any minimal bimachine realizing f has size at most 2-exponential in the size of $\mathcal{B}$.*

Proof. Let $\mathcal{B}$ be a bimachine realizing a function f. From Th. 2.2.28, we know that $\mathcal{R}_f$ is smaller than $\mathcal{B}$'s right automaton. From Lem. 2.2.25 we know that the size of $\mathcal{L}_f(\mathcal{R}_f)$ is at most 2-exponential in the size of $\mathcal{B}$. Symmetrically we can bound the size of $\mathcal{R}_f(\mathcal{L}_f)$ in the same way. Overall, from Th. 2.2.34, we know that the size of a minimal bimachine is at most 2-exponential in the size of $\mathcal{B}$. $\square$

Example 2.2.36. We give an example of a minimal bimachine which is exponentially larger than an equivalent bimachine. Let $A = \{a, b\}$, let $n \in \mathbb{N}$ and let $f_n(uv) := v$ for $u \in A^*$ and $v \in A^n$ and $f_n(v) = v$ if $|v| < n$. The function f_n is both sequential and right-sequential. However, while a right sequential transducer only needs $n+1$ states to realize f_n, a sequential transducer needs $2^{n+1} - 1$ states since it needs to remember the last n letters read.

Remark 2.2.37. We have shown in Ex. 2.2.36 that a minimal bimachine may be exponentially bigger than another bimachine. However, Th. 2.2.35 gives a doubly exponential upper bound. Up until now we still don't know if this upper bound is met or not.

Deciding C-rationality We now state our main result of this section, the decision of **C**-rationality for rational functions.

Theorem 2.2.38. *Let $\boldsymbol{C}$ be a decidable class of congruences such that one can decide if an automaton $\mathcal{A}$ is in $\boldsymbol{C}$ in space $K(|\mathcal{A}|)$. Then one can decide if a bimachine $\mathcal{B}$ realizes a $\boldsymbol{C}$-rational function with space $K(2^{2^{P(|\mathcal{B}|)}})$ for some polynomial P.*

Proof. Let **C** be a decidable class, and let f be a transduction given as a bimachine $\mathcal{B}$. Since the size of any minimal bimachine of f is at most 2-exponential in the size of $\mathcal{B}$ (Th. 2.2.35), one can enumerate all the minimal bimachines of f with 2-exponential space and test whether one of them is in **C** or not. By property of minimality, and since **C** is closed under coarser congruences, there exists a **C**-bimachine realizing f if and only if there exists a minimal bimachine of f in **C**. Furthermore according to Cor. 2.2.16, a transduction is a **C**-transduction if and only if it is realized by a **C**-bimachine, which concludes the proof. □

Deciding aperiodicity According to [CH91], testing aperiodicity of an automaton, and hence a bimachine is PSpace-complete. Hence with the algorithm presented in Th. 2.2.38 we can decide **A**-rationality in 2ExpSpace. However, by the robustness of **A** to determinization, we are able to improve this complexity.

Theorem 2.2.39. *A rational transduction is aperiodic if and only if its canonical bimachine is.*

Proof. If f's canonical bimachine is aperiodic then f is aperiodic. Let us assume that f is realized by an aperiodic transducer $\mathcal{T}$. Then according to Th. 2.2.28, $\mathcal{R}_f$ is aperiodic as well. Thus from Lem. 2.2.25, we obtain that the canonical bimachine of f is aperiodic. □

This tells us that instead of computing all minimal bimachines of f and test them for aperiodicity, one only needs to compute the canonical bimachine. However, since deciding aperiodicity of a bimachine is PSpace-complete, this stills yields a 2ExpSpace complexity.

Theorem 2.2.40. *A rational transduction is aperiodic if and only if all its minimal bimachines are aperiodic.*

Proof. Let us show the one non-trivial direction. Let f be an aperiodic transduction, and let $\mathcal{B}$ be a minimal bimachine of f. According to Th. 2.2.39, we know that $\mathcal{B}_f(\mathcal{R}_f)$ is aperiodic, and hence $\mathcal{L}_f(\mathcal{R}_f)$ is aperiodic. Symmetrically, we have that $\mathcal{R}_f(\mathcal{L}_f)$ is aperiodic. From Th. 2.2.34, we obtain that the left and right automata of $\mathcal{B}$ are coarser than $\mathcal{L}_f(\mathcal{R}_f)$ and $\mathcal{R}_f(\mathcal{L}_f)$, respectively, and are thus aperiodic. □

We obtain our second main result of this section:

Theorem 2.2.41. *Deciding if a bimachine realizes an aperiodic function is* PSpace-*complete.*

Proof. From [CH91] we know that the problem is PSpace-hard. Let $\mathcal{B}$ be a bimachine with right automaton $\mathcal{R}$, then from Th. 2.2.14 one can obtain the bimachine $\mathcal{B}' := \mathcal{R}_f(\mathcal{L}_f(\mathcal{R}))$ in PTime. Furthermore from Th. 2.2.22, $\mathcal{B}'$ is a minimal bimachine for f. Finally from Th. 2.2.40, we know that f is aperiodic if and only if $\mathcal{B}'$ is, which we can check in PSpace ([CH91]). □

Remark 2.2.42. We leave open the question whether one can decide the same problem in PSpace when the input is given as a transducer. Deciding whether an automaton recognizes an aperiodic language can be done in PSpace (see [DG08]), however the proof does not seem to transfer straightforwardly to transducers.

2.3 Logical transducers

The goal of this section is to establish a transfer theorem of logic-algebra equivalences from languages to functions. We start by defining, for a logical fragment F, 2-F transducers, an *ad hoc* model of logical transducers which exactly coincides with bimachines. Then we give sufficient conditions under which 2-F transducers and F are equivalent. Under some conditions on a logical fragment F equivalent to some decidable class of congruences **C**, we thus obtain an algorithm to decide F-definability of rational functions, using the result of Section 2.2. In particular we show the decidability of FO-definability of rational functions.

2.3.1 2-F transducers

We define 2-F transducers, a different formalism from F-transducers defined in Section 1.5.2, and show their equivalence with **C**-transducers, under the equivalence of F-languages and **C**-languages.

Pairs of formulas We define pairs of formulas, a logical formalism for pointed languages, where the first component talks about the prefix of the word up to the pointed position and the second component talks about the suffix from the pointed position. Let F be a logical fragment of MSO[$\leq$], we define 2-F-sentences over an alphabet A by the following grammar, where ϕ denotes an F-sentence and $a \in A$:

$$F ::= F \vee F \;\mid\; (\phi, \phi)_a$$

Let us define by induction on 2-MSO[$\leq$] sentences what it means for a pointed word to satisfy a 2-MSO[$\leq$] sentence. Let (u,i) be a pointed word over A and let $F = (\phi_1, \phi_2)_a$ be a 2-MSO[$\leq$] atomic sentence. Then we say that $(u,i) \models F$ if the following three conditions are satisfied: 1) $u(i) = a$ 2) $u(:i-1) \models \phi_1$ and 3) $u(i+1:) \models \phi_2$. Let $F := F_1 \vee F_2$, then $(u,i) \models F$ if either $(u,i) \models F_1$ or $(u,i) \models F_2$.

Example 2.3.1. We define the same pointed language as in Ex. 1.5.6, $F_{a,\text{middle}} := (\exists x\, \top, \exists x\, \top)_a$. This 2-formula states that the pointed position is labeled by a and that the prefix and suffix are not empty.

2-transducers We define an alternative formalism of logical transducers from the one defined in Section 1.5.2. Let F be a fragment of MSO[$\leq$], a *2-F transducer* over an alphabet A is a tuple $\mathcal{T} = \big(K, \phi_{\text{dom}}, (\phi_\alpha)_{\alpha \in K}\big)$ where K is a finite subset of A^*, ϕ_{dom} is an F-sentence, called the *domain formula*, and for each $\alpha \in K$, ϕ_α is a 2-F sentence. The semantics is defined exactly as for F-transducers.

2.3.2 2-F transducers and C-bimachines

Pure machines A transducer or a bimachine is called *pure* if its initial and final outputs are all equal to ϵ. A class of congruences **C** is called *pure* if for any alphabet A, $\{\epsilon\}$ is a $\mathbf{C}(A)$-language. In other words a class **C** is pure if for any alphabet A, the congruence $\equiv_\epsilon := \{(u,v) \mid\; u, v \neq \epsilon\} \cup \{(\epsilon,\epsilon)\}$ is in $\mathbf{C}(A)$.

Proposition 2.3.2. *Let* **C** *be a pure class of congruences, then any* **C***-transduction can be realized by a pure* **C***-bimachine.*

Proof. Let **C** be a pure class. Let $\mathcal{B} = (\mathcal{L}, \mathcal{R}, i, o, t)$ be a bimachine realizing a function f, with **C**-automata $\mathcal{L} := (Q_\mathcal{L}, \Delta_\mathcal{L}, \{l_0\}, F_\mathcal{L})$ and $\mathcal{R} := (Q_\mathcal{R}, \Delta_\mathcal{R}, I_\mathcal{R}, \{r_0\})$. We define $\mathcal{L}'$ by taking $\sim_{\mathcal{L}'} := \sim_\mathcal{L} \sqcap \equiv_\epsilon$ and we define $\mathcal{R}'$ the same way. Since **C** is pure, and is closed under intersection, we have that $\mathcal{L}'$ and $\mathcal{R}'$ are still **C**-automata. We define a new **C**-bimachine $\mathcal{B}' := (\mathcal{L}', \mathcal{R}', i', o', t')$ by $i' : r \mapsto \epsilon$, $t' : l \mapsto \epsilon$. Let $u, v \in A^+$ be non-empty words and let $a \in A$ be a letter, we define the new output function by:

- $o'([\epsilon]_{\mathcal{L}'}, a, [\epsilon]_{\mathcal{R}'}) := i([a]_{\mathcal{R}'})o([\epsilon]_{\mathcal{L}'}, a, [\epsilon]_{\mathcal{R}'})t([a]_{\mathcal{L}'})$
- $o'([\epsilon]_{\mathcal{L}'}, a, [v]_{\mathcal{R}'}) := i([av]_{\mathcal{R}'})o([\epsilon]_{\mathcal{L}'}, a, [v]_{\mathcal{R}'})$
- $o'([u]_{\mathcal{L}'}, a, [\epsilon]_{\mathcal{R}'}) := o([u]_{\mathcal{L}'}, a, [\epsilon]_{\mathcal{R}'})t([ua]_{\mathcal{L}'})$
- $o'([u]_{\mathcal{L}'}, a, [v]_{\mathcal{R}'}) := o([u]_{\mathcal{L}'}, a, [v]_{\mathcal{R}'})$

By construction the pure **C**-bimachine $\mathcal{B}'$ realizes f, which concludes the proof. □

Theorem 2.3.3. *Let **C** be a pure class of congruences equivalent to some logical fragment* F*. Then **C**-transductions are exactly the* 2-F *transductions.*

Proof. Let $\mathcal{B} = (\mathcal{L}, \mathcal{R}, i, o, t)$, be a **C**-bimachine realizing a function f. Since **C** is pure, we can assume from Prop. 2.3.2 that $\mathcal{B}$ is pure. Since $\mathcal{L}$ and $\mathcal{R}$ are **C**-automata, we have for any word $w \in A^*$ that $[w]_{\mathcal{L}}$ and $[w]_{\mathcal{R}}$ are **C**-languages. From the equivalence between **C** and F, there exist F-formulas $\phi_{[w]_{\mathcal{L}}}$ and $\phi_{[w]_{\mathcal{R}}}$ recognizing these languages, respectively. We define the 2-F transducer $\mathcal{T} := \big(K, \phi_{\text{dom}}, (\phi_\alpha)_{\alpha\in K}\big)$ by:

- $K := \{o([u]_{\mathcal{L}}, a, [v]_{\mathcal{R}}) \mid\ a \in A, u \in A^*, v \in A^*\}$
- $\phi_{\text{dom}} := \bigvee_{u\in\text{dom}(f)} \phi_{[u]_{\mathcal{L}}}$
- $\phi_\alpha := \bigvee_{o([u]_{\mathcal{L}},a,[v]_{\mathcal{R}})=\alpha} (\phi_{[u]_{\mathcal{L}}}, \phi_{[v]_{\mathcal{R}}})_a$

We thus obtain a 2-F transducer realizing f.

Let $\mathcal{T} = \big(K, \phi_{\text{dom}}, (\phi_\alpha)_{\alpha\in K}\big)$ be a 2-F transducer. Let $\alpha \in K$ and let us write the 2-F formula $\phi_\alpha = \bigvee_{i\in\mathbf{n}_\alpha} (\phi_{\alpha,i}, \psi_{\alpha,i})_{a_{\alpha,i}}$. For a formula ϕ, we define $\equiv_\phi$ the syntactic congruence of $[\![\phi]\!]$. By equivalence between **C** and F, the congruences $\equiv_{\phi_{\alpha,i}}$ and $\equiv_{\psi_{\alpha,i}}$ are in **C**. Let $\mathcal{L}$ be the left automaton recognizing dom(f) with right transition congruence $\equiv_{\phi_{\text{dom}}} \bigsqcap_{\alpha,i\in\mathbf{n}_\alpha} \equiv_{\phi_{\alpha,i}}$. Symmetrically we define $\mathcal{R}$ as the right automaton recognizing dom(f) with right transition congruence $\equiv_{\phi_{\text{dom}}} \bigsqcap_{\alpha,i\in\mathbf{n}_\alpha} \equiv_{\psi_{\alpha,i}}$. We define the pure **C**-bimachine $\mathcal{B} := (\mathcal{L}, \mathcal{R}, i, o, t)$, with $o([u]_{\mathcal{L}}, a, [v]_{\mathcal{R}}) := \alpha$ if there is $i \in \mathbf{n}_\alpha$ such that $(uav, |u| + 1) \models (\phi_{\alpha,i}, \psi_{\alpha,i})_{a_{\alpha,i}}$. By construction $\mathcal{B}$ realizes f which concludes the proof. □

2.3.3 Logic-algebra transfer result

Here we give sufficient conditions on a fragment F so that 2-F and $\mathsf{F_c}$ (the pointed version of F, see Section 1.5.2) recognize the same pointed languages. This means, according to Th. 2.3.3, that these conditions are sufficient to transfer a logic algebra equivalence from languages to functions.

Languages over a pointed alphabet A pointed word over an alphabet A can alternatively be seen as a words over an extended alphabet $A \uplus \dot{A}$. Of course not all words over this extended alphabet are pointed words, but in most logics one can define a formula to enforce the existence of a unique pointed position. A language L over the extended alphabet $A \uplus \dot{A}$ is called *pointed* if any word of L has exactly one position with a pointed letter, *i.e.* $L \subseteq A^*\dot{A}A^*$.

Example 2.3.4. Let $A = \{a, b\}$ be an alphabet, we define a formula $\phi_{\text{pointed}} := \exists x\ \dot{A}(x) \wedge \forall y\ y = x \vee A(y)$, where $A(x)$ is a macro defined as $\bigvee_{a\in A} a(x)$. We define the same language as in Ex. 1.5.6, but seen as a language over a pointed alphabet: $\phi_{a,\text{middle}} := \phi_{\text{pointed}} \wedge \exists x\ \dot{a}(x) \wedge \neg \min(x) \wedge \neg \max(x)$.

Sufficient conditions Let F be a logical fragment of $\mathsf{MSO}[\leq]$. Our goal is to show that 2-F and $\mathsf{F_c}$ coincide under assumptions (1)-(3), and assumption (4) will ensure that the class of congruences we consider is pure.

(1) $\mathsf{F_c}$-formulas over an alphabet A and F-formulas over an alphabet $A \uplus \dot{A}$ define the same pointed languages.

(2) A language over an alphabet A is definable by a F-formula over A if and only if it is definable by a F-formula over a larger alphabet $A \cup B$.

(3) F-languages are closed under *pointed concatenation*, meaning that for two F-languages L_1, L_2 over an alphabet A, the language $L_1 \sharp L_2$ is an F-language over $A \uplus \{\sharp\}$.

(4) $\{\epsilon\}$ is an F-language over some alphabet.

Remark 2.3.5. As we will see these assumptions hold for most known logical fragments which have access to the linear order predicate, and which are equivalent to some class of congruences. A co-example is the fragment of first-order logic with successor predicate, $\mathsf{FO}[+1]$, which does not satisfy property (3).

From pairs of formulas to pointed formulas and back We now show the equivalence of 2-F and F under assumptions (2) and (3).

Lemma 2.3.6. *Let F be a logical fragment equivalent to some class of congruences $\mathbf{C}$. Under assumption (2), any F-definable pointed language is 2-F definable.*

Proof. Let F be a logical fragment equivalent to some class of congruences $\mathbf{C}$ and satisfying property (2). Let ϕ be an F-formula recognizing a pointed language L over the alphabet $A \uplus \dot{A}$. Let $\mathcal{A}$ be $\mathbf{C}(A \uplus \dot{A})$-automaton recognizing L, with a transition relation Δ. Let us define

$$\mathcal{F} := \bigvee_{(p,\dot{a},q)\in\Delta} (\phi_I^p, \phi_q^F)_a$$

where ϕ_I^p and ϕ_q^F denote F-formulas recognizing the languages of words which have an initial run to p and a final run from q, respectively. We know that these languages are $\mathbf{C}(A \uplus \dot{A})$-languages since they are recognized by automata obtained from $\mathcal{A}$ by changing the initial and final states, which does not change the transition congruence. From the equivalence between F and $\mathbf{C}$, these languages are definable by F-formulas over the extended alphabet $A \uplus \dot{A}$. From assumption (2) we thus obtain that these languages can be defined by F-formulas over A. □

Lemma 2.3.7. *Let F be a logical fragment equivalent to some class of congruences $\mathbf{C}$. Under assumptions (2)-(3), any 2-F definable pointed language is F-definable.*

Proof. Let F be a logical fragment equivalent to some class $\mathbf{C}$, satisfying assumptions (2)-(3). Let $F = \bigvee_{i\in\mathbf{n}}(\phi_i, \psi_i)_{a_i}$ be a 2-F formula over an alphabet A. We define the languages $L_i := [\![\phi_i]\!]\dot{a}_i[\![\psi_i]\!]$ which are pointed concatenations of F-langugages, and are thus each F-languages over $A \uplus \{\dot{a}_i\}$, respectively, from property (3). Using (2) we have that each L_i is a pointed F-language over $A \uplus \dot{A}$. Since F is equivalent to $\mathbf{C}$, and since $\mathbf{C}$-languages are closed under boolean combinations, we have that $\bigcup_{i\in\mathbf{n}} L_i$ is a pointed F-language over $A \uplus \dot{A}$. □

By combining the two previous lemmas we obtain the correspondence between $\mathsf{F_c}$ and 2-F pointed languages.

Corollary 2.3.8. *Let F be a logical fragment equivalent to some class of congruences $\mathbf{C}$. Under assumptions (1)-(3), $\mathsf{F_c}$ and 2-F define the same pointed languages.*

Proof. Let F be a logical fragment equivalent to some class of congruences $\mathbf{C}$ satisfying (1)-(3). From Lem. 2.3.6 and 2.3.7 we know that F and 2-F define the same pointed languages. Using (1) we obtain our result. □

Lemma 2.3.9. *Let F be a logical fragment equivalent to some class of congruences $\boldsymbol{C}$. Under assumptions (2) and (4), $\boldsymbol{C}$ is pure.*

Proof. According to (4), $\{\epsilon\}$ is an F-language over some alphabet A, which means, according to (2) that $\{\epsilon\}$ is an F-language over any alphabet. By equivalence with $\mathbf{C}$, we obtain that $\mathbf{C}$ is pure. □

We are now able to state our main transfer theorem.

Theorem 2.3.10. *Let F be a logical fragment satisfying properties (1)-(4) and equivalent to a class of congruences $\boldsymbol{C}$. Then any transduction is F-definable if and only if it is $\boldsymbol{C}$-definable.*

Proof. From Lem. 2.3.9, we know that $\mathbf{C}$ is pure. From (1)-(3) and Cor. 2.3.8 we have an equivalence between 2-F transducers and F-transducers. From Th. 2.3.3 we have the equivalence between 2-F transducers and $\mathbf{C}$-bimachines. Hence we obtain the equivalence between F-transducers and $\mathbf{C}$-bimachines. □

2.3.4 Decidable fragments

We apply our transfer theorem to three well-known fragments of $\mathsf{MSO}[\leq]$, namely first-order logic $\mathsf{FO}[\leq]$, two-variable logic $\mathsf{FO}^2[\leq]$ and the boolean closure of existential first-order logic $\mathsf{B}\Sigma_1[\leq]$.

Proposition 2.3.11. *The fragments $\mathsf{FO}[\leq]$, $\mathsf{FO}^2[\leq]$ and $\mathsf{B}\Sigma_1[\leq]$ all satisfy properties (1)-(4).*

Proof. Let us show that $\mathsf{FO}^2[\leq]$ satisfies properties (1)-(4).

Let us show property (1). Let ϕ be an $\mathsf{FO}^2[\leq]$-formula defining a pointed language over the alphabet $A \uplus \dot{A}$. We define ϕ' the $\mathsf{FO}^2_\mathsf{c}[\leq]$-formula obtained by substituting $(x = \mathsf{c}) \wedge a(x)$ for each occurrence of a predicate $\dot{a}(x)$. Then the $\mathsf{FO}^2_\mathsf{c}[\leq]$-formula ϕ' recognizes the same pointed language as ϕ. Let ϕ be an $\mathsf{FO}^2_\mathsf{c}[\leq]$-formula over A, and let ϕ_{pointed} denote the $\mathsf{FO}^2[\leq]$-formula from Ex. 2.3.4 defining the language of pointed words over $A \uplus \dot{A}$. Let ϕ' be the $\mathsf{FO}^2[\leq]$-formula obtained from ϕ by the following syntactic substitutions. Each occurrence of a predicate $a(x)$ is replaced by $a(x) \vee \dot{a}(x)$, $a(\mathsf{c})$ is replaced by $\exists x\ \dot{a}(x)$, and $x < c$ is replaced by $\exists y\ \dot{A}(y) \wedge (x < y)$. Finally, the formula $\phi_{\text{pointed}} \wedge \phi'$ is in $\mathsf{FO}^2[\leq]$ and defines the same pointed language as ϕ.

Let us show (2). Let ϕ be an $\mathsf{FO}^2[\leq]$-formula over an alphabet A recognizing a language L. In particular, ϕ is an $\mathsf{FO}^2[\leq]$-formula over $A \uplus B$, but it may recognize a language larger than L in $(A \uplus B)^*$. Let $\phi_A := \forall x\ A(x)$ be the formula stating that each position is labeled by a letter of A. Then $\phi \wedge \phi_A$ defines L over $A \uplus B$. Let ϕ be an $\mathsf{FO}^2[\leq]$-formula over an alphabet $A \uplus B$ recognizing a language $L \subseteq A^*$. Let ϕ' be the formula obtained from ϕ by substituting $\perp$ for any predicate $b(x)$ for $b \in B$. Then any word is a model for ϕ over $A \uplus B$ if and only it is a model for ϕ' over A.

Let us show (3). Given a formula ϕ in the fragment $\mathsf{FO}^2[\leq]$, we define inductively the guarded $\mathsf{FO}^2_\mathsf{c}[\leq]$-formula $\phi^{<\mathsf{c}}$ by restricting every quantification to positions before the position of c. More formally, if $\phi = \exists x\ \psi(x)$, then we define $\phi^{<\mathsf{c}} := \exists x\ (x < \mathsf{c}) \wedge \psi^{<\mathsf{c}}(x)$. Boolean connectives and atomic formulas are not affected. Let L_1, L_2 be two languages over A recognized by the $\mathsf{FO}^2[\leq]$-formulas ϕ_1, ϕ_2 respectively, and let $\sharp \notin A$. We define the $\mathsf{FO}^2_\mathsf{c}[\leq]$-formula $\phi := \phi_1^{<\mathsf{c}} \wedge \sharp(\mathsf{c}) \wedge \phi_2^{<\mathsf{c}}$ which recognizes the pointed language $L_1\dot{\sharp}L_2$. Using (1), we know that this language can be recognized by an $\mathsf{FO}^2[\leq]$-formula.

Property (4) is satisfied since $\phi_{\text{empty}} := \forall x \perp$ defines the language $\{\epsilon\}$.
The same arguments work for $\mathsf{FO}[\leq]$, and adapting them slightly gives a proof for $\mathsf{B}\Sigma_1[\leq]$. □

Now we have all the ingredients to state the main result of this section.

Theorem 2.3.12. *Given a bimachine realizing a function f, one can decide if f is:*

- $\mathsf{FO}[\leq]$*-definable (in* PSPACE*)*
- $\mathsf{FO}^2[\leq]$*-definable*
- $\mathsf{B}\Sigma_1[\leq]$*-definable*

Proof. The equivalence between $\mathsf{FO}[\leq]$ and **A** is due to [Sch65, MP71]. In [TW98], the authors show the equivalence between $\mathsf{FO}^2[\leq]$ and **DA**. The equivalence between $\mathsf{B}\Sigma_1[\leq]$ and **J** is partly due to [Sim75] and can be found in [DGK08]. From Prop. 2.3.11 and Th. 2.3.10 we have that these equivalences transfer to transductions. Furthermore the equivalences described above are effective, meaning that classes of congruences **A**, **DA** and **J** are decidable. Hence from Th. 2.2.38, we obtain our result. In the particular case of $\mathsf{FO}[\leq]$, Th. 2.2.41 says that one can decide $\mathsf{FO}[\leq]$-definability in PSPACE. □

Chapter 3

Characterizations of rational functions over infinite words

"Le café est un breuvage qui fait dormir quand on n'en prend pas."

– Alphonse Allais

Rational ω-languages inherit many of the good properties of rational languages over finite words. Most of the celebrated results for rational languages over finite words have been successfully transferred to infinite words, often at some cost, if not conceptual then at least technical. Our goal is to apply the same program as for rational functions over finite words. With this plan in mind, new difficulties arise (we count four main ones) which can be classified into three categories: those that we overcome with a satisfying solution, those that are overcome but at some cost of elegance or importance and those which we have not been able to overcome yet.

Our first obstacle manifests itself already for sequential ω-functions, and even for rational ω-languages. As we mentioned in Chap. 1, the link between right congruences and deterministic automata is not as tight for ω-languages as it is in the finite case. Concretely, there is no canonical (nor indeed minimal) deterministic automaton recognizing a given ω-language. Somewhat surprisingly we are however able to circumvent this issue for sequential ω-functions. We show, in some precise way, that the problem of realizing a sequential transduction with a minimal device and the problem of recognizing the domain of the function can be considered independently, and we are thus able to minimize sequential ω-transducers.

Although finding a suitable notion of right-sequential transducer is not straightforward, Carton proposed in [Car10] a solution based on *prophetic automata* (see [CM03]). This solution seems natural and in [Car10] the author even obtained an Elgot-Mezei theorem for rational functions over infinite words.

Perhaps the biggest issue in the case of ω-functions, is the breaking of the symmetry between sequential and right sequential ω-functions. However, we have not been able to obtain a canonical way to realize a right sequential function. This problem transfers to bimachines where we are able to minimize left automata with respect to a fixed right automaton, but not the other way around. This problem does not seem easy to overcome, which gives little hope of characterizing all minimal bimachines for rational ω-functions.

Our third issue is the definition of the delay congruence which does not make sense when applied directly to infinite words. Using a slight variation, originating from [BLN12], we are

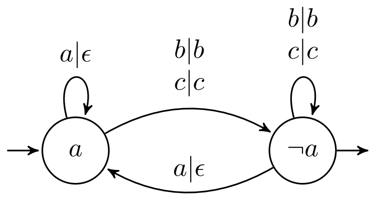


Figure 3.1: A sequential Büchi transducer realizing f_{erase}.

however able to recover this congruence which has the same minimality property as in the finite case. However, this congruence just falls short of giving enough information to realize a function sequentially, in the general case, which brings us to the fourth issue.

In order to compensate for the lack of information given by the delay congruence, we define a second canonical look-ahead congruence, which we call the *ultimate congruence*, and which does give enough information to realize any function sequentially. We thus regain a canonical bimachine for rational ω-functions. This ultimate congruence comes at some cost however, since we have not been able to show any minimality property. Obtaining a minimal look-ahead automaton would be much more satisfying (especially since we have not been able to characterize the minimal bimachines) and we are still working towards this goal.

Let us now describe the results of this section. The first is the extension of the minimization results of sequential functions (from [Cho03]) to sequential ω-functions. The second result is an Elgot-Mezei theorem for rational ω-functions, which was already half-obtained in [Car10]. Our third result is the definition of a canonical look-ahead which gives, together with a left minimization procedure, a canonical bimachine to realize a rational ω-transduction. Finally we show that this canonical bimachine is aperiodic for any aperiodic transduction which gives, as a corollary, a procedure to decide FO-definability of rational ω-transductions.

3.1 Sequential and quasi-sequential functions

Here we extend the results from [Cho03] to sequential ω-functions. Given a sequential ω-function f we define the minimal sequential transducer realizing an *extension* of f. We show that this minimal transducer is coarser than any transducer realizing f, and we show how to obtain this minimal transducer in PTime, from a sequential transducer realizing f. The main difference with the finite case is that one cannot define a minimal automaton recognizing a given rational ω-language. To circumvent this issue we extend any sequential function over the topological closure (for the prefix distance) of its domain, in a canonical way.

Another difference with the finite case is that having a syntactic right congruence of finite index is no longer sufficient for a function to be sequential, but rather characterizes a larger class of functions that we call *quasi-sequential*. Quasi-sequential functions can be thought of as the generalization of sub-sequential transductions from finite words to infinite words. A function is quasi-sequential if it can be realized by a sequential transducer extended with the possibility to append a (possibly infinite) word at the "end" of the computation. It turns out that quasi-sequential functions are exactly the functions for which the determinization procedure from [BC04] terminates. However the procedure is not always correct when the function is not sequential. We also show, just like for the finite case, that determinization preserves aperiodicity of ω-transducers.

3.1.1 Algebraic characterization of sequential functions

We define the right syntactic congruence of an ω-function. For that we need the notions of infimum and liminf of a function. As we show below, the liminf of a sequential function extends it over the topological closure of its domain. The minimality result we obtain holds for liminfs but we also show that the minimal sequential transducer for a liminf is coarser than any sequential transducer realizing the original function.

Inf and liminf Let A, B be alphabets. Let $f : A^\omega \to B^\infty$ be a partial function. We define $\widehat{f} : A^* \to B^\infty$ the *infimum* of f over $\downarrow\mathrm{dom}(f) \cap A^*$ by $\widehat{f}(u) := \bigwedge\{f(ux) \mid ux \in \mathrm{dom}(f)\}$. Intuitively, $\widehat{f}$ outputs over a word u the longest possible word, knowing that the input begins with u. We also define $\overline{f} : A^\omega \to B^\infty$ the *limit infimum* of f over $\overline{\mathrm{dom}(f)}$ by $\overline{f}(x) := \lim_{u<x} \widehat{f}(u)$.

Example 3.1.1. Let $f_{\text{erase}} : (a^*(b+c))^\omega \to (b+c)^\omega$ be the function which erases as of infinite words with an infinite number of non a letters over $A = \{a,b,c\}$ (given in Fig. 3.1 on the preceding page). $\widehat{f}_{\text{erase}} : A^* \to (b+c)^*$ is simply the morphism which erases as and leaves b and c unchanged. Thus $\overline{f}_{\text{erase}}$ is defined over $\overline{\mathrm{dom}(f)} = A^\omega$ and erases as and leaves bs and cs unchanged. The function $\overline{f}_{\text{erase}}$ can be given by the transducer of Fig. 3.1 on the facing page, by making all states final. We can see that the sequential function $\overline{f}_{\text{erase}}$ extends the sequential function f_{erase}.

Let $f_{\#a} : (a+b)^\omega \to (a+b)^\omega$ be defined by $f_{\#a}(x) = a^\omega$ if $a \in \mathrm{Inf}(x)$ and $f_{\#a}(x) = b^\omega$ otherwise. Here for any finite word u, we have $\widehat{f}_{\#a}(u) = \epsilon$, and hence for any infinite word x we have $\overline{f}_{\#a}(x) = \epsilon$, thus $\overline{f}_{\#a}$ does not extend $f_{\#a}$.

Let $A = \{a,b\}$, let $u_n \in AA^+$ for any $n \in \mathbb{N}$ and let $x = \prod_{n\in\mathbb{N}}(u_n\#)$, we define $f_{\text{blocks}} : (AA^+\#)^\omega \to (a+b+\#)^\omega$ by $f_{\text{blocks}}(x) = \prod_{n\in\mathbb{N}}(f_{\text{swap}}(u_n)\#)$. Then we obtain the infimum function $\widehat{f}_{\text{blocks}}(u_1\#\cdots\#u_n) = f_{\text{swap}}(u_1)\#\cdots\#f_{\text{swap}}(u_{n-1})\#$. Let $x \in A^\omega$, then the liminf extends f_{blocks} over words with a finite number of #s, by $\overline{f}_{\text{blocks}}(u_1\#\cdots\#u_n\#x) = f_{\text{swap}}(u_1)\#\cdots\#f_{\text{swap}}(u_{n-1})\#$.

Here we show that the liminf of a sequential function is an extension of it.

Proposition 3.1.2. *Let f be a sequential ω-function, then $\overline{f}_{|\mathrm{dom}(f)} = f$.*

Proof. Let $\mathcal{T} = (\mathcal{A}, i, o)$ be a sequential transducer realizing a function $f : A^\omega \to B^\infty$, and let $x \in \mathrm{dom}(f)$. Let us first remark that $\overline{f}(x) \leq f(x)$. If $f(x)$ is a finite word then we have the initial run $q_0 \xrightarrow{u|\alpha} p$, with $i(q_0)\alpha = f(x)$ for some $u < x$. Thus we have $f(x) \leq \widehat{f}(u) \leq \overline{f}(x) \leq f(x)$. Let us now assume that $f(x)$ is infinite and let $q_0 \xrightarrow{x(:n)|\alpha_n} p_n$ denote the initial run of the prefix of x of size $n \in \mathbb{N}$. We have that $i(q_0)\alpha_n \leq \widehat{f}(x(:n))$ for any $n \in \mathbb{N}$, and by taking the limit we obtain: $f(x) \leq \overline{f}(x)$ and thus $f(x) = \overline{f}(x)$. $\square$

Remark 3.1.3. From this we deduce that the function $f_{\#a}$ given in Ex. 3.1.1 is not sequential. Since the function f_{blocks} is not sequential, we also deduce that the above condition is necessary but not sufficient to be sequential.

Syntactic congruence

Definition 3.1.4. The syntactic congruence of a function $f : A^\omega \to B^\infty$ is defined by $u \sim_f v$ if:

- $u^{-1}\overline{\mathrm{dom}(f)} = v^{-1}\overline{\mathrm{dom}(f)}$.

- either $\widehat{f}(u)$ and $\widehat{f}(v)$ are both infinite and ultimately equal, or $\forall x \in A^\omega$, such that $ux, vx \in \mathrm{dom}(f)$ we have $\widehat{f}(u)^{-1}f(ux) = \widehat{f}(v)^{-1}f(vx)$.

Proposition 3.1.5. *Let $f : A^\omega \to B^\infty$ be a rational ω-function, then $\sim_f$ is a right congruence.*

Proof. The proof goes almost like in the finite case. Let $f : A^\omega \to B^\infty$ be a rational ω-function, let $u \sim_f v$ and let $a \in A$. We want to show that $ua \sim_f va$. First $(ua)^{-1}\overline{\mathrm{dom}(f)} = (va)^{-1}\overline{\mathrm{dom}(f)}$. Let $x \in A^\omega$ such that $uax, vax \in \mathrm{dom}(f)$. If $\widehat{f}(u)$ and $\widehat{f}(v)$ are ultimately equal, then it is also the case for $\widehat{f}(ua)$ and $\widehat{f}(va)$. If $\widehat{f}(u)$ and $\widehat{f}(v)$ are finite then let $g(y) := \widehat{f}(u)^{-1}f(uy) = \widehat{f}(v)^{-1}f(vy)$. If $\bigwedge_y g(ay)$ is infinite then it is an infinite suffix of $\widehat{f}(ua)$ and $\widehat{f}(va)$ which are thus ultimately equal. Let us assume $\bigwedge_y g(ay)$ is finite:

$$\begin{aligned}
\widehat{f}(ua)^{-1}f(uax) &= \left(\bigwedge_y f(uay)\right)^{-1} f(uax)\\
&= \left(\bigwedge_y \widehat{f}(u)g(ay)\right)^{-1} \widehat{f}(u)g(ax)\\
&= \left(\widehat{f}(u)\bigwedge_y g(ay)\right)^{-1} \widehat{f}(u)g(ax)\\
&= \left(\bigwedge_y g(ay)\right)^{-1} g(ax)\\
&= \widehat{f}(va)^{-1}f(vaw)
\end{aligned}$$

Hence $ua \sim_f va$ which concludes the proof. □

Example 3.1.6. We compute for the three functions of Ex. 3.1.1 the right syntactic congruence. We have for any word $ux \in A^\omega$ that $\widehat{f}_{\text{erase}}(u)^{-1}f_{\text{erase}}(ux) = f_{\text{erase}}(x)$, and thus for any finite words u, v we have $u \sim_{f_{\text{erase}}} v$. Hence $\sim_{f_{\text{erase}}}$ is just the trivial congruence.

Similarly we have $\widehat{f}_{\#a}(u)^{-1}f_{\#a}(ux) = f_{\#a}(x)$, and thus $\sim_{f_{\#a}}$ is also trivial.

Let $u \in (a+b)^*$, $\alpha \in (a+b)$, and let $x := (\#aa)^\omega$ then we have $\widehat{f}_{\text{blocks}}(\alpha u\alpha)^{-1}f_{\text{blocks}}(\alpha u\alpha x) = f_{\text{blocks}}(\alpha u\alpha x) = \alpha u\alpha x$, which means that the right syntactic congruence of f_{blocks} has an infinite index.

Lemma 3.1.7. *Let $f : A^\omega \to B^\infty$ be a rational ω-function, then for any finite word $u \in A^*$, $\widehat{f}(u)$ is a rational word.*

Proof. Let $\mathcal{T}$ be an ω-transducer realizing the function f. Let $u \in A^*$ be a finite word, we can define an automaton recognizing $L(u)$, the words in B^∞ which can be output by reading a word beginning with u. Thus $L(u)$ is a rational subset of B^∞, and thus $\widehat{f}(u) = \bigwedge L(u)$ is a rational word. □

In the following, when a Büchi automaton has only final states, we will rather write it as a triple (Q, Δ, I) instead of a quadruple (Q, Δ, I, Q), for simplicity. We call this class of deterministic automata *left automata.* Note that a left automaton recognizes a closed language. A sequential transducer whose underlying automaton is a left automaton is called *left-sequential.*

Let f be a rational ω-function. Based on the right syntactic congruence $\sim_f$, we define $\mathcal{T}_f := (\mathcal{A}_f, i, o)$ with $\mathcal{A}_f := (Q_f, \Delta_f, I_f, F_f)$ the *minimal sequential transducer* of f by:

- $Q_f := \{[u] \mid\ u \in {\downarrow}\mathrm{dom}(f) \cap A^*\}$
- $\Delta_f := \{([u], a, [ua]) \mid\ a \in A\}$
- $I_f := \{[\epsilon]\}$

- $F_f := Q_f$
- $i([\epsilon]) := \left[\begin{array}{ll} \widehat{f}(\epsilon) & \text{if } \widehat{f}(\epsilon) \text{ is finite} \\ \alpha & \text{if } \widehat{f}(\epsilon) = \alpha\beta^\omega \text{ (in normal form) and } \beta \neq \epsilon \end{array}\right.$
- $o([u], a, [ua]) := \left[\begin{array}{ll} \widehat{f}(u)^{-1}\widehat{f}(ua) & \text{if } \widehat{f}(ua) \text{ is finite} \\ \beta & \text{if } \widehat{f}(u) = \alpha\beta^\omega \text{ (in n.f.) and } \beta \neq \epsilon \\ \alpha & \text{if } \widehat{f}(u) \text{ finite, } \widehat{f}(u)^{-1}\widehat{f}(ua) = \alpha\beta^\omega \text{ (in n.f.) and } \beta \neq \epsilon \end{array}\right.$

Remark 3.1.8. Just like for the case of finite words, notice that $\sim_{\mathcal{A}_f} = \sim_f$, by definition, and that the number of states of $\mathcal{A}_f$ is the number of equivalence classes of $\sim_f$ (up to adding a sink state). In particular if $\sim_f$ has infinite index then $\mathcal{A}_f$ actually has an infinite number of states.

Proposition 3.1.9. *The outputs of the minimal sequential transducer of a rational ω-function are well-defined.*

Proof. Let $f : A^\omega \to B^\infty$ be a rational ω-function. We show that the output functions are well-defined, *i.e.* do not depend on the choice of representative. According to Lem. 3.1.7, for any finite word $u \in A^*$, $\widehat{f}(u)$ is rational, and in particular $\widehat{f}(\epsilon)$, hence i is well-defined. Let $u \sim_f v$ and let $a \in A$ such that $ua \in {\downarrow}\mathrm{dom}(f)$. We want to show that the output function o is well-defined. Let us assume that $\widehat{f}(u)$ is infinite, then according to Lem. 3.1.7, there exists $\alpha \in A^*, \beta \in A^+$ such that $\widehat{f}(u) = \alpha\beta^\omega$ (in normal form). Since $u \sim_f v$, we know that $\widehat{f}(u)$ and $\widehat{f}(v)$ are ultimately equal and in particular, $\widehat{f}(v) = \gamma\beta^\omega$ for some $\gamma \in B^*$. Let us now assume that $\widehat{f}(u)$ is finite, in that case, so is $\widehat{f}(v)$. Since $u \sim_f v$ we have $\widehat{f}(u)^{-1}f(ux) = \widehat{f}(v)^{-1}f(vx)$, for any $x \in A^\omega$ such that $ux, vx \in \mathrm{dom}(f)$, and we denote this word by $g(x)$.

$$\begin{array}{rcl} \widehat{f}(u)^{-1}\widehat{f}(ua) & = & \left(\bigwedge_x f(ux)\right)^{-1} \bigwedge_x f(uax) \\ & = & \left(\bigwedge_x \widehat{f}(u)g(x)\right)^{-1} \bigwedge_x \widehat{f}(u)g(ax) \\ & = & \left(\bigwedge_x g(x)\right)^{-1} \bigwedge_x g(ax) \\ & = & \widehat{f}(v)^{-1}\widehat{f}(va) \end{array}$$

Hence we obtain that $o([u], a, [ua])$ is well-defined. ☐

Theorem 3.1.10. *The transducer $\mathcal{T}_f$ realizes $\overline{f}$.*

Proof. Let $f : A^\omega \to B^\infty$ be a rational ω-function and let g denote the function realized by $\mathcal{T}_f$. Let us first remark that $\mathrm{dom}(g) = \overline{\mathrm{dom}(f)} = \mathrm{dom}(\overline{f})$. Let $x \in \overline{\mathrm{dom}(f)}$. Let us assume that $\widehat{f}(\epsilon) = \alpha\beta^\omega$ is infinite. Then we have $i([\epsilon]) = \alpha$ and the output of any transition in $\mathcal{T}_f$ is equal to β. This means that $\overline{f}$ and g are both the constant function equal to $\alpha\beta^\omega$ over $\overline{\mathrm{dom}(f)}$.

We now assume that there exists a letter a and a prefix of x, $u < ua < x$ such that $\widehat{f}(u)$ is finite and $\widehat{f}(ua)$ is infinite (and thus equal to $\overline{f}(x)$). Let $[\epsilon] \xrightarrow{ua|\gamma}_{\mathcal{T}_f} [ua]$ denote the initial run of $\mathcal{T}_f$ over ua. Let $\widehat{f}(u)^{-1}\widehat{f}(ua) = \alpha\beta^\omega$, with $\beta \neq \epsilon$ by assumption. By definition of $\mathcal{T}_f$ we have that $i([\epsilon])\gamma = \widehat{f}(u)\alpha$, and we also know that all the transitions after $[ua]$ will output β. Hence $g(x) = \widehat{f}(u)\alpha\beta^\omega = \widehat{f}(ua) = \overline{f}(x)$.

Let us finally assume that for all prefixes $u < x$ we have $\widehat{f}(u)$ finite. By definition of the outputs of $\mathcal{T}_f$, $g(x) = \widehat{f}(\epsilon) \cdot \left(\prod_{i\in\mathbb{N}} \widehat{f}(x(:i))^{-1}\widehat{f}(x(:i+1))\right) = \lim_{i\in\mathbb{N}} \widehat{f}(x(:i)) = \overline{f}(x)$. ☐

In contrast with the case of finite words, the minimal sequential transducer of a function f is actually the minimal sequential transducer of $\overline{f}$. We still show that this minimal sequential transducer is coarser than any sequential transducer realizing f.

Theorem 3.1.11. *Let f be a sequential ω-function realized by a sequential ω-transducer with underlying automaton $\mathcal{A}$, then $\sim_{\mathcal{A}} \sqsubseteq \sim_f$.*

Proof. Let $\mathcal{T} = (\mathcal{A}, i, o)$ be a sequential ω-transducer realizing f with $\mathcal{A} = (Q, \Delta, \{q_0\}, F)$ deterministic. Let $u \sim_{\mathcal{A}} v$ and let p be the state of $\mathcal{A}$ reached from the initial state by u and v. Let $\mathcal{T}_p := (\mathcal{A}_p, p \mapsto \epsilon, o)$ with $\mathcal{A}_p := (Q, \Delta, \{p\}, F)$, and let $f_p := [\![\mathcal{T}_p]\!]$. We have $x \in u^{-1}\overline{\mathrm{dom}(f)}$, if and only if the run of $\mathcal{A}$ from p over x always stays in the co-accessible part of $\mathcal{A}$, and thus in particular $u^{-1}\overline{\mathrm{dom}(f)} = v^{-1}\overline{\mathrm{dom}(f)}$.

Let $x \in A^\omega$ such that $ux, vx \in \mathrm{dom}(f)$. Let us denote the initial runs of $\mathcal{T}$ over u and v by $q_0 \xrightarrow{u|\gamma} p$ and $q_0 \xrightarrow{v|\gamma'} p$, respectively. Let $\alpha\beta^\omega := \bigwedge \{f_p(y) \mid y \in \mathrm{dom}(f_p)\}$, and let $p \xrightarrow{x|\delta} F$. If $\beta \neq \epsilon$ then $\widehat{f}(u) = i(q_0)\gamma\alpha\beta^\omega$ and $\widehat{f}(v) = i(q_0)\gamma'\alpha\beta^\omega$ are ultimately equal. If $\beta = \epsilon$, we have $\widehat{f}(u) = i(q_0)\gamma\alpha$, $\widehat{f}(v) = i(q_0)\gamma'\alpha$, $f(ux) = i(q_0)\gamma\delta$ and $f(vx) = i(q_0)\gamma'\delta$. Hence we obtain that $\widehat{f}(u)^{-1}f(ux) = \alpha^{-1}\delta = \widehat{f}(v)^{-1}f(vx)$, and thus $u \sim_f v$. □

We obtain a Myhill-Nerode-like characterization for sequential functions. As we will see later, this characterization is closely related to the characterization given in [BC04] in terms of *weak twinning property* and continuity.

Theorem 3.1.12. *A rational ω-function f is sequential if and only if the two following conditions hold:*

- *$\sim_f$ has finite index*
- *$\overline{f}$ extends f*

Proof. If f is sequential then from Th. 3.1.11, we know that $\sim_f$ has finite index. Furthermore, according to Prop. 3.1.2, $\overline{f}$ extends f. Conversely, If $\sim_f$ has finite index, then $\mathcal{T}_f$ is a finite transducer realizing $\overline{f}$, according to Th. 3.1.10. Since f is rational, then so is its domain, thus by taking the product of $\mathcal{T}_f$ with a deterministic (Muller) automaton recognizing $\mathrm{dom}(f)$, we obtain a sequential transducer realizing $\overline{f}_{|\mathrm{dom}(f)}$, which is equal to f by assumption. □

Remark 3.1.13. The characterization of Th. 3.1.12 is tightly linked to the one from [BC04] which is expressed in terms of the *weak twinning property* (defined below) and continuity of the function for the prefix distance. As we will see below, the finite index of the syntactic congruence is actually equivalent to being realized by a transducer satisfying the weak twinning property. Over functions which only produce infinite words then we also have an equivalence between continuity and the liminf extending the function. In the general case however, the liminf extending the function does not mean continuity for the prefix distance topology but for the topology induced by the prefix partial-order (both topologies coincide over infinite words).

Example 3.1.14. As we have seen in Ex 3.1.1, the function $\overline{f}_{\text{blocks}}$ extends f_{blocks}. However, from in Ex. 3.1.6 we have that the right syntactic congruence of f_{blocks} is infinite, and thus the function is not sequential.

From Ex. 3.1.6 we know that $\sim_{f_{\#a}}$ has index 1. However, in Ex. 3.1.1 we have seen that $\overline{f}_{\#a}$ does not extend $f_{\#a}$. Hence $\sim_{f_{\#a}}$ is not sequential either.

3.1.2 Minimization of sequential transducers

Minimization of sequential transducers over infinite words works in almost the same way as in the finite case. The difference is that we actually minimize the transducer of the liminf of the function.

Let $\mathcal{T} = (\mathcal{A}, i, o)$ be an ω-transducer, let p be a state and let the longest common prefix of outputs from p be $\alpha_p\beta_p^\omega := \bigwedge \left\{\alpha \mid\ p \xrightarrow{u|\alpha} P \text{ final}\right\}$, in normal form. In the following, a state p such that $\beta_p \neq \epsilon$ is called a *constant state*. Then $\mathcal{T}$ is called in *earliest normal form* if 1) for any state p, $\alpha_p = \epsilon$ and 2) for any transition (p, a, q) such that p is constant, we have $o(p, a, q) = \beta_p$.

Proposition 3.1.15. *Given an ω-transducer $\mathcal{T}$, one can obtain in* PTime *a new transducer $\widehat{\mathcal{T}}$ in earliest normal form realizing the same function.*

Proof. We give the definition of $\widehat{\mathcal{T}}$, show that it realizes the same function and is indeed in earliest normal form, and finally we give a procedure to compute it. Let $\mathcal{T} = (\mathcal{A}, i, o, t)$ be a transducer realizing a function f, let p be a state of $\mathcal{A}$ and let $\alpha_p\beta_p^\omega$ denote the longest common prefix of outputs from p in normal form. We define $\widehat{\mathcal{T}} = \left(\mathcal{A}, \widehat{i}, \widehat{o}\right)$ by:

- $\widehat{i}(p) := i(p)\alpha_p$
- $\widehat{o}(p, a, q) := \left[\begin{array}{ll} \alpha_p^{-1} o(p, a, q)\alpha_q & \text{if } \beta_p = \epsilon \\ \beta_p & \text{if } \beta_p \neq \epsilon \end{array} \right.$

If x is a word for which $\mathcal{T}$ never reaches a constant state in an accepting run, one can easily see that all the α_ps cancel out, and thus $[\![\widehat{\mathcal{T}}]\!](x) = f(x)$. If $\mathcal{T}$ reaches a constant state q in an accepting run over x, then up until this constant state, all the α_ps cancel out except for α_q and then all transitions output β_q, which means again that $[\![\widehat{\mathcal{T}}]\!](x) = f(x)$.

By construction, $\widehat{T}$ is in earliest normal form. We have left to show that we can compute the words α_p, β_p in PTime. We denote by $\mathcal{A}_p$ the automaton recognizing L_p the language of words produced from p. $\mathcal{A}_p$ is obtained from $\mathcal{A}$ by replacing the labels of transitions by their outputs.

We split the proof into three distinct case: 1) L_p contains a finite word, 2) L_p contains a unique infinite word and 3) L_p contains at least two distinct infinite words. To decide if the first case holds, one only has to remove all non ϵ-transition and check if there is a circuit of ϵ-transitions from a state which is reachable from p and which is final. This can be done in PTime and in that case $\beta_p = \epsilon$, and the size of α_p is at most linear in $|Q|$. This means that we can reduce the problem to computing the longest common prefix of an automaton on finite words and thus, using [BC00] we obtain α_p in PTime.

If we are not in case 1), let us show how to decide if 2) or 3) holds, in PTime. We can check for emptiness of L_p, and if it is not empty then we can obtain an infinite word uv^ω, in PTime. From this, assuming $\mathcal{A}_p$ is trim, we can check if there is a finite word w such that $w(|w|) \neq uv^\omega(|w|)$, in NLogSpace. If there is no such word, then $\alpha_p\beta_p^\omega = uv^\omega$, so we only have to put uv^ω in normal form. Otherwise, if there is such a w, then there is one of polynomial size, and again using the result of [BC00], we obtain α_p in PTime. □

We now give the minimization theorem for sequential ω-transducers, but before that we need two small lemmas.

Lemma 3.1.16. *Let $f : A^\omega \to B^\infty$ be a function. Then $\sim_f \sqsubseteq \sim_{\overline{f}}$.*

Proof. Let $f : A^\omega \to B^\infty$ be a function, and let $u \sim_f v$. Of course we have $\overline{\overline{\mathrm{dom}(f)}} = \overline{\mathrm{dom}(f)}$. Let us notice that we have $\widehat{\overline{f}} = \widehat{f}$. Let x be such that $ux, vx \in \overline{\mathrm{dom}(f)}$. Let us assume that $\widehat{f}(u)$ is infinite, then since $u \sim_f v$, we know that $\widehat{f}(v)$ is also infinite and they are ultimately equal. If we assume that $\widehat{f}(u)$ is finite then $\widehat{f}(u)^{-1}\overline{f}(ux) = \widehat{f}(u)^{-1} \lim_{n\in\mathbb{N}} \widehat{f}(ux(:n)) = \lim_{n\in\mathbb{N}} \widehat{f}(u)^{-1}\widehat{f}(ux(:n))$. However, according to the proof of Prop. 3.1.9 we have for any word w that $\widehat{f}(u)^{-1}\widehat{f}(uw) = \widehat{f}(v)^{-1}\widehat{f}(vw)$, and thus we obtain that $\widehat{f}(u)^{-1}\overline{f}(ux) = \widehat{f}(v)^{-1}\overline{f}(vx)$ which concludes the proof. □

Lemma 3.1.17. *Let $f : A^\omega \to B^\infty$ be a function such that $\overline{f}$ extends f. Then $\sim_{\overline{f}} \sqsubseteq \sim_f$.*

Proof. Let $f : A^\omega \to B^\infty$ be such that $\overline{f}$ extends f. Again we have $\overline{\overline{\mathrm{dom}(f)}} = \overline{\mathrm{dom}(f)}$, and $\widehat{\overline{f}} = \widehat{f}$. Let $u \sim_{\overline{f}} v$, and let x be such that $ux, vx \in \mathrm{dom}(f)$, then in particular, $ux, vx \in \overline{\mathrm{dom}(f)}$,$\overline{f}(ux) = f(ux)$ and $\overline{f}(vx) = f(vx)$. If $\widehat{f}(u)$ is infinite then $\widehat{f}(v)$ is also infinite and they are ultimately equal since $u \sim_{\overline{f}} v$. If $\widehat{f}(u)$ is finite then $\widehat{f}(u)^{-1}f(ux) = \widehat{f}(u)^{-1}\overline{f}(ux) = \widehat{f}(v)^{-1}\overline{f}(vx) = \widehat{f}(v)^{-1}f(vx)$. □

Corollary 3.1.18. *Let f be a sequential ω-function, then $\sim_f = \sim_{\overline{f}}$.*

Proof. We conclude using Lem. 3.1.16 on one hand, and Prop. 3.1.2 and Lem. 3.1.17 on the other hand. □

We can now state and prove our minimization result.

Theorem 3.1.19. *Given a sequential ω-transducer, one can compute the associated minimal sequential transducer (up to renaming states) in* PTime.

Proof. Let $\mathcal{T} = (\mathcal{A}, i, o)$, with $\mathcal{A} = (Q, \Delta, I, F)$ be a sequential transducer realizing a function f. According to Prop. 3.1.15 we can assume without loss of generality that $\mathcal{T}$ is in earliest normal form.

The algorithm is almost the same as in the finite case.

In the following we identify the states of $\mathcal{A}$ with the equivalence classes of $\sim_{\mathcal{A}}$. The initial partition is given for all words u, v by $u \sim_0 v$ if: 1) $u^{-1}\overline{\mathrm{dom}(f)} = v^{-1}\overline{\mathrm{dom}(f)}$ and 2) $\forall a \in A$, $o([u]_{\mathcal{A}}, a, [ua]_{\mathcal{A}}) = o([v]_{\mathcal{A}}, a, [va]_{\mathcal{A}})$.

Let $i \in \mathbb{N}$, we define inductively for all words u, v, $u \sim_{i+1} v$ by: $u \sim_i v$ and $\forall a \in A$, $ua \sim_i va$. Of course we have $\forall i \in \mathbb{N}$ that $\sim_{\mathcal{A}} \sqsubseteq \sim_{i+1} \sqsubseteq \sim_i$, which means that we reach a fixpoint relation for some i, which we denote by $\sim_*$.

Let us first show that $\sim_*$ is finer than $\sim_f$. We will actually show that $\sim_* \sqsubseteq \sim_{\overline{f}}$, since we have $\sim_f = \sim_{\overline{f}}$ from Cor. 3.1.18. For this we construct a sequential transducer with right transition congruence $\sim_*$ and which realizes $\overline{f}$. Then according to Th. 3.1.11, we do obtain $\sim_* \sqsubseteq \sim_{\overline{f}}$. The relation $\sim_*$ is a right congruence since it is a fixpoint for right multiplication. Let $\mathcal{T}_* = (\mathcal{A}_*, i_*, o_*, t_*)$ with $\mathcal{A}_* = (Q_*, \Delta_*, I_*)$ be defined as:

- $Q_* := \{[u]_* \mid\ u \in {\downarrow}\mathrm{dom}(f) \cap A^*\}$
- $\Delta_* := \{([u]_*, a, [ua]_*) \mid\ a \in A\}$
- $I_* := \{[\epsilon]_*\}$
- $i_*([\epsilon]_*) := i([\epsilon]_{\mathcal{A}})$
- $o_*([u]_*, a, [ua]_*) := o([u]_{\mathcal{A}}, a, [ua]_{\mathcal{A}})$

The output function is well-defined according to point 2). Given a word in the closure of the domain of f, its outputs for $\mathcal{T}_*$ will be exactly the same as for $\mathcal{T}$. Since $\mathcal{T}$ is in earliest normal form, we can see that $\mathcal{T}_*$ thus realizes $\overline{f}$.

We only have left to show that $\sim_f \sqsubseteq \sim_*$, which we do by showing by induction on $i \geq 0$ that $\sim_f \sqsubseteq \sim_i$. Let $u \sim_f v$ we have 1) $u^{-1}\overline{\mathrm{dom}(f)} = v^{-1}\overline{\mathrm{dom}(f)}$. Let us assume first $\widehat{f}(u)$ is infinite equal to $\alpha\beta^\omega$. This means that $\widehat{f}(v)$ is ultimately equal to $\widehat{f}(u)$ and thus of the form $\alpha'\beta^\omega$. Since $\mathcal{T}$ is in earliest normal, we know that all the transitions after $[u]_\mathcal{A}$ or $[v]_\mathcal{A}$ have to produce β, which proves 2). If we now assume that $\widehat{f}(u)$ is finite, then so is $\widehat{f}(v)$, and let us consider the initial runs $q_0 \xrightarrow{u|\gamma}_\mathcal{T} p$ and $q_0 \xrightarrow{v|\gamma'}_\mathcal{T} p'$. Since $\mathcal{T}$ is in earliest normal form, we have that the longest common prefix of outputs from both p and p' is ϵ, and thus $\widehat{f}(u) = i(q_0)\gamma$ and $\widehat{f}(v) = i(q_0)\gamma'$. Let $a \in A$, let $\widehat{f}(ua) = \alpha\beta^\omega$ and let $\widehat{f}(va) = \alpha'\beta^\omega$ (with possibly $\beta = \epsilon$). Since $\mathcal{T}$ is in earliest form, we have that $p \xrightarrow{a|\alpha} q$ and $p' \xrightarrow{a|\alpha'} q'$. However, we know from the proof of Prop. 3.1.9 that $\alpha = \alpha'$. Thus we obtain $o(p, a, q) = o(p', a, q')$ which proves 2), and thus $u \sim_0 v$.

Now let us assume that for some $i \in \mathbb{N}$ we have $\sim_f \sqsubseteq \sim_i$. Let $u \sim_f v$, we have by hypothesis that $u \sim_i v$. Since $\sim_f$ is a right congruence we also have for any $a \in A$ that $ua \sim_f va$ which means that $ua \sim_i va$, and hence $u \sim_{i+1} v$, which concludes the proof of correctness.

Like in the finite case, this algorithm runs in PTime. □

3.1.3 Quasi-sequential transductions

Sequential functions over infinite words are characterized by the conditions given in Th. 3.1.12, namely the finiteness of the index of the syntactic right congruence and the fact that the liminf extends the function. If we remove the constraint on the liminf, we obtain a class of functions that we call *quasi-sequential* functions, and which has already been (implicitly) studied in [BC04]. Intuitively, quasi-sequential functions can be thought of as the generalization of subsequential functions from finite to infinite words, in the following way: they can be realized by sequential transducers extended with the possibility to append a (possibly infinite) word at the "end" of the run (if it produces a finite word) depending on some regular property of the input. A typical example of quasi-sequential function which is not sequential is the function $f_{\#a}$ from Ex. 3.1.1 defined over the alphabet $\{a, b\}$ by $f_{\#a}(x) = a^\omega$ if the number of as in x is infinite and $f_{\#a}(x) = b^\omega$ otherwise.

We also show below that quasi-sequential functions can be characterized by the *weak twinning property*, a machine characterization from [BC04] generalizing the one for finite words introduced by Choffrut. Finally we show that quasi-sequential functions are exactly the functions for which determinization terminates (although it is not always correct when the function is not sequential).

Determinization algorithm We define the determinization procedure from [BC04], which we call (as in the finite case) *powerset construction with delays*, and we state some of its properties. The construction is very similar to the case of finite words.

Let $\mathcal{T} = (\mathcal{A}, i, o)$ be a transducer, with underlying automaton $\mathcal{A} = (Q, \Delta, I, F)$, realizing a function f. We describe $\mathcal{S} := (\mathcal{D}, i', o')$, with $\mathcal{D} := (Q', \Delta', S_0)$. Note that $\mathcal{S}$ is sequential by definition but may have an infinite number of states in general, but the authors of [BC04] show that $\mathcal{S}$ is finite and realizes f when f is sequential. Actually we show slightly more general results, which are present in [BC04], but not explicitly. We assume that $\mathcal{T}$ is trim and in earliest normal form, without loss of generality. Let $C \subseteq Q$ be the set of constant states of $\mathcal{T}$, and let β_p^ω denote the longest common prefix of outputs from state p of $\mathcal{T}$.

Let $\alpha\beta^\omega := \bigwedge_{q\in I} i(q)\beta_q^\omega$ and let $S_0 := \big\{(q, \alpha^{-1}i(q)) \mid q \in I \setminus C\big\} \uplus \big\{(q, \alpha^{-1}i(q)\beta_q^\omega\big\}$, with $i'(S_0) := \alpha$. We build Q' and Δ' up from S_0 using the following steps. Let P be a state already constructed in Q' and let $a \in A$. We define:

$$\begin{aligned} R := \ & \Big\{(q,\gamma\beta) \mid (p,\gamma)\in P,\ p \xrightarrow{a|\beta}_{\mathcal{T}} q,\ q\notin C\Big\} \uplus \Big\{(q,\gamma\beta^\omega) \mid (p,\gamma\beta^\omega)\in P,\ p \xrightarrow{a}_{\mathcal{A}} q, p\in C\Big\} \\ & \uplus \Big\{(q,\delta\gamma\beta_q^\omega) \mid (p,\delta)\in P,\ p \xrightarrow{a|\gamma}_{\mathcal{T}} q,\ p\notin C, q\in C\Big\} \end{aligned}$$

Let $\alpha\beta^\omega := \bigwedge_{(q,\gamma)\in R}\gamma$. If $\alpha \neq \epsilon$, then we add the new state $S := \big\{(q, \alpha^{-1}\gamma) \mid (q,\gamma)\in R\big\}$ to Q' and the transition $P \xrightarrow{a|\alpha}_{\mathcal{S}} S$ to Δ'. If $\alpha = \epsilon$, then we add the state R to Q' and the transition $P \xrightarrow{a|\beta}_{\mathcal{S}} S$ to Δ'.

Remark 3.1.20. Note that $\mathcal{S}$ is deterministic in its transitions but may not be finite in general.

Proposition 3.1.21. *Let f be a rational ω-function realized by a transducer $\mathcal{T}$ and let $\mathcal{S}$ be the transducer obtained by powerset construction with delays. Then $\mathcal{S}$ realizes $\overline{f}$.*

Proof. Let $\mathcal{T}$ be a transducer realizing a function $f : A^\omega \to B^\infty$, let $\mathcal{S}$ be the transducer obtained by powerset construction with delays and let g denote the function realized by $\mathcal{S}$. We assume that $\mathcal{T}$ is trim without loss of generality and we thus obtain that $\mathrm{dom}(\mathcal{S}) = \overline{\mathrm{dom}(f)}$. Let $x \in \overline{\mathrm{dom}(f)}$ and let $u < x$. Let $S_0 \xrightarrow{u|\alpha} S$ denote the initial run of $\mathcal{S}$ over u. By construction of $\mathcal{S}$, we know that the output along the initial run is a common prefix to all outputs of initial runs of $\mathcal{T}$ over u and thus we have $i'(S_0)\alpha \leq \widehat{f}(u)$. Let $S = \{(p_1,\gamma_1),\dots,(p_n,\gamma_n)\}$, for some $n\in\mathbb{N}$. By definition of $\mathcal{S}$ we have $\bigwedge_{i\in\mathbf{n}}\gamma_i = \beta^\omega$ (with β potentially empty).

Let us assume that for all $u < x$, $\widehat{f}(u)$ is finite. Then we have that $\beta = \epsilon$, and there are two cases. Either $\gamma_i = \epsilon$ for some $i \in \mathbf{n}$ or we have $\gamma_i \wedge \gamma_j = \epsilon$, for $i, j \in \mathbf{n}$. In the latter case we have that $i'(S_0)\alpha = \widehat{f}(u)$ since $\mathcal{T}$ is in earliest normal form. In the first case, if p_i is constant then we have $i'(S_0)\alpha = \widehat{f}(u)$. Finally if p_i is non-constant this means there are two runs final from p_i producing words δ, δ' such that $\delta \wedge \delta' = \epsilon$ again because $\mathcal{T}$ is in earliest normal form. In all cases, we obtain $i'(S_0)\alpha = \widehat{f}(u)$. Thus we have $g(x) = \lim_{u<x}\widehat{f}(u) = \overline{f}(x)$.

Let us now assume that $\widehat{f}(u)$ is infinite for some $u < x$. This must mean that $\beta \neq \epsilon$ and actually we must have $\gamma_i = \beta^\omega$ for all $i \in \mathbf{n}$, by definition of $\mathcal{S}$. Thus we have $\widehat{f}(u) = \delta\beta^\omega$ and $i(S_0)\alpha = \delta\beta^k$ for some integer k. Furthermore, by definition of $\mathcal{S}$, any transition after P must output β and thus we obtain $g(x) = \delta\beta^\omega = \overline{f}(x)$. □

Weak twinning property A transducer $\mathcal{T}$ is said to satisfy the *weak twinning property* (WTP) if for any initial runs $p_1 \xrightarrow{u|\alpha_1} q_1 \xrightarrow{v|\beta_1} q_1$ and $p_2 \xrightarrow{u|\alpha_2} q_2 \xrightarrow{v|\beta_2} q_2$ the following holds:

- if q_1, q_2 are not constant then $\mathsf{del}(i(p_1)\alpha_1, i(p_2)\alpha_2) = \mathsf{del}(i(p_1)\alpha_1\beta_1, i(p_2)\alpha_2\beta_2)$
- if q_1 is not constant and q_2 is constant and produces the rational word γ, then either $\beta_1 = \epsilon$, or $i(p_1)\alpha_1\beta_1^\omega = i(p_2)\alpha_2\gamma$.

Note that if q_2 is constant and $\beta_2 \neq \epsilon$ then $\gamma = \beta_2^\omega$.

Proposition 3.1.22 ([BC04])**.** *Let $\mathcal{T}$ be a transducer satisfying the WTP, then the transducer obtained by powerset construction with delays terminates. The procedure is in* ExpTime.

Proof. The authors of [BC04] have a very slightly different model of transducers where a run has to produce an infinite word in order to be final. However this does not change the proofs here and in particular, they show that the sizes of the delays are polynomial with respect to the original transducer. Thus the construction causes, as for the finite case, an exponential blow-up of the state space. □

Characterizations of quasi-sequential transductions We need two lemmas before giving the theorem characterizing quasi-sequential transductions.

Lemma 3.1.23. *Let $\mathcal{T}$ be an ω-transducer, and let $\mathcal{S}$ be the ω-transducer obtained by powerset construction with delays with underlying automaton $\mathcal{D}$. Then $\sim_{\mathcal{D}} \sqsubseteq \sim_f$.*

Proof. Let $\mathcal{T}$ be a transducer realizing a function $f : A^\omega \to B^\infty$, and let $\mathcal{S}$ be the transducer obtained by powerset construction with delays with underlying automaton $\mathcal{D}$. Let $u, v \in A^*$ such that $u \sim_{\mathcal{D}} v$, and let $S_0 \xrightarrow{u|\alpha} S$ and $\xrightarrow{v|\alpha'} S$ denote the initial runs of $\mathcal{S}$ over u and v, respectively. Our goal is to show that $u \sim_f v$. First since $\mathcal{S}$ realizes $\overline{f}$ and $\mathrm{dom}(\overline{f}) = \overline{\mathrm{dom}(f)}$, we have $u^{-1}\overline{\mathrm{dom}(f)} = v^{-1}\overline{\mathrm{dom}(f)}$. If S is constant and produces an infinite word γ, then $\widehat{f}(u) = i'(S_0)\alpha\gamma$ and $\widehat{f}(v) = i'(S_0)\alpha'\gamma$ which are ultimately equal. Otherwise, there must exist pairs $(p_1, \gamma_1), (p_2, \gamma_2) \in S$ such that $\gamma_1 \wedge \gamma_2 = \epsilon$. Since $\mathcal{T}$ is in normal form, we obtain $\widehat{f}(u) = i'(S_0)\alpha$ and $\widehat{f}(v) = i'(S_0)\alpha'$. Let $x \in A^\omega$ and let $(p, \gamma) \in S$ such that x has a final run from p producing δ. If γ is infinite, we have $f(ux) = i'(S_0)\alpha\gamma$ and $f(vx) = i'(S_0)\alpha'\gamma$, and if γ is finite we have $f(ux) = i'(S_0)\alpha\gamma\delta$ and $f(vx) = i'(S_0)\alpha'\gamma\delta$. Either way we obtain $\widehat{f}(u)^{-1}f(ux) = \widehat{f}(u)^{-1}f(ux)$, and thus $u \sim_f v$. □

Lemma 3.1.24. *Let $\mathcal{T}$ be an ω-transducer realizing f such that $\sim_f$ has finite index. Then $\mathcal{T}$ satisfies the WTP.*

Proof. Let $\mathcal{T}$ be an ω-transducer realizing f such that $\sim_f$ has finite index. Let us assume towards a contradiction that $\mathcal{T}$ does not satisfy the WTP. Let $p_1 \xrightarrow{u|\alpha_1} q_1 \xrightarrow{v|\beta_1} q_1$ and $p_2 \xrightarrow{u|\alpha_2} q_2 \xrightarrow{v|\beta_2} q_2$ denote two initial runs.

We first consider the case where q_1, q_2 are not constant states and $\mathsf{del}(i(p_1)\alpha_1, i(p_2)\alpha_2) \neq \mathsf{del}(i(p_1)\alpha_1\beta_1, i(p_2)\alpha_2\beta_2)$. If we have $|\beta_1| = |\beta_2|$ this means that there exists a mismatch, *i.e.* a position k such that $i(p_1)\alpha_1\beta_1(k) \neq i(p_2)\alpha_2\beta_2(k)$. Hence we obtain that $\widehat{f}(uv^n) < i(p_2)\alpha_2\beta_2$ for any positive integer n, which means that there exists an integer N such that $\widehat{f}(uv^n) = \widehat{f}(uv^N)$, for $n \geq N$. Since q_2 is non-constant, and $\mathcal{T}$ is in earliest normal form, there is a word x over which there is an accepting run from q_2 producing γ such that $\gamma \wedge \beta_2 = \epsilon$. Thus we have that $\widehat{f}(uv^n)^{-1}f(uv^nx) = \widehat{f}(uv^N)^{-1}i(p_2)\alpha_2\beta_2^n\gamma$ which takes an infinite number of different values, contradicting the fact that $\sim_f$ has finite index. If $|\beta_1| \neq |\beta_2|$, we assume without loss of generality that $|\beta_1| < |\beta_2|$. Since q_1 is non constant and $\mathcal{T}$ is in earliest normal form, we have for any $n \in \mathbb{N}$ that $\widehat{f}(uv^n) \leq i(p_1)\alpha_1\beta_1^n$. Hence we have that $\widehat{f}(uv^n)^{-1}i(p_2)\alpha_2\beta_2^n$ takes an infinite number of values. As above, using the fact that q_2 is constant, we can obtain a word x producing γ over a final run from q_2 such that $\gamma \wedge \beta_2 = \epsilon$. Thus we obtain again that $\widehat{f}(uv^n)^{-1}f(uv^nx) = \widehat{f}(uv^n)^{-1}i(p_2)\alpha_2\beta_2^n\gamma$ takes an infinite number of values, contradicting that $\sim_f$ has finite index.

Let us assume that q_1 is non-constant and q_2 is constant and produces γ (with $\mathcal{T}$ in earliest normal form). We also assume that $\beta_1 \neq \epsilon$ and $i(p_1)\alpha_1\beta_1^\omega \neq i(p_2)\alpha_2\gamma$. This means in particular that there is an integer N such that for all $n \geq N$, we have $\widehat{f}(uv^n) = \widehat{f}(uv^N)$. Since q_1 is not constant, let x be a word producing δ over a final run from q_1, such that $\beta_1 \wedge \delta = \epsilon$. Then as before we obtain that for $n \geq N$, $\widehat{f}(uv^n)^{-1}f(uv^n x) = \widehat{f}(uv^N)^{-1}i(p_1)\alpha_1\beta_1^n\delta$ which takes an infinite number of values, and thus contradicts the finite index of $\sim_f$. □

We can now state our result characterizing quasi-sequential transductions, which will prove useful in the following section.

Theorem 3.1.25. *Let $\mathcal{T}$ be an ω-transducer realizing a function f, and let $\mathcal{S}$ be the transducer obtained by powerset construction with delays. The following are equivalent:*

- *$\mathcal{T}$ satisfies the WTP*
- *$\mathcal{S}$ is finite*
- *$\sim_f$ has finite index*

Proof. The proof from (1) to (2) follows Prop. 3.1.22, the proof from (2) to (3) follows Lem. 3.1.23 and finally (3) to (1) comes from Lem. 3.1.24. □

3.1.4 Determinization preserves aperiodicity

Just like in the finite case determinization preserves aperiodicity. This will allow us to show that the canonical bimachine of an aperiodic transduction is aperiodic.

Lemma 3.1.26. *Let $\mathcal{T}$ be an ω-transducer satisfying the WTP, and let $\mathcal{S}$ be the transducer obtained by powerset construction with delays. If $\mathcal{T}$ is aperiodic then $\mathcal{S}$ is aperiodic.*

Proof. We adapt the proof of Lem. 2.1.15 to the infinite case. Let $\mathcal{T} = (\mathcal{A}, i, o)$, with $\mathcal{A} = (Q, \Delta, I, F)$, be an $\mathbf{A}$-transducer realizing a quasi-sequential function $f : A^\omega \to B^\infty$. Let $\mathcal{S} = (\mathcal{D}, i', o')$, with $\mathcal{D} = (Q', \Delta', S_0)$, be the sequential transducer obtained from $\mathcal{T}$ by powerset construction with delays. We want to show that $\mathcal{S}$ is aperiodic and actually we will show that $\mathcal{D}$ is counter-free, which is equivalent according to Prop. 1.3.9 since $\mathcal{S}$ is sequential. $\mathcal{A}$ is aperiodic so there is an integer n such that $\forall u \in A^*$, $u^n \equiv_{\mathcal{A}} u^{n+1}$.

Let $u \in A^+$ be a word, let k be a positive integer and let $R_0 \xrightarrow{u|\alpha_0}_{\mathcal{S}} R_1 \ldots R_{k-1} \xrightarrow{u|\alpha_{k-1}}_{\mathcal{S}} R_0$ denote a counter in $\mathcal{S}$. Let us assume that k is the size of the smallest such counter, which means that all R_js are pairwise distinct, we want to show $k = 1$.

Let $\Gamma_0 := \alpha_0 \cdots \alpha_{k-1}$, for $1 \leq j < k$ let $\Gamma_j := \alpha_j \cdots \alpha_{k-1}\alpha_0 \cdots \alpha_{j-1}$ and let us note that $\Gamma_j\alpha_j = \alpha_j\Gamma_{j+1 \mod k}$. Let $R = \{q_1, \ldots, q_m\}$ denote the states appearing in R_0. For $0 \leq j < k$, the states of R_j are exactly the states which can be reached in $\mathcal{A}$ from some state of R_0 by reading $u^{kn+j} \equiv_{\mathcal{A}} u^{kn}$. This means that the states of R_j are the same as the states of R_0, namely $q_1, \ldots, q_m$. Thus let $R_j = \{(q_1, \beta_{1,j}), \ldots, (q_m, \beta_{m,j})\}$. Let us notice that $\beta_{i,j}$ is infinite if and only if q_i is constant and produces an infinite word. In that case all states reachable from q_i are constant with an infinite delay. If we assume that all states in R are constant with infinite delay, then either they have an empty common prefix and thus $\alpha_j = \epsilon$ for any $0 \leq j < k$ which means that all R_js are equal and thus $k = 1$, or they are all equal to some word in normal form β^ω and $\alpha_j = \beta$ for any $0 \leq j < k$ and again $k = 1$. Note that the common prefix cannot be a non-empty finite word, otherwise we could not loop back to R_0.

Let us assume that some states have a finite delay and some have an infinite one. Let q_l be a constant state with infinite delay from which there is a run reading u^ω. Adapting equation (2) in the proof of Lem. 2.1.15 we obtain that $\beta_{l,j} = \Gamma_j^\omega$. Similarly, if q_i a constant state which can be reached by such a state q_l, we have by equation (4) that $\beta_{i,j} = \Gamma_j^\omega$.

For the states with finite delay, we use the proof of Lem. 2.1.15 and we obtain that for some $x \in \{j, j+1\}$, γ is a prefix of all finite $\beta_{i,x}$s. However, γ is also a prefix of Γ_x, which means that γ is a common prefix to all $\beta_{i,x}$s, and thus $\gamma = \epsilon$ by definition of $\mathcal{S}$. □

As a corollary, we obtain a way to decide if a sequential ω-transduction is aperiodic.

Theorem 3.1.27. *A sequential ω-transduction is $\boldsymbol{A}$-rational if and only if its domain is aperiodic and its minimal sequential transducer is aperiodic.*

Proof. If f is sequential and in $\mathbf{A}$, then there exists an aperiodic transducer realizing it. From Lem. 3.1.26, we know that the determinization of the transducer produces an aperiodic transducer, which is finite (Th. 3.1.25) and realizes $\overline{f}$ (Th. 3.1.10). Furthermore, we have that the minimal sequential transducer of $\overline{f}$ is the minimal sequential transducer of f, from Cor. 3.1.18. Finally, if f is aperiodic, then in particular $\mathrm{dom}(f)$ is aperiodic.

Conversely, if $\mathrm{dom}(f)$ is aperiodic, then from Th. 1.3.12, there is an aperiodic deterministic Muller automaton recognizing it. Thus by taking the product of the minimal sequential transducer and the deterministic automaton, we obtain an aperiodic sequential Muller transducer realizing f. □

3.2 Canonical models for rational functions over infinite words

In this section we extend some of our results for transductions over finite words to transductions over infinite words. We are not able to describe all the minimal bimachines of a transduction like in the finite case, however we can at least exhibit a canonical way to compute a transduction. In particular, we show that this canonical bimachine is always aperiodic for an aperiodic transduction, which is the main result of the section. Unlike for the finite case, the delay look-ahead does not always give enough information to realize the function sequentially. Indeed, when we annotate the input with the delay look-ahead information, what we obtain is a *quasi-sequential* transduction. For this reason we introduce a second look-ahead which is sufficient to make any quasi-sequential transduction sequential. However this second look-ahead, called the *ultimate look-ahead*, does not share the same minimality properties as the delay look-ahead. Composing the two look-aheads, we obtain nonetheless a canonical look-ahead fine enough to realize any rational transduction sequentially.

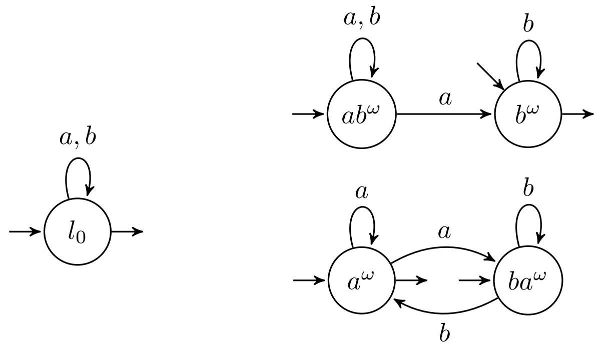
3.2.1 Bimachines and transductions

We define bimachines over infinite words and show that one can go from a bimachine to a transducer while preserving the transition congruence, just like in the finite case.

Bimachines An *ω-bimachine* over alphabets A, B is given as a tuple $\mathcal{B} = (\mathcal{L}, \mathcal{R}, i, o)$ where $\mathcal{L} = (Q_{\mathcal{L}}, \Delta_{\mathcal{L}}, \{l_0\})$ is a deterministic accessible (Büchi) automaton with only accepting states, called the *left automaton of* $\mathcal{B}$, $\mathcal{R} = (Q_{\mathcal{R}}, \Delta_R, I_{\mathcal{R}}, F_{\mathcal{R}})$ is a co-deterministic co-accessible automaton, called the *right automaton of* $\mathcal{B}$, $i : Q_{\mathcal{R}} \to B^*$ is the *initial function*, $o : Q_{\mathcal{L}} \times A \times Q_{\mathcal{R}} \to B^*$ is the *output function*. We also add the semantic restriction that $[\![\mathcal{L}]\!] = [\![\mathcal{R}]\!]$.

Let $u \in A^*$ be a finite word and let l and r be runs of $\mathcal{L}$ and $\mathcal{R}$, respectively, over u. We extend o to $Q_{\mathcal{L}} \times A^* \times Q_{\mathcal{R}}$ by setting $o(l(1), u, r(|r|)) := \prod_{1 \leq j \leq |u|} o(l(j), u(j), r(j+1))$. Let $x \in A^\omega$ be an infinite word and let l and r be final runs of $\mathcal{L}$ and $\mathcal{R}$, respectively, over x. We extend o to $Q_{\mathcal{L}} \times A^\omega$ by setting $o(l(1), x) := \lim_{n \in \mathbb{N}} o(l(1), x(:n), r(n+1))$. Furthermore if l and r are accepting let $\alpha := i(r(1))o(l(1), x)$, then we say that (x, α) is realized by $\mathcal{B}$ and we denote by $[\![\mathcal{B}]\!]$ the set of pairs realized by $\mathcal{B}$.

Example 3.2.1. The bimachine given in Fig. 3.2 on the next page realizes the function $f_{\#a} : \{a, b\}^\omega \to \{a, b\}^\infty$ which maps words with an infinite number of as to a^ω, and other words to b^ω.



l	α	r	$o(l,\alpha,r)$
_	_	ab^ω	b
_	_	b^ω	b
_	_	ba^ω	a
_	_	a^ω	a

Figure 3.2: Left and right Büchi automata and output table of a bimachine realizing $f_{\#a}$.

From bimachines to transducers As in the finite case, it is easy to show that from a bimachine one can obtain an unambiguous transducer realizing the same function just by taking the product of the two automata.

Proposition 3.2.2. *Let $\mathcal{B}$ be a bimachine realizing a rational ω-function f with left and right automata $\mathcal{L}$ and $\mathcal{R}$, respectively. Then one can construct in* PTime *an unambiguous transducer realizing f with underlying automaton $\mathcal{L} \times \mathcal{R}$.*

Proof. Let $\mathcal{B} = (\mathcal{L}, \mathcal{R}, i, o)$ be a bimachine realizing a function f with left automaton $\mathcal{L} = (Q_{\mathcal{L}}, \Delta_L a, \{l_0\})$ and right automaton $\mathcal{R} = (Q_{\mathcal{R}}, \Delta_R, I_{\mathcal{R}}, F_{\mathcal{R}})$. The main idea is to consider the product of the two automata. The transducer will guess the run of the right automaton, and co-determinism ensures that there is only one possible run. We define a transducer $\mathcal{T} := (\mathcal{A}, i', o')$ with $\mathcal{A} := (Q, \Delta, I, F)$ by:

- $Q := Q_{\mathcal{L}} \times Q_{\mathcal{R}}$
- $\Delta := \{((l, r'), a, (l', r)) \mid a \in A,\ (l, a, l') \in \Delta_{\mathcal{L}},\ (r', a, r) \in \Delta_{\mathcal{R}}\}$
- $I := \{l_0\} \times I_{\mathcal{R}}$
- $F := Q_{\mathcal{L}} \times F_{\mathcal{R}}$ if $\mathcal{R}$ is a Büchi automaton, and
  $F := \{P_{\mathcal{L}} \times P_{\mathcal{R}} \mid P_{\mathcal{L}} \subseteq Q_{\mathcal{L}},\ P_{\mathcal{R}} \in F_{\mathcal{R}}\}$ if $\mathcal{R}$ is a Muller automaton.
- $i'(l_0, r) := i(r)$
- $o'((l, r'), a, (l', r)) := o(l, a, r)$

By construction $\mathcal{T}$ realizes f and its underlying automaton is indeed $\mathcal{L} \times \mathcal{R}$. □

Like in the case of transductions over finite words, we are able to show conversely that from a transducer, one can obtain a bimachine with the same transition congruence. Again, we first need the notion of bimachine left minimization before showing this result.

3.2.2 Left minimization of bimachines

We are able to minimize the left automaton of bimachines, but not the right one. We define the syntactic right congruence of a function with respect to a fixed right automaton. This subsection very closely follows the corresponding Subsec. 2.2.2 over finite words. All the main ideas needed here are obtained by combining those of Subsec. 2.2.2 or Sec. 3.1.

$\mathcal{R}$-syntactic congruence Let $f : A^\omega \to B^\infty$ be a function and let $\mathcal{R}$ be a right automaton recognizing $\mathrm{dom}(f)$. We define for each state of $\mathcal{R}$, *i.e.* each equivalence class $[x]_\mathcal{R}$ of $\approx_\mathcal{R}$, a function $\widehat{f}_{[x]_\mathcal{R}} : u \mapsto \bigwedge \{f(uy) \mid \ y \approx_\mathcal{R} x\}$. Note that the right automaton will often be implicit and we will rather write $\widehat{f}_x$. Intuitively, this function outputs over a word u the longest possible word, with the look-ahead information that the suffix is in $[x]_\mathcal{R}$. We define $\overline{f}^\mathcal{R} : A^\omega \to B^\infty$ by $\overline{f}^\mathcal{R}(x) = \lim_{uy=x} \widehat{f}_y(u)$ for any $x \in \mathrm{dom}(f)$.

Definition 3.2.3. The *$\mathcal{R}$-syntactic congruence* of $f : A^\omega \to B^\infty$ is defined for $u, v \in A^*$ by $u \sim_f^\mathcal{R} v$ if:

- $u^{-1}\overline{\mathrm{dom}(f)} = v^{-1}\overline{\mathrm{dom}(f)}$ and
- $\forall x$ such that $ux, vx \in \mathrm{dom}(f)$, either $\widehat{f}_x(u)$ and $\widehat{f}_x(v)$ are ultimately equal or $\widehat{f}_x(u)^{-1}f(ux) = \widehat{f}_x(v)^{-1}f(vx)$

Intuitively, u and v are equivalent if when we remove the contributions of u and v from $f(ux)$ and $f(vx)$, respectively, we obtain the same word left to write. Note here that the contributions of u and v are computed with the knowledge that the suffix is equivalent to x, with respect to $\mathcal{R}$.

Proposition 3.2.4. *Let f be a function and let $\mathcal{R}$ be a right automaton recognizing* $\mathrm{dom}(f)$*, then $\sim_f^\mathcal{R}$ is a right congruence.*

Proof. Let $f : A^\omega \to B^\infty$ be a function and let $\mathcal{R}$ be a right automaton recognizing $\overline{\mathrm{dom}(f)}$, let $u \sim_f^\mathcal{R} v$ and let $a \in A$. We want to show that $ua \sim_f^\mathcal{R} va$. First $(ua)^{-1}\overline{\mathrm{dom}(f)} = a^{-1}(u^{-1}\overline{\mathrm{dom}(f)}) = (va)^{-1}\overline{\mathrm{dom}(f)}$. Let $x \in A^\omega$ such that $ux, vx \in \mathrm{dom}(f)$. If $\widehat{f}_{ax}(u)$ is infinite then it is ultimately equal to $\widehat{f}_{ax}(v)$. We have $\widehat{f}_x(ua) = \widehat{f}_{ax}(u)$ and $\widehat{f}_x(va) = \widehat{f}_{ax}(v)$ and thus $\widehat{f}_x(ua)$ and $\widehat{f}_x(va)$ are ultimately equal. If $\widehat{f}_{ax}(u)$ and $\widehat{f}_{ax}(v)$ are finite then we have $\widehat{f}_{ax}(u)^{-1}f(uax) = \widehat{f}_{ax}(v)^{-1}f(vax)$, and we let $g(x)$ denote this word. If $\bigwedge_{y \approx_\mathcal{R} x} g(y)$ is infinite then $\widehat{f}_x(ua)$ and $\widehat{f}_x(va)$ are ultimately equal to this word. Otherwise, we have:

$$\begin{array}{rcl}
\widehat{f}_x(ua)^{-1}f(uax) & = & \left(\bigwedge_{y \approx_\mathcal{R} x} f(uay)\right)^{-1} f(uax) \\
& = & \left(\bigwedge_{y \approx_\mathcal{R} x} \widehat{f}_{ax}(u)g(y)\right)^{-1} \widehat{f}_{ax}(u)g(x) \\
& = & \left(\widehat{f}_{ax}(u) \bigwedge_{y \approx_\mathcal{R} x} g(y)\right)^{-1} \widehat{f}_{ax}(u)g(x) \\
& = & \left(\bigwedge_{y \approx_\mathcal{R} x} g(y)\right)^{-1} g(x) \\
& = & \widehat{f}_x(va)^{-1}f(vax)
\end{array}$$

Hence $ua \sim_f^\mathcal{R} va$ which concludes the proof. □

Example 3.2.5. Let $\mathcal{R}$ be the right automaton of Fig. 3.2 on the facing page. Then for any finite word $u \in A^*$, we have $\widehat{f}_{a^\omega}(u) = \widehat{f}_{ba^\omega}(u) = a^\omega$ and $\widehat{f}_{b^\omega}(u) = \widehat{f}_{ab^\omega}(u) = b^\omega$. This means that $\sim_f^\mathcal{R}$ is the trivial congruence, which is consistent with the fact that the left automaton of the bimachine in Fig. 3.2 on the preceding page is trivial.

Left minimal bimachine Let $f : A^\omega \to B^\infty$ be a rational function over infinite words and let $\mathcal{R} := (Q_\mathcal{R}, \Delta_\mathcal{R}, I_\mathcal{R}, F_\mathcal{R})$ be a right automaton recognizing $\mathrm{dom}(f)$. Using the $\mathcal{R}$-syntactic congruence of f we define the *left minimal bimachine* of f with respect to $\mathcal{R}$. We set $\mathcal{B}_f(\mathcal{R}) := (\mathcal{L}_f(\mathcal{R}), \mathcal{R}, i, o)$ with $\mathcal{L}_f(\mathcal{R}) := (Q_\mathcal{L}, \Delta_\mathcal{L}, l_0)$ defined by:

- $Q_{\mathcal{L}} := \left\{ [u]_f^{\mathcal{R}} \mid u \in {\downarrow}\mathrm{dom}(f) \cap A^* \right\}$
- $\Delta_{\mathcal{L}} := \left\{ ([u]_f^{\mathcal{R}}, a, [ua]_f^{\mathcal{R}}) \right\}$
- $l_0 := [\epsilon]_f^{\mathcal{R}}$
- $i([x]_{\mathcal{R}}) := \left[\begin{array}{ll} \widehat{f}_x(\epsilon) & \text{if } \widehat{f}_x(\epsilon) \text{ is finite} \\ \alpha & \text{if } \widehat{f}_x(\epsilon) = \alpha\beta^\omega \text{ and } \beta \neq \epsilon \end{array} \right.$
- $o([u]_f^{\mathcal{R}}, a, [x]_{\mathcal{R}}) := \left[\begin{array}{ll} \widehat{f}_{ax}(u)^{-1}\widehat{f}_x(ua) & \text{if } \widehat{f}_x(ua) \text{ is finite} \\ \beta & \text{if } \widehat{f}_{ax}(u) = \alpha\beta^\omega \text{ and } \beta \neq \epsilon \\ \alpha & \text{if } \widehat{f}_{ax}(u) \text{ is finite, } \widehat{f}_{ax}(u)^{-1}\widehat{f}_x(ua) = \alpha\beta^\omega,\ \beta \neq \epsilon \end{array} \right.$

Like in the finite case, the left minimal bimachine outputs the longest possible word (except if it is infinite), given the look-ahead information. Again this bimachine may have an infinite number of states, in general, when the look-ahead is not fine enough.

We show that the outputs of the left minimal bimachine are well-defined.

Proposition 3.2.6. *Let f be a rational ω-function and let $\mathcal{R}$ be a right automaton recognizing* $\mathrm{dom}(f)$*. The outputs of $\mathcal{B}_f(\mathcal{R})$ are well-defined.*

Proof. Let $f : A^\omega \to B^\infty$ be a rational ω-function and let $\mathcal{R}$ be a right automaton recognizing $\mathrm{dom}(f)$. We show that the output functions of $\mathcal{B}_f(\mathcal{R})$ are well-defined. Using Lem. 3.1.7, we know that for any word $x \in \mathrm{dom}(f)$, we have that $\widehat{f}_x(\epsilon) = \widehat{f_{|[x]_{\mathcal{R}}}}(\epsilon)$ is rational and thus the initial outputs of $\mathcal{B}_f(\mathcal{R})$ are well-defined.

Let $u \sim_f^{\mathcal{R}} v$, let $a \in A$ and let $x \in A^\omega$ such that $uax, vax \in \mathrm{dom}(f)$. If $\widehat{f}_{ax}(u)$ is infinite then so is $\widehat{f}_{ax}(v)$ and since they are ultimately equal (and rational from Lem. 3.1.7) we know that they have the same ultimate period. If $\widehat{f}_{ax}(u)$ is finite then we have for any $y \approx_{\mathcal{R}} ax$ that $\widehat{f}_{ax}(u)^{-1}f(uy) = \widehat{f}_{ax}(v)^{-1}f(vy)$, and we denote this word by $g(y)$.

$$\begin{array}{rcl} \widehat{f}_{ax}(u)^{-1}\widehat{f}_x(ua) & = & \left(\bigwedge_{y \approx_{\mathcal{R}} ax} f(uy)\right)^{-1} \bigwedge_{y \approx_{\mathcal{R}} x} f(uay) \\ & = & \left(\bigwedge_{y \approx_{\mathcal{R}} ax} \widehat{f}_{ax}(u)g(y)\right)^{-1} \bigwedge_{y \approx_{\mathcal{R}} x} \widehat{f}_{ax}(u)g(ay) \\ & = & \left(\bigwedge_{y \approx_{\mathcal{R}} ax} g(y)\right)^{-1} \bigwedge_{y \approx_{\mathcal{R}} x} g(ay) \\ & = & \widehat{f}_{ax}(v)^{-1}\widehat{f}_x(va) \end{array}$$

□

We now can show that $\mathcal{B}_f(\mathcal{R})$ realizes f. However, $\mathcal{B}_f(\mathcal{R})$ may be infinite if $\mathcal{R}$ does not give enough look-ahead information to realize f sequentially.

Theorem 3.2.7. *Let f be a rational ω-function and let $\mathcal{R}$ be a right automaton recognizing* $\mathrm{dom}(f)$*. The bimachine $\mathcal{B}_f(\mathcal{R})$ realizes $\overline{f}^{\mathcal{R}}$.*

Proof. Let $f : A^\omega \to B^\infty$ be a rational ω-function, let $\mathcal{R}$ be a right automaton recognizing $\mathrm{dom}(f)$ and let $x \in \mathrm{dom}(f)$. Let us first assume that for all $uy = x$, $\widehat{f}_y(u)$ is finite, then by definition of the outputs of the left minimal bimachine $\mathcal{B}_f(\mathcal{R})$, $[\![\mathcal{B}_f(\mathcal{R})]\!](x) = \widehat{f}_u(\epsilon) \cdot \left(\prod_{i\in\mathbb{N}} \widehat{f}_{x(i+1:)}(x(:i))^{-1}\widehat{f}_{x(i+2:)}(x(:i+1))\right) = \overline{f}^{\mathcal{R}}(x)$. If $\widehat{f}_x(\epsilon)$ is infinite equal to $\alpha\beta^\omega$, then

$i([x]_\mathcal{R}) = \alpha$ and $\forall uay = x$, we have $o([u]_f^\mathcal{R}, a, [y]_\mathcal{R}) = \beta$ and thus we obtain $[\![\mathcal{B}_f(\mathcal{R})]\!](x) = \overline{f}^\mathcal{R}(x)$. Similarly let us assume that $uay = x$, with $a \in A$ such that $\widehat{f}_{ay}(u)$ is finite and $\widehat{f}_y(ua) = f(uay)$ is infinite. By definition of $\mathcal{B}_f(\mathcal{R})$, we have $i([x]_\mathcal{R})o([\epsilon]_f^\mathcal{R}, u, [ay]_\mathcal{R}) = \widehat{f}(u)$. Let $\widehat{f}_{ay}(u)^{-1}\widehat{f}_y(ua) = \alpha\beta^\omega$ in normal form. Then we have $i([x]_\mathcal{R})o([\epsilon]_f^\mathcal{R}, ua, [y]_\mathcal{R}) = \widehat{f}(u)\alpha$ and for any $n \in \mathbb{N}$ we have $i([x]_\mathcal{R})o([\epsilon]_f^\mathcal{R}, uay(:n), [y(n+1:)]_\mathcal{R}) = \widehat{f}(u)\alpha\beta^n$. Finally, we obtain that $[\![\mathcal{B}_f(\mathcal{R})]\!](x) = \overline{f}^\mathcal{R}(x)$. □

We show that when a right automaton gives enough look-ahead information, then the associated left minimal bimachine realizes the function.

Theorem 3.2.8. *Let $\mathcal{B}$ be a bimachine with right automaton $\mathcal{R}$ realizing a function f. Then $\overline{f}^\mathcal{R} = f$.*

Proof. Let $\mathcal{B} = (\mathcal{L}, \mathcal{R}, o)$ be a bimachine realizing f. We know that $\overline{f}^\mathcal{R} \leq f$. Let $ux \in \mathrm{dom}(f)$, and let $\alpha = i([ux]_\mathcal{R})o([\epsilon]_\mathcal{L}, u, [x]_\mathcal{R})$. We must have that $\alpha \leq \widehat{f}_x(u)$ and thus by taking the limit we have that $f(ux) \leq \overline{f}^\mathcal{R}(x)$, which concludes the proof. □

As in the finite case, the left minimal bimachine, with respect to a right automaton $\mathcal{R}$, has a coarser left automaton than any bimachine realizing f with $\mathcal{R}$ as right automaton.

Theorem 3.2.9. *Let f be a transduction realized by a bimachine with left and right automata $\mathcal{L}$ and $\mathcal{R}$, respectively. Then $\sim_\mathcal{L} \sqsubseteq \sim_f^\mathcal{R}$.*

Proof. Let $\mathcal{B} = (\mathcal{L}, \mathcal{R}, i, o)$ be a bimachine with left and right automata $\mathcal{L} = (Q_\mathcal{L}, \Delta_\mathcal{L}, \{l_0\})$ and $\mathcal{R} = (Q_\mathcal{R}, \Delta_\mathcal{R}, I_\mathcal{R}, F_\mathcal{R})$ realizing f and let $u \sim_\mathcal{L} v$. We have $u^{-1}\mathrm{dom}(f) = v^{-1}\mathrm{dom}(f)$ since $\mathcal{L}$ recognizes $\mathrm{dom}(f)$.

Let $x \in A^\omega$ be such that $ux, vx \in \mathrm{dom}(f)$, let $\alpha = i([ux]_\mathcal{R})o([\epsilon]_\mathcal{L}, u, [x]_\mathcal{R})$, let $\beta = i([vx]_\mathcal{R})o([\epsilon]_\mathcal{L}, v, [x]_\mathcal{R})$, let $\gamma = o([u]_\mathcal{L}, x)$ and let $\delta = \bigwedge_{y \approx_\mathcal{R} x} o([u]_\mathcal{L}, y)$. We have $f(ux) = \alpha\gamma$, $\widehat{f}_x(u) = \alpha\delta$, $f(vx) = \beta\gamma$ and $\widehat{f}_x(v) = \beta\delta$. If $\widehat{f}_x(u)$ is infinite, then it is ultimately equal to δ, and so is $\widehat{f}_x(v)$. Otherwise, we have that $\widehat{f}_x(u)^{-1}f(ux) = \delta^{-1}\gamma = \widehat{f}_x(v)^{-1}f(vx)$. Hence we have shown that $u \sim_f v$. □

Left minimization algorithm In order to define the minimization algorithm, we need a notion of earliest normal form for ω-bimachines.

Let $\mathcal{B} = (\mathcal{L}, \mathcal{R}, i, o)$ be a bimachine, and let $ux \in \mathrm{dom}(f)$. Let us denote by $\alpha_{u,x}\beta_{u,x}^\omega := \bigwedge\{o([u]_\mathcal{L}, y) \mid y \approx_\mathcal{R} x\}$ the longest common prefix of outputs from $[u]_\mathcal{L}$ with look-ahead $[x]_\mathcal{R}$ (in normal form). Then $\mathcal{B}$ is called in *earliest normal form* if for any $ux \in \mathrm{dom}(f)$, we have 1) $\alpha_{u,x} = \epsilon$ and 2) if $\beta_{u,ax} \neq \epsilon$ then $o([u]_\mathcal{L}, a, [x]_\mathcal{R}) = \beta_{u,ax}$.

Proposition 3.2.10. *Given a bimachine $\mathcal{B}$, one can obtain in* PTime *a new bimachine $\widehat{\mathcal{B}}$ in earliest form realizing the same function.*

Proof. We give the definition of $\widehat{\mathcal{B}}$, show that it realizes the same function and is indeed in earliest form, and finally we give a procedure to compute it. Let $\mathcal{B} = (\mathcal{L}, \mathcal{R}, i, o)$ be a bimachine realizing a function f and let $ux \in \mathrm{dom}(f)$. Let $\alpha_{u,x}\beta_{u,x}^\omega$ denote the longest common prefix of outputs from $[u]_\mathcal{L}$ with look-ahead $[x]_\mathcal{R}$. We define $\widehat{\mathcal{B}} = \left(\mathcal{L}, \mathcal{R}, \widehat{i}, \widehat{o}\right)$ by:

- $\widehat{i}([x]_\mathcal{R}) := i([x]_\mathcal{R})\alpha_{\epsilon,[x]_\mathcal{R}}$

- $\widehat{o}([u]_{\mathcal{L}}, a, [v]_{\mathcal{R}}) := \left[\begin{array}{ll} \alpha_{u,ax}^{-1} o([u]_{\mathcal{L}}, a, [x]_{\mathcal{R}}) \alpha_{ua,x} & \text{if } \beta_{u,ax} = \epsilon \\ \beta_{u,ax} & \text{if } \beta_{u,ax} \neq \epsilon \end{array} \right.$

Let x be a word such that for all $uy = x$, $\widehat{f}_y(u)$ is finite. In that case, one can easily see that all the $\alpha_{u,x}$s cancel out, and thus $\widehat{\mathcal{B}}$ realizes f as well. If $\widehat{f}_x(\epsilon) = \alpha\beta^\omega$ is infinite, then we have $\beta_{\epsilon,x} = \beta$, and $i([x]_{\mathcal{R}})\alpha_{\epsilon,x}\beta^\omega = f(x)$. For any $uy = x$, we have by definition that $\widehat{o}([\epsilon]_{\mathcal{L}}, u, [y]_{\mathcal{R}}) = \beta^{|u|}$, and thus we obtain $[\![\widehat{B}]\!](x) = f(x)$. Similarly, if we have some $uay = x$ such that $\widehat{f}_{ay}(u)$ is finite and $\widehat{f}_y(ua)$ is infinite, then we have $\widehat{i}([uay]_{\mathcal{R}})\widehat{o}([\epsilon]_{\mathcal{L}}, u, [ay]_{\mathcal{R}}) = \widehat{f}_{ay}(u)$. Let $\widehat{f}_{ay}(u)^{-1}\widehat{f}_y(ua) = \alpha\beta^\omega$, then we have $\widehat{i}([uay]_{\mathcal{R}})\widehat{o}([\epsilon]_{\mathcal{L}}, ua, [y]_{\mathcal{R}}) = \widehat{f}_{ay}(u)\alpha$ and also $\widehat{i}([uay]_{\mathcal{R}})\widehat{o}([\epsilon]_{\mathcal{L}}, uay(:n), [y(n+1:)]_{\mathcal{R}}) = \widehat{f}_{ay}(u)\alpha\beta^n$ for any $n \in \mathbb{N}$. Thus we obtain that $\widehat{\mathcal{B}}$ realizes f and by construction, $\widehat{\mathcal{B}}$ is in earliest normal form.

Let $u \in A^*, x \in A^\omega$, we only have left to show that computing $\alpha_{u,x}$ and $\beta_{u,x}$ can be done in PTIME. We consider the transducer $\mathcal{T}$ obtained from $\mathcal{B}$ by Prop. 3.2.2. The state space of $\mathcal{T}$ is $\mathcal{L} \times \mathcal{R}$ and we have that the longest common prefix of outputs from state $([u]_{\mathcal{L}}, [x]_{\mathcal{R}})$ is $\alpha_{([u]_{\mathcal{L}},[x]_{\mathcal{R}})}\beta^\omega_{([u]_{\mathcal{L}},[x]_{\mathcal{R}})} = \alpha_{u,x}\beta^\omega_{u,x}$. Thus using the same procedure as in the proof of Prop. 3.1.15 we can compute $\alpha_{u,x}$ and $\beta_{u,x}$ in PTIME. □

We now give the minimization theorem:

Theorem 3.2.11. *Given a bimachine with right automaton $\mathcal{R}$ realizing a rational ω-function, one can compute in* PTIME *$\mathcal{B}_f(\mathcal{R})$, the left minimal bimachine with respect to $\mathcal{R}$.*

Proof. Let $\mathcal{B} = (\mathcal{L}, \mathcal{R}, i, o)$ be a bimachine realizing a function f with left and right automata $\mathcal{L} = (Q_{\mathcal{L}}, \Delta_{\mathcal{L}}, \{l_0\})$ and $\mathcal{R} = (Q_{\mathcal{R}}, \Delta_{\mathcal{R}}, I_{\mathcal{R}}, F_{\mathcal{R}})$, respectively. According to Prop. 3.2.10 we can assume without loss of generality that $\mathcal{B}$ is in earliest form.

We use exactly the same ideas as in the finite case.

As usual, we identify the states of $\mathcal{L}$ with the classes of $\sim_{\mathcal{L}}$. The initial partition is given for all words u, v by $u \sim_0 v$ if: 1) $u \in {\downarrow}\mathrm{dom}(f) \Leftrightarrow v \in {\downarrow}\mathrm{dom}(f)$ and 2) $\forall x$ with $uax, vax \in \mathrm{dom}(f)$, we have $o([u]_{\mathcal{L}}, a, [x]_{\mathcal{R}}) = o([v]_{\mathcal{L}}, a, [x]_{\mathcal{R}})$.

Let $i \in \mathbb{N}$, for all finite words u, v we define inductively $u \sim_{i+1} v$ by: $u \sim_i v$ and $\forall a \in A$, $ua \sim_i va$. Of course we have $\forall i \in \mathbb{N}$ that $\sim_{\mathcal{L}} \sqsubseteq \sim_{i+1} \sqsubseteq \sim_i$, which means that we reach a fixpoint relation for some i, which we denote by $\sim_*$.

Let us first show that $\sim_*$ is fine enough to realize f with $\mathcal{R}$ as right automaton, which means that $\sim_* \sqsubseteq \sim_f^{\mathcal{R}}$ according to Th. 3.2.9. The relation $\sim_*$ is a right congruence since it is a fixpoint for right multiplication. Let $\mathcal{B}_* = (\mathcal{L}_*, \mathcal{R}, i_*, o_*)$ with $\mathcal{L}_* = (Q_*, \Delta_*, I_*)$ be defined as:

- $Q_* := \{[u]_* \mid u \in {\downarrow}\mathrm{dom}(f)\}$
- $\Delta_* := \{([u]_*, a, [ua]_*) \mid a \in A\}$
- $I_* := \{[\epsilon]_*\}$
- $i_*([u]_{\mathcal{R}}) := i([u]_{\mathcal{R}})$
- $o_*([u]_*, a, [x]_{\mathcal{R}}) := o([u]_{\mathcal{L}}, a, [x]_{\mathcal{R}})$

The output function is well-defined according to point 2). Given a word in the domain of f, its outputs for $\mathcal{B}_*$ will be exactly the same as for $\mathcal{B}$ which means that $\mathcal{B}_*$ realizes f and hence $\sim_* \sqsubseteq \sim_f^{\mathcal{R}}$, according to Th. 3.2.9.

We only have left to show that $\sim_f^{\mathcal{R}} \sqsubseteq \sim_*$, which we do by showing by induction on $i \geq 0$ that $\sim_f^{\mathcal{R}} \sqsubseteq \sim_i$.

Let $u \sim_f^{\mathcal{R}} v$, we have 1) $u \in \downarrow\mathrm{dom}(f) \Leftrightarrow v \in \downarrow\mathrm{dom}(f)$. Let x be a word such that $uax, vax \in \mathrm{dom}(f)$. If we assume that $\widehat{f}_{ax}(u)$ is finite, then so is $\widehat{f}_{ax}(v)$. Since $\mathcal{B}$ is in normal form, we have that $i([uax]_{\mathcal{R}})o([\epsilon]_{\mathcal{L}}, u, ax) = \widehat{f}_{ax}(u)$ and $i([vax]_{\mathcal{R}})o([\epsilon]_{\mathcal{L}}, v, ax) = \widehat{f}_{ax}(v)$. Since $u \sim_f^{\mathcal{R}} v$, we have $\widehat{f}_{ax}(u)^{-1}f(uax) = \widehat{f}_{ax}(v)f(vax) = \alpha\beta^\omega$ which means that $o([u]_{\mathcal{L}}, a, [x]_{\mathcal{R}}) = o([v]_{\mathcal{L}}, a, [x]_{\mathcal{R}}) = \alpha$. Similarly, if $\widehat{f}_{ax}(u)$ is infinite, then so is $\widehat{f}_{ax}(v)$, and let β their ultimate period. In that case we have $o([u]_{\mathcal{L}}, a, [x]_{\mathcal{R}}) = o([v]_{\mathcal{L}}, a, [x]_{\mathcal{R}}) = \beta$. Hence $\sim_f^{\mathcal{R}} \sqsubseteq \sim_0$.

Now let us assume that for some $i \in \mathbb{N}$ we have $\sim_f^{\mathcal{R}} \sqsubseteq \sim_i$. Let $u \sim_f v$, we have by hypothesis that $u \sim_i v$. Since $\sim_f^{\mathcal{R}}$ is a right congruence we also have for any $a \in A$ that $ua \sim_f^{\mathcal{R}} va$ which means that $ua \sim_i va$, and hence $u \sim_{i+1} v$, which concludes the proof. □

From transducers to bimachines We show how one can go from a transducer to a bimachine while preserving the transition congruence.

Theorem 3.2.12. *Let $\mathcal{T}$ be an ω-transducer with underlying automaton $\mathcal{A}$ realizing a function f. Let $\mathcal{R}$ be a right automaton recognizing* $\mathrm{dom}(f)$ *such that $\approx_{\mathcal{R}} \sqsubseteq \approx_{\mathcal{A}}$, then one can obtain a bimachine with automata $\mathcal{L}$ and $\mathcal{R}$ such that $\sim_{\mathcal{A}} \sqsubseteq \sim_{\mathcal{L}}$.*

Proof. Let $\mathcal{T} = (\mathcal{A}, i, o)$ be a transducer with underlying automaton $\mathcal{A} = (Q, \Delta, I, F)$ realizing f. Let $\mathcal{R}$ be a right automaton, such that $\approx_{\mathcal{R}} \sqsubseteq \approx_{\mathcal{A}}$ and recognizing $\mathrm{dom}(f)$. We want to show that $\sim_{\mathcal{A}} \sqsubseteq \sim_f^{\mathcal{R}}$ which will conclude the proof, since $\mathcal{B}_f(\mathcal{R})$ realizes f according to Th. 3.2.7 and 3.2.8. Let $u \sim_{\mathcal{A}} v$, our goal is to show that $u \sim_f^{\mathcal{R}} v$. We have of course that $u^{-1}\overline{\mathrm{dom}(f)} = v^{-1}\overline{\mathrm{dom}(f)}$. Let x be such that $ux, vx \in \mathrm{dom}(f)$, and let $r_1, \ldots, r_n \in Q$ denote the states of $\mathcal{A}$ which can be reached from I by reading u (or v) and from which there is a final run by reading x. Let $j \in \mathbf{n}$, let $p_j \xrightarrow{u|\alpha_j} r_j$ and $q_j \xrightarrow{v|\beta_j} r_j$ denote initial runs over u and v respectively to r_j. Let $r_j \xrightarrow{x|\gamma_j} S_i$ denote a final run over x from r_j, and let $\delta_j := \bigwedge \left\{\delta \mid\ r_j \xrightarrow{y|\delta} S \text{ final },\ y \approx_{\mathcal{A}} x\right\}$. Then for any $j \in \mathbf{n}$ we have $\widehat{f}_x(u) = i(p_j)\alpha_j\delta_j$, $f(ux) = i(p_j)\alpha_j\gamma_j$, $\widehat{f}_x(v) = i(q_j)\beta_j\delta_j$ and $f(vx) = i(q_j)\beta_j\gamma_j$. Taking for instance $j = 1$, we obtain that either $\widehat{f}_x(u)$ and $\widehat{f}_x(v)$ are infinite and ultimately equal to δ_1 or $\widehat{f}_w(u)^{-1}f(uw) = \delta_1^{-1}\gamma_1 = \widehat{f}_w(v)^{-1}f(vw)$, hence $u \sim_f^{\mathcal{R}} v$ which concludes the proof. □

A first corollary is that ω-bimachines realize all the rational ω-transductions.

Corollary 3.2.13. *An ω-function is rational if and only if it is realized by some ω-bimachine.*

In particular this transfers to aperiodic transductions.

Corollary 3.2.14. *An ω-transduction is aperiodic if and only if it is realized by an aperiodic bimachine.*

Proof. Let $\mathcal{B}$ be an aperiodic bimachine. According to Prop. 3.2.2, we can obtain an aperiodic transducer realizing the same function.

Let $\mathcal{T}$ be an ω-transducer, with underlying aperiodic automaton $\mathcal{A} = (Q, \Delta, I, F)$, realizing a function f. Let us show that for any $x \in A^\omega$, $\{y \mid\ y \approx_{\mathcal{R}} x\}$ is aperiodic. Given a state p, the automaton $\mathcal{A}_p := (Q, \Delta, \{p\}, F)$ is aperiodic since its transition congruence is that of $\mathcal{A}$. Let $P := \left\{p \in Q \mid\ \exists \text{ a final run } p \xrightarrow{x}_{\mathcal{A}} S\right\}$. Thus we have $\{y \mid\ y \approx_{\mathcal{R}} x\} = \bigcap_{p \in P} [\![\mathcal{A}_p]\!]$, which is aperiodic since aperiodic languages are closed under intersection. According to Th 1.3.13, we can thus obtain an aperiodic right automaton $\mathcal{R}$ such that $\approx_{\mathcal{R}} \sqsubseteq \approx_{\mathcal{A}}$. Finally, from Th. 3.2.12 we obtain our result. □

Another consequence is an Elgot-Mezei theorem for rational functions over infinite words, which generalizes the result of [Car10]. Indeed a bimachine can be seen as realizing the composition of a left and a right sequential function (in both ways). Furthermore, we can always assume that one of them is letter-to-letter.

Theorem 3.2.15. *Let f be a rational ω-function over infinite words, then there exist g sequential (resp. right-sequential) and h letter-to-letter and right-sequential (resp. sequential) such that $f = g \circ h$.*

3.2.3 Look-ahead versus labeling

We introduce the notion of labeling, like in the finite case, and show the relationship between labelings and look-aheads.

Labeling Let $f : A^\omega \to B^\infty$ be an ω-function and let $\mathcal{R}$ be a right automaton recognizing $\mathrm{dom}(f)$. We define the *labeling function associated with* $\mathcal{R}$ by the right sequential transducer $\ell(\mathcal{R}) := (\mathcal{R}, i, o)$ with $i : r \mapsto \epsilon$ and $o([ax]_{\mathcal{R}}, a, [x]_{\mathcal{R}}) := (a, [x]_{\mathcal{R}})$. We define the transduction $f_{\mathcal{R}} := f \circ [\![\ell(\mathcal{R})]\!]^{-1}$. Note that $f_{\mathcal{R}}$ is a function, since $\ell(\mathcal{R})$ is injective (by unambiguity of $\mathcal{R}$). Intuitively, $f_{\mathcal{R}}$ produces the same output as f over an input annotated by the look-ahead information given by $\mathcal{R}$.

Our goal is to link the sequentiality of $f_{\mathcal{R}}$ with the fact that $\mathcal{R}$ gives enough look-ahead to have a bimachine with $\mathcal{R}$ as a right automaton. We start with the easy direction.

Proposition 3.2.16. *Let f be a rational ω-transduction realized by a bimachine with right automaton $\mathcal{R}$. Then $f_{\mathcal{R}}$ is sequential.*

Proof. Let $\mathcal{B} = (\mathcal{L}, \mathcal{R}, i, o)$ with $\mathcal{L} = (Q_{\mathcal{L}}, \Delta_{\mathcal{L}}, \{l_0\})$ be a bimachine realizing f. Let us consider the sequential transducer $\mathcal{T} := (\mathcal{A}, i', o')$ with $\mathcal{A} := (Q, \Delta, I, F)$ defined by:

- $Q := \{q_0\} \uplus Q_{\mathcal{L}} \times Q_{\mathcal{R}}$
- $\Delta := \begin{array}{l}\{((l, r'), (a, r), (l', r)) \mid (l, a, l') \in \Delta_{\mathcal{L}},\ (r', a, r) \in \Delta_{\mathcal{R}}\} \\ \cup \{(q_0, (a, [x]_{\mathcal{R}}), (l', [x]_{\mathcal{R}})) \mid (l_0, a, l') \in \Delta_{\mathcal{L}},\ ax \in \mathrm{dom}(f)\}\end{array}$
- $I := \{q_0\}$
- $F := Q_{\mathcal{L}} \times F_{\mathcal{R}}$ if $\mathcal{R}$ is a Büchi automaton and
  $F := \{P_{\mathcal{L}} \times P_{\mathcal{R}} \mid P_{\mathcal{L}} \subseteq Q_{\mathcal{L}},\ P_{\mathcal{R}} \in F_{\mathcal{R}}\}$ if $\mathcal{R}$ is a Muller automaton
- $i'(q_0) = \epsilon$
- $o'((l, r'), (a, r), (l', r)) := o(l, a, r)$ and $o'(q_0, (a, [x]_{\mathcal{R}}), (l', [x]_{\mathcal{R}})) := i([ax]_{\mathcal{R}}) o(l_0, a, [x]_{\mathcal{R}})$

The transducer $\mathcal{T}$ is sequential and realizes $f_{\mathcal{R}}$. □

Example 3.2.17. We consider the right automaton $\mathcal{R}$ given in Fig. 3.2 on page 70 and give a transducer realizing $f_{\#a,\mathcal{R}}$ in Fig. 3.3 on the facing page. Notice that $f_{\#a}$ is not sequential while $f_{\#a,\mathcal{R}}$ is.

We can now consider the converse result, which is harder to show but works exactly as in the finite case. We give a constructive proof of the result by exhibiting when possible a bimachine with $\mathcal{R}$ as a right automaton. As a side result, we show that this construction preserves aperiodicity.

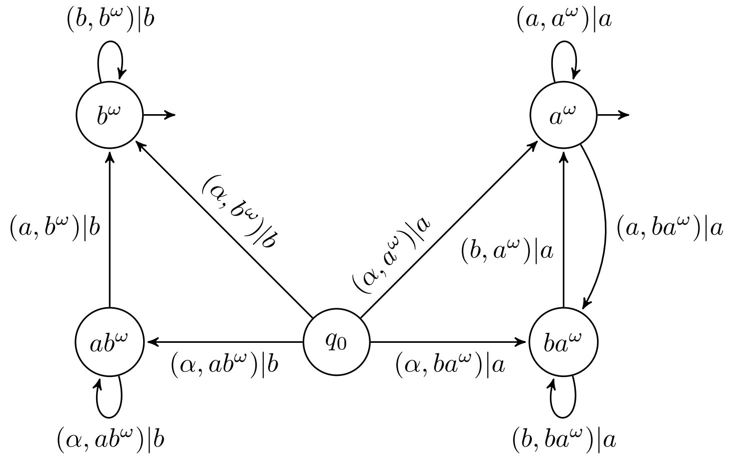


Figure 3.3: Sequential transducer realizing $f_{\#a,\mathcal{R}}$ with $\mathcal{R}$ from Fig. 3.2 on page 70.

Lemma 3.2.18. *Let $\mathcal{T}$ be an ω-transducer realizing a function f and let $\mathcal{R}$ be a right automaton recognizing* $\mathrm{dom}(f)$*. If $f_\mathcal{R}$ is sequential then there is a bimachine realizing f with right automaton $\mathcal{R}$. In that case one can compute $\mathcal{B}_f(\mathcal{R})$ in 2*ExpTime*. Furthermore if $\mathcal{T}$ and $\mathcal{R}$ are aperiodic then $\mathcal{B}_f(\mathcal{R})$ also is.*

Proof. Let $\mathcal{T} = (\mathcal{A}, i, o)$ with $\mathcal{A} = (Q, \Delta, I, F)$ be a transducer realizing a function f, let $\mathcal{R} = (Q_\mathcal{R}, \Delta_\mathcal{R}, I_\mathcal{R}, F_\mathcal{R})$ be a right automaton recognizing $\mathrm{dom}(f)$ and let us assume that $f_\mathcal{R}$ is sequential. Let us describe the steps of the proof. Step 1) we construct a transducer realizing $f_\mathcal{R}$ by taking the product of $\mathcal{T}$ and $\mathcal{R}$. Step 2) we determinize this transducer using the powerset construction with delays of [BC04], in ExpTime. We thus obtain $\mathcal{S}$ a sequential transducer realizing $f_\mathcal{R}$, of exponential size with respect to $\mathcal{T}$. Step 3) we project the input alphabet of $\mathcal{S}$, which is $A \times Q_\mathcal{R}$, to A, to obtain a transducer realizing f. Step 4) we consider the product automaton of the underlying automaton of this transducer with $\mathcal{R}$, and determinize it, using the usual powerset construction for automata on finite words, again with an ExpTime complexity to obtain a deterministic automaton $\mathcal{D}$. We are then able to exhibit a bimachine with $\mathcal{D}$ and $\mathcal{R}$ as automata realizing f. Finally we left minimize this bimachine to obtain $\mathcal{B}_f(\mathcal{R})$ (recall that $\sim_f^\mathcal{R} = \sim_{\mathcal{L}_f}$). We also show that each of the four steps preserves aperiodicity.

1) From $\mathcal{T}$ and $\mathcal{R}$ we can obtain a transducer realizing $f_\mathcal{R}$, by just taking $\mathcal{T}$ and using the states of $\mathcal{R}$ to label the input. Let $\mathcal{T}' := (\mathcal{A}', i', o')$ with $\mathcal{A}' := (Q', \Delta', I', F')$ be defined by:

- $Q' := Q \times Q_\mathcal{R}$
- $\Delta' := \{((p, s), (a, r), (q, r)) \mid (p, a, q) \in \Delta,\ (s, a, r) \in \Delta_\mathcal{R}\}$
- $I' := I \times I_\mathcal{R}$
- $F' := F \times F_\mathcal{R}$ if $\mathcal{A}$ and $\mathcal{R}$ are Büchi automata. If one of the two is a Muller automaton, we can assume that both are Muller automata by changing the final set. In that case we define $F' := \{P \times P_\mathcal{R} \mid P \in F,\ P_\mathcal{R} \in F_\mathcal{R}\}$.
- $i'(q, r) := i(q)$

- $o'((p,s),(a,r),(q,r)) := o(p,a,q)$

By construction $\mathcal{T}'$ realizes $f_{\mathcal{R}}$. Let us assume that $\mathcal{A}$ and $\mathcal{R}$ are aperiodic, and let us show that $\mathcal{A}'$ then must be aperiodic also. Let $u \in (A \times Q_{\mathcal{R}})^*$ be a word and let $\overline{u}$ denote the projection of u onto the alphabet A. Let p, q be states of $\mathcal{A}$ and let $x \in A^\omega$. We assume that we have for some integer n, $(p, [\overline{u}^n x]_{\mathcal{R}}) \xrightarrow{u^n} (q, [x]_{\mathcal{R}})$. By construction of $\mathcal{A}'$, we know that the last letter of u is $(a, [x]_{\mathcal{R}})$ for some $a \in A$. If $n > 1$, then from the definition of $\mathcal{A}'$, we know that we must have $[\overline{u}x]_{\mathcal{R}} = [x]_{\mathcal{R}}$. By aperiodicity of $\mathcal{A}$ we have $p = p_0 \xrightarrow{\overline{u}} p_1 \dots p_n \xrightarrow{\overline{u}} p_{n+1} = q$, for n large enough. We have for any $0 \leq k \leq n$ that $(p_k, [x]_{\mathcal{R}}) \xrightarrow{u} (p_{k+1}, [x]_{\mathcal{R}})$. Hence we can conclude that $(p, [x]_{\mathcal{R}}) \xrightarrow{u^{n+1}} (q, [x]_{\mathcal{R}})$ which means that $\mathcal{A}'$ is aperiodic.

2) We use the powerset construction of [BC04] on $\mathcal{T}'$, presented in Sec. 3.1, to obtain a sequential transducer $\mathcal{S}$ realizing $f_{\mathcal{R}}$ in ExpTime, since $f_{\mathcal{R}}$ is sequential by assumption. According to Lem. 3.1.26, this procedure preserves aperiodicity.

3) We project away the labels of the input letters of $\mathcal{S}$ to obtain a transducer realizing f. Furthermore this transducer is unambiguous, otherwise some word would have two distinct labelings. Let us show that this procedure preserves aperiodicity. Let $\mathcal{A}_{\mathcal{S}}$ be the deterministic underlying automaton of $\mathcal{S}$, which we assume to be aperiodic. We consider $\overline{\mathcal{A}_{\mathcal{S}}}$, the same automaton but with the input labels projected onto A. Let $u \in A^*$, let p, q be states of $\overline{\mathcal{A}_{\mathcal{S}}}$, let n be a positive integer such that $p \xrightarrow{u^n}_{\overline{\mathcal{A}_{\mathcal{S}}}} q$. Let $x \in A^\omega$ be a word such that there is a final run of $\overline{\mathcal{A}_{\mathcal{S}}}$ over x from q. This means that there is a labeling w of u^n, *i.e.* $\overline{w} = u^n$, ending in $(a, [x]_{\mathcal{R}})$ for some $a \in A$, such that: $p \xrightarrow{w}_{\mathcal{A}_{\mathcal{S}}} q$. Since $\mathcal{R}$ is aperiodic, we know that for m large enough, $u^m x \approx_{\mathcal{R}} u^{m+1}x$. Let v be the labeling of u with last letter $(a, [u^m x]_{\mathcal{R}})$, then $w = v^k w'$ with w' a labeling of u^m. By aperiodicity of $\mathcal{A}_{\mathcal{S}}$, if k is large enough we have a run $p \xrightarrow{v^{k+1}w'}_{\mathcal{A}_{\mathcal{S}}} q$. Since $\overline{v^{k+1}w'} = u^{n+1}$, we finally have $p \xrightarrow{u^{n+1}}_{\overline{\mathcal{A}_{\mathcal{S}}}} q$, which concludes the proof.

4) Let us consider the product of automata $\overline{\mathcal{A}_{\mathcal{S}}}$ and $\mathcal{R}$, and let $\mathcal{D}$ be the left automaton obtained by the usual powerset construction of automata over finite words. This construction preserves aperiodicity, and is in ExpTime. We only have left to show that we can obtain a bimachine $\mathcal{B}$ realizing f with $\mathcal{D}$ and $\mathcal{R}$ as automata. Indeed, if it is the case and $\mathcal{D}$ is aperiodic, we have according to Th. 2.2.11 that $\mathcal{D} \sqsubseteq \mathcal{L}_f(\mathcal{R})$ which means that $\mathcal{B}$ is aperiodic *a fortiori*.

Let $\mathcal{S} = (\mathcal{A}_{\mathcal{S}}, i_{\mathcal{S}}, o_{\mathcal{S}})$ with $A_{\mathcal{S}} = (Q_{\mathcal{S}}, \Delta_{\mathcal{S}}, \{q_0\})$. Let us define the output functions of $\mathcal{B} := (\mathcal{D}, \mathcal{R}, i_{\mathcal{B}}, o_{\mathcal{B}})$. Let $i_{\mathcal{B}}([x]_{\mathcal{R}}) := i_{\mathcal{S}}(q_0)$. Let $P := \{(p_1, [x_1]_{\mathcal{R}}), \dots, (p_n, [x_n]_{\mathcal{R}})\}$ be a state of $\mathcal{D}$, let $a \in A$ and let $x \in A^\omega$, we define $o_{\mathcal{B}}(P, a, [x]_{\mathcal{R}}) := o_{\mathcal{S}}(p_i, (a, [x]_{\mathcal{R}}), q)$ such that $x_i \approx_{\mathcal{R}} ax$ and $(p_i, (a, [x]_{\mathcal{R}}), q) \in \delta_{\mathcal{S}}$. Let us show that $o_{\mathcal{B}}$ is well-defined. Let $i, j \in \mathbf{n}$, such that $x_i = x_j = ax$. This means that there exists a word which can reach both p_i and p_j in $\overline{\mathcal{A}_{\mathcal{S}}}$ and that there is a final run over x from both p_i and p_j. Since $\mathcal{A}_{\mathcal{S}}$ is unambiguous, we have $p_i = p_j$. Furthermore, since $\mathcal{A}_{\mathcal{S}}$ is deterministic, the state q is uniquely defined. One can easily see that the outputs of $\mathcal{B}$ exactly match those of $\mathcal{S}$, which means that $\mathcal{B}$ realizes f. ☐

Let us now show a result about aperiodicity which will prove useful later.

Proposition 3.2.19. *Let f be an aperiodic ω-function, and let $\mathcal{R}$ be an aperiodic right automaton recognizing* $\mathrm{dom}(f)$*. Then $f_{\mathcal{R}}$ is aperiodic.*

Proof. If $\mathcal{R}$ is aperiodic, then $\ell(\mathcal{R})$ is also aperiodic. Let us show that $[\![\ell(\mathcal{R})]\!]^{-1}$ is aperiodic, this will conclude the proof since aperiodic functions are closed under composition (see Sec. 3.3). Let $\ell(\mathcal{R})'$ be the transducer realizing $[\![\ell(\mathcal{R})]\!]^{-1}$, obtained by exchanging the input and output labels of transitions of $\mathcal{R}$. Let $\mathcal{R}'$ denote its transitions, such that we have $\Delta_{\mathcal{R}'} = \{([ax]_{\mathcal{R}}, (a, [x]_{\mathcal{R}}), [x]_{\mathcal{R}}) \mid a, \in A, x \in A^\omega\}$. Let $n \in \mathbb{N}$ such that $\forall u \in A^*$, $u^{n+1} \equiv_{\mathcal{R}} u^n$. Let $u \in (A \times Q_{\mathcal{R}})^*$, and let $\pi : (A \times Q_{\mathcal{R}})^* \to A^*$ denote the natural projection. Let $k \geq n$ and let $\left[\pi(u)^k x\right]_{\mathcal{R}} \xrightarrow{u^k} [x]_{\mathcal{R}}$ denote a run of $\mathcal{R}'$ over u^k. Since we have $\left[\pi(u)^k x\right] = [\pi(u)^n x]$, we know that $[\pi(u)^n x]_{\mathcal{R}} \xrightarrow{u} [\pi(u)^n x]_{\mathcal{R}}$, and thus $\left[\pi(u)^k x\right]_{\mathcal{R}} \xrightarrow{u^{k+1}} [x]_{\mathcal{R}}$, which concludes the proof. □

We now obtain the result linking the existence of a bimachine realizing a function f with right automaton $\mathcal{R}$, with the sequentiality of $f_{\mathcal{R}}$.

Theorem 3.2.20. *Let f be a rational ω-transduction and let $\mathcal{R}$ be a right automaton recognizing* $\mathrm{dom}(f)$*. There exists a bimachine $\mathcal{B}$ realizing f with right automaton $\mathcal{R}$ if and only if $f_{\mathcal{R}}$ is sequential. Furthermore, if $\mathcal{R}$ is aperiodic, then f is aperiodic if and only if $f_{\mathcal{R}}$ is.*

Proof. This is a consequence of Prop. 3.2.16 and Lem. 3.2.18. The aperiodicty of $f_{\mathcal{R}}$ follows from Prop. 3.2.19. □

3.2.4 Delay look-ahead

The delay congruence of a function over finite words was introduced in [RS91] and yields, as we have seen in Chap. 2, the coarsest right automaton possible among all bimachines realizing the function. However the definition of delay congruence does not apply to functions producing infinite words. We rather use the alternative definition from [BLN12], which is equivalent to the one from [RS91] in the finite case but also applies to the infinite case. We show that this delay congruence has the same minimality property as in the finite case, but is only fine enough to make the function *quasi*-sequential and not sequential, in general.

Delay congruence

Definition 3.2.21. Let $f : A^\omega \to B^\infty$ be a function, the *left delay congruence* of f is defined for $x, y \in A^\omega$ by $x \overset{\vartriangle}{\approx}_f y$ if:

- $\forall u \in A^*$, $ux \in \mathrm{dom}(f) \Leftrightarrow uy \in \mathrm{dom}(f)$
- $|\{\mathsf{del}(f(ux), f(uy)) \mid ux \in \mathrm{dom}(f)\}| < \infty$

First we show that this left congruence is coarser than any left transition congruence of a machine realizing a given function.

Theorem 3.2.22. *Let $\mathcal{T}$ (resp. $\mathcal{B}$) be a transducer (resp. bimachine) with underlying automaton $\mathcal{A}$ (resp. right automaton $\mathcal{R}$) realizing a rational ω-function f. Then $\approx_{\mathcal{A}} \sqsubseteq \overset{\vartriangle}{\approx}_f$ (resp. $\approx_{\mathcal{R}} \sqsubseteq \overset{\vartriangle}{\approx}_f$).*

Proof. We give the proof for a bimachine, the proof for a transducer can be obtained in the same way. Let $\mathcal{B} = (\mathcal{L}, \mathcal{R}, i, o)$ be a bimachine and let $x \approx_{\mathcal{R}} y$, we want to show that $x \overset{\vartriangle}{\approx}_f y$. Since $\approx_{\mathcal{R}}$ recognizes $\mathrm{dom}(f)$ we have $\forall u \in A^*$, $ux \in \mathrm{dom}(f) \Leftrightarrow uy \in \mathrm{dom}(f)$.

Let $u \in A^*$ and let $l_0 \xrightarrow{u}_{\mathcal{L}} l$ denote the initial run of $\mathcal{L}$ over u. Since $x \approx_{\mathcal{R}} y$ we have that $\mathsf{del}(f(ux), f(uy)) = \mathsf{del}(o(l, x), o(l, y))$, which only depends on the state l. Thus we obtain $|\{\mathsf{del}(f(ux), f(uy)) \mid ux \in \mathrm{dom}(f)\}| \leq |\mathcal{L}|$, and hence $x \overset{\vartriangle}{\approx}_f y$. □

Proposition 3.2.23. *Let $\mathcal{B}$ be a bimachine realizing a rational ω-function f. Then one can compute $\overset{\triangle}{\approx}_f$ in* PTime.

Proof. Let $\mathcal{B} = (\mathcal{L}, \mathcal{R}, i, o)$ be a bimachine with left and right automata $\mathcal{L}$ and $\mathcal{R}$, respectively. According to Th. 3.2.22, we know that $\approx_{\mathcal{R}} \sqsubseteq \overset{\triangle}{\approx}_f$. Let $[u_1 v_1^\omega]_{\mathcal{R}}, [u_2 v_2^\omega]_{\mathcal{R}}$ be two states of $\mathcal{R}$, we can choose u_1, v_1, u_2, v_2 of size linear in $Q_{\mathcal{R}}$, and we write $x_i = u_i v_i^\omega$, $i \in \mathbf{2}$. Thus we only have to show how to decide if $x_1 \overset{\triangle}{\approx}_f x_2$ in PTime. First we can easily check that $\forall u,\ ux_1 \in \mathrm{dom}(f) \Leftrightarrow ux_2 \in \mathrm{dom}(f)$. Now we want to check whether the set $\{\mathsf{del}(f(ux_1), f(ux_2)) \mid ux_1 \in \mathrm{dom}(f)\}$ is finite. For any word $x \in A^\omega$ we define the function $f_x : u \mapsto f(ux)$. From the bimachine $\mathcal{B}$, we define for any $x \in A^\omega$, a transducer $\mathcal{T}_x = (\mathcal{A}_x, i, o, t)$ over finite words. The underlying automaton $\mathcal{A}_x$ is obtained by taking the product of $\mathcal{L}$ and $\mathcal{R}$ and setting as final states the states (l, r) such that r is the state from which $\mathcal{R}$ has a final run over x. The initial output function and the output function are obtained naturally as in Th. 3.2.12, whereas the final function always outputs ϵ. Let g_x denote the function realized by $\mathcal{T}_x$. We have that $f_x(u) = g_x(u) \cdot o([u]_{\mathcal{L}}, x)$. Thus deciding whether $x_1 \overset{\triangle}{\approx}_f x_2$ amounts to deciding if $\{\mathsf{del}(g_{x_1}(u) \cdot o([u]_{\mathcal{L}}, x_1), g_{x_2}(u) \cdot o([u]_{\mathcal{L}}, x_2)) \mid u \in A^*\}$ is finite.

We define a twinning-like property which will be equivalent to having a finite set of delays $\{\mathsf{del}(g_{x_1}(u) \cdot o([u]_{\mathcal{L}}, x_1), g_{x_2}(u) \cdot o([u]_{\mathcal{L}}, x_2)) \mid u \in A^*\}$. We say that a state p of a transducer $\mathcal{T}$ is *γ-constant* if there exists a word δ such that for any final run outputting α from p we have $\alpha\gamma = \delta$. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two transducers with the same domain and let γ_1, γ_2 be two rational words. We say that $(\mathcal{T}_1, \gamma_1)$ and $(\mathcal{T}_2, \gamma_2)$ satisfy the *weak adjacency twinning property* if for any pair of initial runs $p_1 \xrightarrow{u|\alpha_1}_{\mathcal{T}_1} q_1 \xrightarrow{v|\beta_1}_{\mathcal{T}_1} q_1$ and $p_2 \xrightarrow{u|\alpha_2}_{\mathcal{T}_2} q_2 \xrightarrow{v|\beta_2}_{\mathcal{T}_2} q_2$ such that there is a word w with a final run from both q_1 and q_2 the following hold:

- if q_1, q_2 are not γ_1, γ_2-constant, respectively, then we have that $\mathsf{del}(i(p_1)\alpha_1, i(p_2)\alpha_2) = \mathsf{del}(i(p_1)\alpha_1\beta_1, i(p_2)\alpha_2\beta_2)$
- if q_i is not γ_i-constant and p_j is γ_j-constant, then we have $\mathsf{del}(i(p_i)\alpha_i, i(p_j)\alpha_j\gamma_j) = \mathsf{del}(i(p_i)\alpha_i\beta_i, i(p_j)\alpha_j\gamma_j)$ for $\{i, j\} = \mathbf{2}$

Claim: $\{\mathsf{del}(\llbracket \mathcal{T}_1(u) \rrbracket \gamma_1, \llbracket \mathcal{T}_2(u) \rrbracket \gamma_2)\}$ is finite if and only if (T_1, γ_1) and (T_2, γ_2) satisfy the weak adjacency twinning property.

Let us show this claim. Assume that $\mathcal{T}_1$ and $\mathcal{T}_2$ do not satisfy the weak adjacency twinning property. Then let $p_1 \xrightarrow{u|\alpha_1}_{\mathcal{T}_1} q_1 \xrightarrow{v|\beta_1}_{\mathcal{T}_1} q_1$ and $p_2 \xrightarrow{u|\alpha_2}_{\mathcal{T}_2} q_2 \xrightarrow{v|\beta_2}_{\mathcal{T}_2} q_2$ be two initial runs such that there is a word w with a final run from both q_1 and q_2 and let us assume that p_i is not γ_i-constant for all $i \in \mathbf{2}$. Then we have $\mathsf{del}(i(p_1)\alpha_1, i(p_2)\alpha_2) \neq \mathsf{del}(i(p_1)\alpha_1\beta_1, i(p_2)\alpha_2\beta_2)$. Since p_2 is not γ_2-constant , this means that we can choose two final runs from p_2 producing μ, ν such that $\mu\gamma_2 \wedge \nu\gamma_2$ is a finite word. This means that either γ_2 is finite or there is a mismatch between $\mu\gamma_2$ and $\nu\gamma_2$. Either way we obtain

$$\left|\left\{\mathsf{del}(i(p_1)\alpha_1\beta_1^n\delta_1\gamma_1, i(p_2)\alpha_2\beta_2^n\delta_2\gamma_2) \mid n \in \mathbb{N},\ q_i \xrightarrow{w|\delta_i}_{\mathcal{T}_i} s_i \text{ final}\right\}\right| = \infty$$

Let us now assume that p_1 is not γ_1-constant and p_2 is γ_2-constant, and $\mathsf{del}(i(p_1)\alpha_1, i(p_2)\alpha_2\gamma_2) \neq \mathsf{del}(i(p_1)\alpha_1\beta_1, i(p_2)\alpha_2\gamma_2)$. Using the same reasoning, we also obtain that either γ_1 is finite or we can find two final runs which yield a mismatch, and thus we obtain an infinite set of delays. Conversely, let us assume that T_1 and T_2 satisfy the weak adjacency twinning property. This basically means that over synchronized loops, the delay between $\mathcal{T}_1$ and $\mathcal{T}_2$ cannot increase. For any word, one can decompose its runs over $\mathcal{T}_1$ and T_2 into small parts of bounded size and a bounded number of synchronized loops. Overall, the number of delays must stay bounded.

Now we only have left to show that one can decide the weak adjacency twinning property in PTime. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two transducers and let γ_1 and γ_2 be two output words. This is done again using the pattern logic of [FMR18]. For each pair of states q_1 of $\mathcal{T}_1$ and q_2 of $\mathcal{T}_2$, such that q_i is not γ_i constant for all $i \in \mathbf{2}$ and both states are co-reachable by some common word, we consider the transducer obtained by union of the two transducer $\mathcal{T}_1, T_2$ and with final states q_1 and q_2. The we check in PTime if this transducer satisfies the twinning property. If it does then it means in particular that there are no pairs of runs $p_1 \xrightarrow{u|\alpha_1}_{\mathcal{T}_1} q_1 \xrightarrow{v|\beta_1}_{\mathcal{T}_1} q_1$ and $p_2 \xrightarrow{u|\alpha_2}_{\mathcal{T}_2} q_2 \xrightarrow{v|\beta_2}_{\mathcal{T}_2} q_2$ with $\mathsf{del}(i(p_1)\alpha_1, i(p_2)\alpha_2) \neq \mathsf{del}(i(p_1)\alpha_1\beta_1, i(p_2)\alpha_2\beta_2)$ which concludes the proof. Conversely, if the transducer does not satisfy the twinning property, then there are two runs $p_1 \xrightarrow{u|\alpha_1}_{\mathcal{T}_1} q_1' \xrightarrow{v|\beta_1}_{\mathcal{T}_1} q_1'$ and $p_2 \xrightarrow{u|\alpha_2}_{\mathcal{T}_2} q_2' \xrightarrow{v|\beta_2}_{\mathcal{T}_2} q_2'$ with $\mathsf{del}(i(p_1)\alpha_1, i(p_2)\alpha_2) \neq \mathsf{del}(i(p_1)\alpha_1\beta_1, i(p_2)\alpha_2\beta_2)$. Since q_i' can reach q_i, this means that q_i' is not γ_i constant for all $i \in \mathbf{2}$ and thus $(\mathcal{T}_1, \gamma_1)$ and $(\mathcal{T}_2, \gamma_2)$ do *not* satisfy the WATP.

Let q_1 be a non γ_1-constant state of $\mathcal{T}_1$ and let q_2 be a γ_2-constant state of $\mathcal{T}_2$, such that both states are co-reachable for some common word. Again we consider the union of $\mathcal{T}_1$ and $\mathcal{T}_2$ with final states q_1 q_2. We also modify the outputs from q_2 such that it produces γ_2 which does not change the final value since q_2 is γ_2 constant. This time we do not check for the twinning property but only for absence of a mismatch. Again if we do not find a mismatch then it means that there are no pairs of runs $p_1 \xrightarrow{u|\alpha_1}_{\mathcal{T}_1} q_1 \xrightarrow{v|\beta_1}_{\mathcal{T}_1} q_1$ and $p_2 \xrightarrow{u|\alpha_2}_{\mathcal{T}_2} q_2 \xrightarrow{v|\beta_2}_{\mathcal{T}_2} q_2$ such that $\mathsf{del}(i(p_1)\alpha_1, i(p_2)\alpha_2\gamma_2) \neq \mathsf{del}(i(p_1)\alpha_1\beta_1, i(p_2)\alpha_2\gamma_2)$. Conversely, if there exists a mismatch for a different pair of states, then *a fortiori* $(\mathcal{T}_1, \gamma_1)$ and $(\mathcal{T}_2, \gamma_2)$ do not satisfy the WATP.

Thus by checking properties for each pair of states of $\mathcal{T}_1, \mathcal{T}_2$ we can decide if $(\mathcal{T}_1, \gamma_1)$ and $(\mathcal{T}_2, \gamma_2)$ satisfy the WATP in PTime. $\square$

Example 3.2.24. We consider the functions defined in Ex. 3.1.1.

Since $\mathsf{del}(f_{\text{erase}}(ux), f_{\text{erase}}(uy)) = \mathsf{del}(f_{\text{erase}}(x), f_{\text{erase}}(y))$ then we have that $\overset{\triangle}{\approx}_{f_{\text{erase}}}$ is trivial, and the same goes for $f_{\#a}$.

The equivalence classes of the delay congruence of f_{blocks} depend on the first letter of the words. Indeed, let $x = (\#aa)^\omega$, then $\mathsf{del}(f(a^n ax), f(a^n bx)) = (aa^n x, ba^n x)$ for any positive integer n which means that $ax \overset{\triangle}{\not\approx}_{f_{\text{blocks}}} bx$.

We show in the two following lemmas that the right automata $\mathcal{R}$ fine enough to have $f_{\mathcal{R}}$ quasi-sequential are exactly those finer than $\overset{\triangle}{\approx}_f$.

Lemma 3.2.25. *Let f be a rational ω-function and let $\mathcal{R}$ be a right automaton recognizing* $\mathrm{dom}(f)$. *If $\approx_{\mathcal{R}} \sqsubseteq \overset{\triangle}{\approx}_f$ then $f_{\mathcal{R}}$ is quasi-sequential.*

Proof. Let $f : A^\omega \to B^\infty$ be a rational ω-function and let $\mathcal{R}$ be a right automaton recognizing $\mathrm{dom}(f)$, such that $\approx_{\mathcal{R}} \sqsubseteq \overset{\triangle}{\approx}_f$. Let $\mathcal{T}$ be a transducer realizing f. Let us consider two initial runs $p_1 \xrightarrow{u|\alpha_1} q_1 \xrightarrow{v|\beta_1} q_1$ and $p_2 \xrightarrow{u|\alpha_2} q_2 \xrightarrow{v|\beta_2} q_2$. Up to taking $u' = uv$, we can assume that u and v have the same last letter (a, r), without loss of generality. Let $\pi : (A \times S_{\mathcal{R}})^\infty \to A^\infty$ denote the natural projection, such that $\pi \circ [\![\ell(\mathcal{R})]\!] = Id$. Let $q_1 \xrightarrow{x_1|\gamma_1} P_1$ and $q_2 \xrightarrow{x_2|\gamma_2} P_2$ denote final runs, we must have that $\pi(x_1) \approx_{\mathcal{R}} \pi(x_2)$ since the two words have a final run from r. Let us first consider the case where p_1, p_2 are non-constant, and we assume towards a contradiction that we have $\mathsf{del}(i(p_1)\alpha_1, i(p_2)\alpha_2) \neq \mathsf{del}(i(p_1)\alpha_1\beta_1, i(p_2)\alpha_2\beta_2)$. Since p_1, p_2 are non constant, we can choose x_i such that $\beta_i^\omega \neq \gamma_i$, for $i \in \mathbf{2}$. This means that the set $\{\mathsf{del}(i(p_1)\alpha_1\beta_1^n\gamma_1, i(p_2)\alpha_2\beta_2^n\gamma_2) \mid n \in \mathbb{N}\}$ is infinite and thus $\pi(x_1) \not\approx_{\mathcal{R}} \pi(x_2)$, which yields a contradiction. Let us now assume that p_1 is non constant and p_2 is constant. We assume that $\beta_1 \neq \epsilon$, and $i(p_1)\alpha_1\beta_1^\omega \neq i(p_2)\alpha_2\gamma_2$.

Since q_1 is non-constant, we can choose x_1 such that $\beta_1^\omega \neq \gamma_1$, and thus we obtain that the set $\{\mathsf{del}(i(p_1)\alpha_1\beta_1^n\gamma_1, i(p_2)\alpha_2\gamma_2) \mid n \in \mathbb{N}\}$ is infinite. Finally we conclude that $\pi(x_1) \not\approx_{\mathcal{R}} \pi(x_2)$, which yields a contradiction, thus $\mathcal{T}$ does satisfy the WTP. □

Lemma 3.2.26. *Let f be a rational ω-function and let $\mathcal{R}$ be a right automaton recognizing* $\mathrm{dom}(f)$*. If $f_{\mathcal{R}}$ is quasi-sequential then $\approx_{\mathcal{R}} \sqsubseteq \overset{\vartriangle}{\approx}_f$.*

Proof. Let $f : A^\omega \to B^\infty$ be a rational ω-function and let $\mathcal{R}$ be a right automaton recognizing $\mathrm{dom}(f)$. Let $\mathcal{T}$ be a transducer realizing $f_{\mathcal{R}}$, and let us assume that $\mathcal{T}$ satisfies the WTP. Let $x_1, x_2 \in A^\omega$ such that $x_1 \approx_{\mathcal{R}} x_2$, our goal is to show that $x_1 \overset{\vartriangle}{\approx}_f x_2$. Since $\mathcal{R}$ recognizes $\mathrm{dom}(f)$, we have $\forall u$, $ux_1 \in \mathrm{dom}(f) \Leftrightarrow ux_2 \in \mathrm{dom}(f)$. Let us consider two initial runs of $\mathcal{T}$ over $u = u_1u_2u_3$ with a simultaneous loop, $p_1 \xrightarrow{u_1|\alpha_1} q_1 \xrightarrow{u_2|\beta_1} q_1 \xrightarrow{u_3|\gamma_1} r_1$ and $p_2 \xrightarrow{u_1|\alpha_2} q_2 \xrightarrow{u_2|\beta_2} q_2 \xrightarrow{u_3|\gamma_2} r_2$, such that there is a final run from r_1 over x_1 producing δ_1 and a final run from r_2 over x_2 producing δ_2.

If q_1 and q_2 are constant, then we have $\mathsf{del}(f(ux_1), f(ux_2)) = \mathsf{del}(\alpha_1\delta_1, \alpha_2\delta_2)$. If q_1 and q_2 are non-constant, then by WTP, we have $\mathsf{del}(f(ux_1), f(ux_2)) = \mathsf{del}(\alpha_1\gamma_1\delta_1, \alpha_2\gamma_2\delta_2)$. If q_1 is non-constant and q_2 is constant then $\mathsf{del}(f(ux_1), f(ux_2)) = \mathsf{del}(\alpha_1\gamma_1\delta_1, \alpha_2\delta_2)$. In all cases, we see that we can remove the simultaneous loops of u without changing the delays, and thus we obtain that $|\{\mathsf{del}(f(ux_1), f(ux_2)) \mid u \in A^*\}| \leq |A|^{(k+1)|Q|^2}$ where k is the size of the longest output of $\mathcal{T}$. □

Theorem 3.2.27. *Let f be a rational ω-function and let $\mathcal{R}$ be a right automaton recognizing* $\mathrm{dom}(f)$*. Then $f_{\mathcal{R}}$ is quasi-sequential if and only if $\approx_{\mathcal{R}} \sqsubseteq \overset{\vartriangle}{\approx}_f$.*

Proof. This is a consequence of the two previous results, Lem. 3.2.25 and 3.2.26. □

3.2.5 The ultimate look-ahead and a canonical bimachine for quasi-sequential functions

The look-ahead given by the delay congruence is only fine enough to make a function quasi-sequential. We define a second canonical left congruence, the *ultimate left congruence*, which will be fine enough to make any quasi-sequential function sequential. Although it is defined in a machine independent way, this congruence does not share the same minimality property as the delay congruence. Nevertheless, combining the two look-aheads we are able to define a canonical bimachine for any rational ω-transduction.

Ultimate left congruence

Definition 3.2.28. The *ultimate left congruence* of a function $f : A^\omega \to B^\infty$ is defined by setting $x \overset{\cup}{\approx}_f y$ if $\forall u \in A^*$:

- $ux \in \mathrm{dom}(f) \Leftrightarrow uy \in \mathrm{dom}(f)$
- if $ux \in \mathrm{dom}(f)$ then 1) $\widehat{f}(u) = \overline{f}(ux) \Leftrightarrow \widehat{f}(u) = \overline{f}(uy)$ and 2) if $\widehat{f}(ux) = \overline{f}(ux)$ then $f(ux) = f(vx)$

Observe that $\widehat{f}(u) \leq \overline{f}(ux)$, so the intuition for $\widehat{f}(u) = \overline{f}(ux)$ is that nothing more can be output sequentially after reading u over ux. The idea behind point 2) is that the missing outputs $\overline{f}(ux)^{-1}f(ux)$ and $\overline{f}(uy)^{-1}f(uy)$ should be equal, which means that $f(ux) = f(uy)$ from point 1). Intuitively, a sequential transducer with the look-ahead given by $\overset{\cup}{\approx}_f$ would have enough information to produce the missing output, and know when to start producing it.

Proposition 3.2.29. *Let $f : A^\omega \to B^\infty$ be a function, $\overset{\cup}{\approx}_f$ is a left congruence.*

Proof. Let $f : A^\omega \to B^\infty$ be a function, let $x \overset{\cup}{\approx}_f y$ and let $a \in A$, our goal is to show that $ax \overset{\cup}{\approx}_f ay$. Let $u \in A^*$, of course we have that $uax \in \mathrm{dom}(f) \Leftrightarrow uay \in \mathrm{dom}(f)$. Let us assume that $uax \in \mathrm{dom}(f)$, we know that $\widehat{f}(ua) = \overline{f}(uax) \Leftrightarrow \widehat{f}(ua) = \overline{f}(uay)$. If $\widehat{f}(u) = \widehat{f}(ua)$ then we have 1) $\widehat{f}(u) = \overline{f}(uax) \Leftrightarrow \widehat{f}(u) = \overline{f}(uay)$. In the case where $\widehat{f}(u) = \overline{f}(uax)$, then we have $\widehat{f}(ua) = \overline{f}(uax)$ and thus $f(uax) = f(uay)$ which proves 2). If $\widehat{f}(u) < \widehat{f}(ua)$, then we know that $\widehat{f}(u) \neq \overline{f}(uax)$ and $\widehat{f}(u) \neq \overline{f}(uay)$, and thus we trivially obtain 1) and 2). Hence we have $ax \overset{\cup}{\approx}_f ay$, which concludes the proof. □

Example 3.2.30. Recall the function $f_{\#\mathrm{a}}$, which maps words with an infinite number of a letters to a^ω and other words to b^ω, over the alphabet $A = \{a, b\}$. One can see that the ultimate left congruence of $f_{\#\mathrm{a}}$ contains two classes, $(b^*a)^\omega$ and A^*b^ω.

Theorem 3.2.31. *Let f be a quasi-sequential function, the ultimate left congruence of f has finite index. If f is given as a bimachine, one can compute $\overset{\cup}{\approx}_f$ in 2*ExpTime*. Furthermore, if f is aperiodic, then so is $\overset{\cup}{\approx}_f$.*

Proof. Let $\mathcal{B}$ be a bimachine with automata $\mathcal{L}$ and $\mathcal{R}$ realizing a quasi-sequential function $f : A^\omega \to B^\infty$. Let $\mathcal{S}$ be the sequential transducer obtained by powerset construction from the transducer with underlying automaton $\mathcal{L} \times \mathcal{R}$ (Prop. 3.2.2). Since f is quasi-sequential we know that $\mathcal{S}$ is finite (Th. 3.1.25), furthermore $\mathcal{S}$ realizes $\overline{f}$ (Prop. 3.1.21), and has exponential size with respect to $\mathcal{B}$. Our approach here is to define a left congruence $\approx$ such that $\approx \sqsubseteq \overset{\cup}{\approx}_f$, show that $\approx$ has a doubly exponential size with respect to $\mathcal{B}$ and show that if $\mathcal{B}$ is aperiodic then so is $\approx$. Finally, we show how to obtain $\overset{\cup}{\approx}_f$ from $\approx$.

Let $x, y \in A^\omega$, we set $x \approx y$ if for any state (r, P) of $\mathcal{R} \times \mathcal{S}$ the two following conditions hold:

- the final runs over x and y from (r, P) visit the same states infinitely often
- the run of $\mathcal{S}$ from P produces ϵ over x if and only if it produces ϵ over y

Let us show that $\approx$ is a left congruence. Let $x \approx y$, let $a \in A$ and let (r, P) be a state of $\mathcal{R} \times \mathcal{S}$. If there exists a state r' of $\mathcal{R}$ such that (r, a, r') is a transition of $\mathcal{R}$, and x, y both have final runs from r', then let P' be the successor state of P by reading a in $\mathcal{S}$, and we have that the final runs from (r, P) over ax, ay visit infinitely often the same states as the final runs from (r', P') over x, y, and hence we obtain 1). Now let us assume that the runs from P' over x, y don't produce ϵ. Then the runs from P over ax, ay don't produce ϵ either. If the runs from P' do produce ϵ, then the runs from P over ax, ay produce ϵ if and only if the output of the transition (P, a, P') is equal to ϵ, which proves 2).

Let $x \approx y$, we want to show that $x \overset{\cup}{\approx}_f y$. Since x, y are equivalent with respect to $\mathcal{R}$, we know that $\forall u,\ ux \in \mathrm{dom}(f) \Leftrightarrow uy \in \mathrm{dom}(f)$. Let $u \in A^*$ such that $ux \in \mathrm{dom}(f)$. If $\widehat{f}(u)$ is infinite then $\widehat{f}(u) = \overline{f}(ux) = f(ux) = f(uy) = \overline{f}(uy) = \widehat{f}(u)$. Let us assume that $\widehat{f}(ux)$ is finite, then by definition of $\mathcal{S}$, we have $i_{\mathcal{S}}(S_0)\alpha = \widehat{f}(u)$, where $S_0 \xrightarrow{u|\alpha}_{\mathcal{S}} P$ denotes the initial run of $\mathcal{S}$ over u. If $\widehat{f}(u) < \overline{f}(ux)$, then the run from P over x has to produce something, and thus the run from P over y also produces something and we have $\widehat{f}(u) < \overline{f}(uy)$. Let us assume $\widehat{f}(u) = \overline{f}(ux) = \overline{f}(uy)$ and let (r, S) be a state appearing infinitely often in the final runs from $([x]_{\mathcal{R}}, P)$ over x, y. By determinism of $\mathcal{L}$, there must be one state l such that $((l, r), \beta) \in S$. Thus we obtain that $f(ux) = i_{\mathcal{S}}(S_0)\alpha\beta = f(uy)$, which means that $x \overset{\cup}{\approx}_f y$.

One can see that the index of $\approx$ is exponential in the size of $\mathcal{S}$ and thus doubly exponential in $\mathcal{B}$. Now that we have $\approx \sqsubseteq \overset{\cup}{\approx}_f$ we show how to decide given x, y (rational) whether $x \overset{\cup}{\approx}_f y$. Let P be a state of $\mathcal{S}$, and let $\mathrm{rest}(x, R)$ denote the word w if x produces ϵ from R and the missing output is w, and $\perp$ otherwise. Then one can see that $x \overset{\cup}{\approx}_f y$ if and only if $\forall R$, $\mathrm{rest}(x, R) = \mathrm{rest}(y, R)$.

In order to show that $\overset{\cup}{\approx}_f$ is aperiodic, we only have to show that $\approx$ is aperiodic, given that $\mathcal{B}$ is aperiodic. By assumption, $\mathcal{R}$ is aperiodic, and from Th. 3.1.26, we know that $\mathcal{S}$ is also aperiodic. Let P, R be states of $\mathcal{S}$, let $L_{P,R}$ denote the language of finite words that have a run from P to R, and let L_R denote the language of infinite words that have a final run (*i.e.* a run since all states of $\mathcal{S}$ are accepting) from R. Since $\mathcal{S}$ is aperiodic, these languages are aperiodic. Let $R \xrightarrow{a|\alpha}_{\mathcal{S}} S$ be a producing transition such that $\alpha \neq \epsilon$. Then the language $L_P \cdot (L_{P,R} \cdot a \cdot L_S)^c$ of words which have a final run from P and never use this producing transition is aperiodic. Since aperiodic languages are closed under concatenation and complement, this language is aperiodic. Furthermore, since aperiodic languages are closed under intersection, then the language of words which produce ϵ from P is aperiodic. Furthermore, let us consider $\mathcal{R} \times \mathcal{S}$, we can fix any set of set of states as Muller acceptance condition, the obtained automaton is still aperiodic. Thus we have that any congruence class of $\approx$ is aperiodic, which concludes the proof. $\square$

Canonical bimachine for quasi-sequential functions Let $f : A^\omega \to B^\infty$ be a quasi-sequential function, and let $\mathcal{R}$ be a right automaton recognizing $\mathrm{dom}(f)$ such that $\approx_{\mathcal{R}} \sqsubseteq \overset{\cup}{\approx}_f$. We define the bimachine $\mathcal{U}_f(\mathcal{R}) := (\mathcal{A}_f, \mathcal{R}, i_f, o_{\mathcal{R}})$, with $\mathcal{A}_f$ and i_f defined as in Sec. 3.1, by:

$$o_{\mathcal{R}}([u]_f, a, [x]_{\mathcal{R}}) = \left[\begin{array}{ll} \widehat{f}(u)^{-1}\widehat{f}(ua) & \text{if } \widehat{f}(ua) < \overline{f}(uax) \\ \alpha & \text{if } \widehat{f}(u) < \widehat{f}(ua) = \overline{f}(uax) \text{ and } \widehat{f}(u)^{-1}f(uax) = \alpha\beta^\omega \\ \beta & \text{if } \widehat{f}(u) = \overline{f}(uax) \text{ and } f(uax) = \alpha\beta^\omega \end{array}\right.$$

Remark 3.2.32. Let us note that given f quasi-sequential we have that $\mathcal{A}_f$ is finite from Th. 3.1.25, and thus $\mathcal{U}_f(\mathcal{R})$ is also finite.

Let us first show that the outputs of the above bimachine are well-defined.

Proposition 3.2.33. *Let $f : A^\omega \to B^\infty$ be a quasi-sequential function, let $\mathcal{R}$ be a right automaton recognizing* $\mathrm{dom}(f)$ *such that $\approx_{\mathcal{R}} \sqsubseteq \overset{\cup}{\approx}_f$. The outputs of $\mathcal{U}_f(\mathcal{R})$ are well-defined.*

Proof. Let $f : A^\omega \to B^\infty$ be a rational ω-function, let $\mathcal{R}$ be a right automaton recognizing $\mathrm{dom}(f)$ such that $\approx_{\mathcal{R}} \sqsubseteq \overset{\cup}{\approx}_f$. We want to show that the output function $o_{\mathcal{R}}$ does not depend on the choice of representatives. Let $u \sim_f v$, let $x \overset{\cup}{\approx}_f y$ and let $a \in A$. Let us first show that $o_{\mathcal{R}}([u]_f, a, [x]_{\mathcal{R}}) = o_{\mathcal{R}}([v]_f, a, [x]_{\mathcal{R}})$. If $\widehat{f}(u)$ is infinite, then $\widehat{f}(v)$ is also infinite and they have the same ultimate period. If $\widehat{f}(u)$ is finite, then we know that $\widehat{f}(v)$ is also finite and $\widehat{f}(u)^{-1}f(uax) = \widehat{f}(v)^{-1}f(vax)$ by definition of $\sim_f$. We also have by the proof of Prop. 3.1.9 that $\widehat{f}(u)^{-1}\widehat{f}(ua) = \widehat{f}(v)^{-1}\widehat{f}(va)$. In all cases we obtain $o_{\mathcal{R}}([u]_f, a, [x]_{\mathcal{R}}) = o_{\mathcal{R}}([v]_f, a, [x]_{\mathcal{R}})$.

Let us now show $o_{\mathcal{R}}([u]_f, a, [x]_{\mathcal{R}}) = o_{\mathcal{R}}([u]_f, a, [y]_{\mathcal{R}})$. If $\widehat{f}(ua) < \overline{f}(uax)$, then we have that $\widehat{f}(ua) < \overline{f}(uay)$ since $x \overset{\cup}{\approx}_f y$. If $\widehat{f}(u) < \widehat{f}(ua) = \overline{f}(uax)$, then we have $\widehat{f}(ua) = \overline{f}(uay)$ since $x \overset{\cup}{\approx}_f y$. This means that $f(uax) = f(uay)$ and thus in particular $\widehat{f}(u)^{-1}f(uax) = \widehat{f}(u)^{-1}f(uay)$. Finally, if $\widehat{f}(u) = \overline{f}(uax)$, then $\widehat{f}(u) = \overline{f}(uay)$ since $ax \overset{\cup}{\approx}_f ay$. Thus we have $f(uax) = f(uay)$ which proves $o_{\mathcal{R}}([u]_f, a, [x]_{\mathcal{R}}) = o_{\mathcal{R}}([u]_f, a, [y]_{\mathcal{R}})$. $\square$

Theorem 3.2.34. *Let $f : A^\omega \to B^\infty$ be a quasi-sequential function, and let $\mathcal{R}$ be a right automaton recognizing* $\mathrm{dom}(f)$ *such that $\approx_\mathcal{R} \sqsubseteq \overset{\cup}{\approx}_f$. Then the bimachine $\mathcal{U}_f(\mathcal{R})$ is finite and realizes f.*

Proof. The fact that $\mathcal{U}_f(\mathcal{R})$ is finite comes from Th. 3.1.25 which says that $\sim_f$ is finite. We only have to show that $\mathcal{U}_f(\mathcal{R})$ realizes f. Let g be the function realized by $\mathcal{U}_f(\mathcal{R})$, and let $x \in \mathrm{dom}(f)$. One can see that $\overline{f}(x) \leq g(x) \leq f(x)$. If $\overline{f}(x)$ is infinite, then of course $g(x) = f(x)$. If $\overline{f}(x)$ is finite, let $u < ua < x$ such that $\widehat{f}(u) < \widehat{f}(ua) = \overline{f}(x)$. Let $f(x) = \widehat{f}(u)\alpha\beta^\omega$, then the transition after reading u produces α and all other subsequent transitions produce β, by definition, and thus $g(x) = f(x)$. □

Remark 3.2.35. If we consider as right automaton $\mathcal{R}$ the automaton obtained by the construction of [CM03], we obtain $\mathcal{U}_f(\mathcal{R})$, a completely canonical bimachine for f quasi-sequential. However we can always left minimize this bimachine and thus a "more canonical" bimachine would be $\mathcal{B}_f(\mathcal{R})$. Note here that unlike in the finite case, our notion of canonicity depends on a particular construction to obtain a right automaton.

3.2.6 Composing look-aheads and a canonical bimachine

We defined one canonical look-ahead which makes any rational ω-function quasi-sequential, and one which makes any quasi-sequential function sequential. Here we show how to compose these look-aheads to obtain a canonical bimachine for any transduction. We also show that an aperiodic function has an aperiodic canonical bimachine.

Composing look-aheads Let us first make clear what we mean by composition of look-aheads. Let $\mathcal{R}_1 = (Q_1, \Delta_1, I_1, F_1)$ be a right automaton over an alphabet A. Let $\mathcal{R}_2 = (Q_2, \Delta_2, I_2, F_2)$ be a right automaton over the alphabet $A \times Q_1$. We define the Muller automaton $\mathcal{R}_1 \bowtie \mathcal{R}_2 := (Q_1 \times Q_2, \Delta_{1,2}, I_1 \times I_2, F_{1,2})$ by:

- $\Delta_{1,2} := \{(s_1, s_2), a, (r_1, r_2) \mid \ (s_1, a, r_1) \in \Delta_1 \text{ and } (s_2, (a, r_1), r_2) \in \Delta_2\}$
- $F_{1,2} := \{S \mid \ \{p \mid \ (p,q) \in S\} \in F_1, \ \{q \mid \ (p,q) \in S\} \in F_2\}$

Lemma 3.2.36. *Let $\mathcal{R}_1$ be an automaton with state space Q_1 over A and let Q_2 be an automaton over $A \times Q_1$. Then $[\![\ell(\mathcal{R}_2)]\!] \circ [\![\ell(\mathcal{R}_1)]\!] = [\![\ell(\mathcal{R}_1 \bowtie \mathcal{R}_2)]\!]$ (up to the isomorphism between $(A \times Q_1) \times Q_2$ and $A \times (Q_1 \times Q_2)$).*

Proof. Let $x \in A^\omega$, let r be the run of $\mathcal{R}_1 \bowtie \mathcal{R}_2$ over x. By definition of $\mathcal{R}_1 \bowtie \mathcal{R}_2$, the first component of r depends only on the run of $\mathcal{R}_1$ over x. Then we have that the second component of r is exactly the run of $\mathcal{R}_2$ over $[\![\ell(\mathcal{R}_1)]\!](x)$, which concludes the proof. □

Canonical bimachine We now have all the tools to show the main theorem of this chapter. In particular, this theorem shows that one can decide whether a given rational function over infinite words is aperiodic.

Theorem 3.2.37. *Let f be a rational ω-transduction given by a bimachine $\mathcal{B}$. Let $\mathcal{R}_1$ be the right automaton associated with $\overset{\triangle}{\approx}_f$ and let $\mathcal{R}_2$ be the right automaton associated with $\overset{\cup}{\approx}_{f_{\mathcal{R}1}}$. Then the bimachine $\mathcal{B}_f(\mathcal{R}_1 \bowtie \mathcal{R}_2)$ realizes f. Furthermore, if f is aperiodic, then $\mathcal{B}_f(\mathcal{R}_1 \bowtie \mathcal{R}_2)$ is aperiodic.*

Proof. Let f be a rational ω-transduction given by a bimachine $\mathcal{B}$. Let $\mathcal{R}_1$ be the right automaton obtained by the construction from [CM03], recognizing the delay congruence $\overset{\triangle}{\approx}_f$. Let $\mathcal{R}_2$ be the right automaton recognizing the ultimate congruence $\overset{\cup}{\approx}_{f_{\mathcal{R}1}}$. Since $\approx_{\mathcal{R}_1} \sqsubseteq \overset{\triangle}{\approx}_f$, we have according to Lem. 3.2.25 $f_{\mathcal{R}_1}$ is quasi-sequential. Hence since $\mathcal{R}_2$ is finer than $\overset{\cup}{\approx}_{f_{\mathcal{R}1}}$, we know from Th. 3.2.34 that $\mathcal{U}_{f_{\mathcal{R}1}}(\mathcal{R}_2)$ realizes $f_{\mathcal{R}_1}$. From Th. 3.2.20 we thus have that the function $(f_{\mathcal{R}_1})_{\mathcal{R}_2}$, obtained by composing the labelings of $\ell(\mathcal{R}_2)$ and $\ell(\mathcal{R}_1)$, is sequential. Using Lem. 3.1.17, we obtain that $f_{\mathcal{R}_1\bowtie\mathcal{R}_2}$ is sequential, and from Th. 3.2.20 we know that there exists a bimachine realizing f with $\mathcal{R}_1 \bowtie \mathcal{R}_2$ as right automaton. In particular we have that $\mathcal{B}_f(\mathcal{R}_1 \bowtie \mathcal{R}_2)$ realizes f.

If f is aperiodic, we have from Th. 3.2.22 that $\overset{\triangle}{\approx}_f$ is aperiodic, and from Th. 1.3.13 $\mathcal{R}_1$ is thus aperiodic, and from Th. 3.2.20 we have that $f_{\mathcal{R}_1}$ is aperiodic. From Th. 3.2.31, we have that $\overset{\cup}{\approx}_{f_{\mathcal{R}1}}$ is aperiodic and thus $\mathcal{R}_2$ is aperiodic (Th. 1.3.13). Again from Th. 3.2.20, $(f_{\mathcal{R}_1})_{\mathcal{R}_2} = f_{\mathcal{R}_1\bowtie\mathcal{R}_2}$ is aperiodic. A third time from Th. 3.2.20, we have that $\mathcal{B}_f(\mathcal{R}_1 \bowtie \mathcal{R}_2)$ is aperiodic, which concludes the proof. □

Remark 3.2.38. The complexity of the above construction is several folds exponential (a simple analysis gives 7ExpTime!). We strongly suspect that the complexity can be lowered quite a bit.

Also note that the automata of the canonical bimachine are Büchi automata.

3.3 First-order definability of transductions over infinite words

This section is less general than in the finite case. Indeed here we only focus on first-order-definable ω-transductions. We show the equivalence with the aperiodic ω-transductions, and thus from Th. 3.2.37 we can decide FO-definability of rational ω-functions. For completeness we also show that $\mathsf{FO}[\leq]$-transductions are closed under composition, which is unsurprising for transductions *à la Courcelle*.

3.3.1 Aperiocity and first-order definability

Theorem 3.3.1. *An ω-transduction is aperiodic if and only if it is realizable by an $\mathsf{FO}[\leq]$-transducer.*

Proof. We take the proof of Th. 1.5.8, and adapt it to the aperiodic case.

Let $\mathcal{T} = (\mathcal{A}, i, o)$ be an aperiodic transducer realizing a function $f : A^\omega \to B^\infty$, with underlying automaton $\mathcal{A} = (Q, \Delta, I, F)$. According to Th. 3.2.37 there is an aperiodic bimachine realizing f and from Prop. 3.2.2$\mathcal{T}$ can be assumed unambiguous. Let p, q be states of $\mathcal{A}$, let $L_{p,q} \subseteq A^*$ be the set of finite words which can go from p to q and let $L_q \subseteq A^\omega$ denote the set of infinite words having a final run from q. Since $L_{p,q}, L_q$ are aperiodic, there are $\mathsf{FO}[\leq]$-formulas $\phi_{p,q}, L_q$ recognizing them respectively. Given an $\mathsf{FO}[\leq]$-formula ϕ we define inductively the $\mathsf{FO}_\mathsf{c}[\leq]$-formula $\phi^{<\mathsf{c}}$ by restricting the quantifications to the positions before c. Formally, if $\phi = \exists y\ \psi(y)$ then we set $\phi^{<\mathsf{c}} := \exists y\ (y < \mathsf{c}) \wedge \psi^{<\mathsf{c}}(y)$. Atomic formulas and boolean connectives are left unchanged.

Let us define the $\mathsf{FO}[\leq]$-transducer $\mathcal{T}' = \left(K, \phi_{\text{dom}}, (\phi_v)_{v\in K}\right)$ by:

- $K := (\{\epsilon\} \cup i(I)) \cdot o(\Delta)$
- $\phi_{\text{dom}} := \bigvee_{p,q \in I\times F} \phi_{p,q}$

- $\phi_v := \left[\begin{array}{l} \bigvee_{i(p_0)o(p_0,a,q)=v} \min(\mathsf{c}) \wedge a(\mathsf{c}) \wedge \phi_q^{>\mathsf{c}} \\ \bigvee_{o(p,a,q)=v, p_0 \in I} \neg \min(\mathsf{c}) \wedge \phi_{p_0,p}^{<\mathsf{c}} \wedge a(\mathsf{c}) \wedge \phi_q^{>\mathsf{c}} \end{array} \right.$

By construction, and since $\mathcal{A}$ is unambiguous, we have that $\mathcal{T}'$ is an $\mathsf{FO}[\leq]$-transducer realizing f.

Conversely, let $\mathcal{T} = \left(K, \phi_{\text{dom}}, (\phi_v)_{v \in K}\right)$ be an $\mathsf{FO}[\leq]$-transducer realizing a function $f : A^\omega \to B^\infty$. From $\mathcal{T}$, we define an ω-language over the alphabet $A \times K$. Let ϕ be an $\mathsf{FO}_\mathsf{c}[\leq]$-formula, we syntactically define $\phi'(c)$ an $\mathsf{FO}[\leq]$-formula with one free variable. Each label predicate $a(x)$ is replaced syntactically by $\bigvee_{v \in K}(a, v)(x)$. Then each occurrence of the constant symbol c is replaced by a fresh free variable symbol c. For $v \in K$, we define a predicate $v(x) := \bigvee_{a \in A}(a, v)(x)$. We define the $\mathsf{FO}[\leq]$-formula:

$$\phi_\mathcal{T} := \phi'_{dom} \wedge \forall c \bigvee_{v \in K} v(c) \wedge \phi'_v(c)$$

The formula $\phi_\mathcal{T}$ recognizes words such that the projection over A^ω is in $\mathrm{dom}(f)$. Furthermore each position is labeled by its output. We define an aperiodic deterministic Muller automaton $\mathcal{A}_\mathcal{T} = (Q_\mathcal{T}, \Delta_\mathcal{T}, I, F)$ recognizing the same language as $\phi_\mathcal{T}$ (Th. 1.3.12). From this we naturally define a transducer $\mathcal{T}' := (\mathcal{A}, i, o)$ with $\mathcal{A} := (Q_\mathcal{T}, \Delta, I, F)$ by:

- $\Delta := \{(p, a, q) \mid \ (p, (a, v), q) \in \Delta_\mathcal{T}\}$
- $i : p \mapsto \epsilon$
- $o(p, a, q) := v$ such that $(p, (a, v), q) \in \Delta_\mathcal{T}$

Note that o is well-defined since f is a function. By construction $\mathcal{T}'$ realizes f.

We only have left to show that $\mathcal{A}$ is aperiodic. Let us first remark that $\mathcal{A}$ is unambiguous, otherwise some input word would have two different images. Let $u \in A^+$ be a finite word and let $p \xrightarrow{u^n|w}_\mathcal{A} p$ denote a run of $\mathcal{T}'$ over u^n. Let $p_0 \xrightarrow{u_1}_\mathcal{A} p$ be an initial run and let $p \xrightarrow{x}_\mathcal{A} P$ be a final run. Let $i \in \{1, \ldots, |u|\}$, $j \in \{0, \ldots, n-1\}$, and let $y_{i,j} := (u_1 u^{3n} x, |u_1||u|^{n+j} + i)$ denote the pointed word where the ith position in the jth middle occurrence of u is pointed. Since $u_1 u^{3n} x$ is in the domain of f, $\forall\ i \in \{1, \ldots, |u|\}$, $j \in \{0, \ldots, n-1\}$ there is a word $v_{i,j} \in K$ such that $y_{i,j} \models \phi_{v_{i,j}}$. Since $\mathcal{T}$ is first-order, if n is large enough, we have for any $j, j' \in \{0, \ldots, n-1\}$ that $v_{i,j} = v_{i,j'}$ (this can be obtained *via* an Ehrenfeucht-Fraïssé game argument). This means that we have $p \xrightarrow{s^n}_{\mathcal{A}_\mathcal{T}} p$ with $\pi_A(s) = u$, $\pi_B(s) = v$ and $v^n = w$, where π_A is the projection over the first component and π_B is the concatenation of the words of the second component. Since $\mathcal{A}_\mathcal{T}$ is aperiodic and deterministic, it is counter-free (Prop. 1.3.9). This means that $p \xrightarrow{s}_{\mathcal{A}_\mathcal{T}} p$, and thus $p \xrightarrow{u}_\mathcal{A} p$, hence $\mathcal{A}$ is counter-free. □

Now that we have shown the equivalence of aperiodic and $\mathsf{FO}[\leq]$-definable ω-transductions, we get the main decidability result of this chapter.

Theorem 3.3.2. *One can decide if an ω-transducer realizes an $\mathsf{FO}[\leq]$-definable function.*

Proof. This is a direct consequence of the fact that the canonical bimachine of an aperiodic function is aperiodic (Th. 3.2.37), and of the equivalence between aperiodic transducers and $\mathsf{FO}[\leq]$-transducer (Th. 3.3.1). □

3.3.2 Closure under composition

We finally show that $\mathsf{FO}[\leq]$-transductions are closed under compositions. This is obtained *via* classical $\mathsf{FO}[\leq]$-interpretations of logical structures, but we state it in our formalism.

Theorem 3.3.3. *Let $f : A^\omega \to B^\omega$ and $g : B^\omega \to C^\infty$ be $\mathsf{FO}[\leq]$-definable ω-transductions. Then $g \circ f$ is $\mathsf{FO}[\leq]$-definable.*

Proof. Let $f : A^\omega \to B^\omega$ and $g : B^\omega \to C^\infty$ be given by the $\mathsf{FO}[\leq]$-transducers $\mathcal{T}_1 := \left(K_1, \phi_{\mathrm{dom},1}, (\phi_{v,1})_{v\in K_1}\right)$ and $\mathcal{T}_2 := \left(K_2, \phi_{\mathrm{dom},2}, (\phi_{v,2})_{v\in K_2}\right)$, respectively. Let $k := \max_{v\in K_1} |v|$ be the maximum size of an output of $\mathcal{T}_1$. The main idea is to interpret formulas for words in B^ω over words of A^ω. Let ϕ be a formula over the alphabet B, we define ϕ^f to recognize the words of A^ω which are sent by f to words satisfying ϕ. If $\phi := \exists x\ \psi(x)$, then we set $\phi^k := \exists x_1, \dots, x_k\ \bigvee_{i\in\mathbf{k}} \psi(x_i)$, with each variable annotated by an integer in $\mathbf{k}$. From ϕ^k, we define ϕ'^k by substituting the atomic formulas $(x_i < x_j)$ by the boolean $(i < j)$. For $b \in B$, we also replace $b(x_i)$ by $\bigvee_{v\in K, v(i)=b} \phi_{v,1}(x_i)$. Finally we set $\phi^f := \phi_{\mathrm{dom},1} \wedge \phi'^k$. By definition the words satisfying ϕ^f are exactly the words sent by f to words satisfying ϕ. Additionally, if ϕ is an $\mathsf{FO}_\mathsf{c}[\leq]$-formula, let $v \in K_1$, and let $j \in \{1, \dots, |v|\}$ then we define $\phi^{f,v,j}$ by taking ϕ^f defined as above and substituting each $b(\mathsf{c})$ predicate by $v(j) = b$.

We define $\mathcal{T} := \left(K, \phi_{\mathrm{dom}}, (\phi_v)_{v\in K}\right)$ by:

- $K := \bigcup_{i\in\mathbf{k}} K_2^i$ where $k = \max_{v\in K^1}$
- $\phi_{\mathrm{dom}} := \phi^f_{\mathrm{dom},2}$
- $\phi_{v_1\cdots v_i \in K_2^i} := \bigvee_{v\in K_1, |v|=i} \phi^f_{v,1} \bigwedge_{l\in\mathbf{i}, j\leq|v_l|} \phi^{f,v,j}_{v_l,2}$

By construction, $\mathcal{T}$ realizes the transduction $g \circ f$, which concludes the proof. □

Part II

Specification and Synthesis of Transductions

Chapter 4

Logics for transductions with origins

"I pressed down the mental accelerator. The old lemon throbbed fiercely. I got an idea."

– P.G. Wodehouse

Transductions with origins were introduced in [Boj14], as a richer semantics for transducer models. A classical transduction (or transduction without origins) is a set of pairs of words, whereas a transduction with origins adds for each position of the output word the information of which input position "produced it". It was remarked in [Boj14] that most known models of transducers can be naturally extended with an origin semantics. This includes in particular models of finite state transducers, where the origin of an output position is naturally given by the position of the input head when the letter was produced, as well as logical transducers *à la* Courcelle. Furthermore the known transformations from one model to another preserve origins, so that inclusions of transduction classes are preserved when going to transductions with origins. Of course, transductions with origins can be considered over other structures such as infinite words, trees, graphs *etc*; however in this chapter we choose to focus on word-to-word transductions. One particular class of interest is the so-called *regular* transductions which enjoys several different characterizations. In [EH01] it was shown that MSO-transducers and deterministic two-way transducers both characterize the class of functional regular transductions. A result not presented in this manuscript is that functional regular transductions are also captured by *reversible* (*i.e.* deterministic and co-deterministic) two-way transducers [DFJL17], whereas reversible *one-way* transducers do not capture all the rational functions. A model of one-way deterministic transducers with registers (called streaming string transducers) was introduced in [AC10] where the authors showed that this model also captures regular transductions. Finally, regular expressions for transducers were shown in [AFR14] to also capture the regular functions and have since been a topic of interest (see *e.g.* [ADR15, BR18, DGK18]). All of these equivalences carry over to the origin semantics.

The main model of transducers we study is MSO logic over *origin-graphs*, *i.e.* two-sorted graphs composed of an input structure, an output structure (words here) and an *origin function* from output positions to input positions. The main problems we consider are emptiness, model-checking and synthesis problems. Let $\mathcal{C}$ be a class of transducers, the $\mathcal{C}$*-emptiness* problem asks, given a transducer $\mathcal{T}$ in $\mathcal{C}$ whether $[\![\mathcal{T}]\!] = \varnothing$ (or equivalently $[\![\mathcal{T}]\!]_o = \varnothing$). We first notice that the satisfiability (emptiness) problem for MSO over word-to-word origin graphs is (unsurprisingly) undecidable.

A transduction R_1 (with or without origins) is said to *uniformize*[1] a transduction R_2 if we have $R_1 \subseteq R_2$ and $\mathrm{dom}(R_1) = \mathrm{dom}(R_2)$. Let $\mathcal{C}_1, \mathcal{C}_2$ be classes of transducers, the *model-checking* problem of $\mathcal{C}_1$ against $\mathcal{C}_2$ asks given transducers $\mathcal{T}_1 \in \mathcal{C}_1$, $\mathcal{T}_2 \in \mathcal{C}_2$ if the transduction realized by $\mathcal{T}_1$ uniformizes the transduction realized by $\mathcal{T}_2$, called the *specification*. The model-checking of regular transductions against MSO-formulas was shown to be decidable in [BDGP17], in the origin semantics. This is not the proof used in the article, but one way to obtain this result is to observe that the origin-graphs recognized by a deterministic two-way transducer have bounded tree-width (even path-width) hence, according to Courcelle's theorem ([Cou90]), we obtain the decidability result.

The $\mathcal{C}_1, \mathcal{C}_2$*-synthesis* problem asks given a transducer $\mathcal{T}_2$ in $\mathcal{C}_2$ if one can effectively construct a transducer $\mathcal{T}_1$ in $\mathcal{C}_1$ such that the transduction realized by $\mathcal{T}_1$ uniformizes the specification given by $\mathcal{T}_2$. The Church synthesis problem, introduced in the seminal article [Chu63], considers letter-to-letter transductions over infinite words and asks whether one can synthesize a sequential letter-to-letter transducer uniformizing the input relation. It was shown in [BL69] that when the specification is a rational letter-to-letter relation over infinite words, then the problem is decidable. When removing the letter-to-letter restriction, the problem (without origins) becomes undecidable (see [CL15]) because of the asynchronicity between the input and the output. We consider the origin semantics and we show that the synthesis problem is undecidable when one wants to synthesize a regular function from an MSO-formula over origin-graphs. Our main contribution is to exhibit a fragment of MSO, called Lo, for which synthesis of a regular function is always possible, by providing a uniformization algorithm.

The logic Lo has many interesting properties. First, the satisfiability problem is decidable and the domain of an Lo-transduction is always effectively rational, meaning that one can compute an automaton recognizing it from any Lo-formula. Actually, our synthesis result is even stronger than that since we have a uniformization algorithm which takes as input an Lo-formula and outputs a regular function uniformizing it (given for instance as an MSO-transducer). Furthermore, since Lo is closed under boolean operations, one can decide equivalence of Lo-transductions *with* origins. The fragment Lo is expressive enough to capture all the regular functions. Moreover, we define two extensions of the logic for which we can extend our uniformization algorithm. The first extension, called $\exists$Lo, adds a block of existential monadic second-order quantifications in front of the formula, and just by considering a larger alphabet our synthesis result transfers to $\exists$Lo. The fragment $\exists$Lo subsumes in expressiveness the non-deterministic MSO-transducers, while the latter class is incomparable with Lo. A second extension adds new unary predicates, called *single-origin predicates*, which allow one to speak about rational properties of the subword of all output positions which originate from a single input position. This second extension subsumes the class of one-way non-deterministic transducers, which is again incomparable with Lo-transductions.

We start the chapter by introducing transductions with origins and the known models of transductions, and in particular MSO-transductions *à la* Courcelle. Then we study the frontier of decidability of fragments of MSO over origin graphs, and we compare Lo-transductions to the known classes of transductions. In the third section, we show our main result which is the regular uniformization algorithm for Lo-transductions. The fourth section deals with the two extensions of Lo mentioned above. Finally we introduce data words, *i.e.* words over an infinite alphabet, which have been a recent focus of study (see [BDM$^+$11, SZ12] for instance). We exhibit

[1]Usually, R_1 is assumed functional.

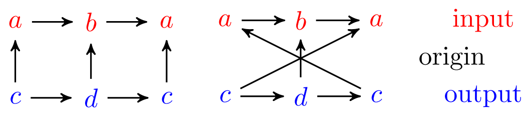


Figure 4.1: Two different origin-graphs, with the same pair of input and output words.

a very tight connection between origin-graphs and data words with linearly ordered data. As a consequence, we obtain a decidable logic for data words strictly more expressive than the one from [SZ12], which is shown to be decidable in the article.

The results in this chapter can be found in the article [DFL18].

4.1 Transductions with origin

We define origin-graphs, transductions with or without origins and known models of transductions.

4.1.1 Origin graphs

Let $\mathfrak{S}^1, \mathfrak{S}^2$ be two signatures without function symbols, called the *input* and *output signature*, respectively. Let u be a (non-empty) $\mathfrak{S}^1$-structure called the *input structure*, let v be a $\mathfrak{S}^2$-structure called the *output* structure and let $o : \mathrm{dom}(v) \to \mathrm{dom}(u)$ be a total function which we call *origin function* from v to u. We define a new signature $\mathfrak{S}^{1,2}_{\mathsf{o}}$ which contains the disjoint union of the predicate symbols of $\mathfrak{S}^1$ and $\mathfrak{S}^2$, plus a new unary function symbol o, called the *origin symbol*. We define the *origin-graph* $(u,(v,o))$ as a $\mathfrak{S}^{1,2}_{\mathsf{o}}$-structure of domain $\mathrm{dom}(u) \uplus \mathrm{dom}(v)$. Predicates symbols are interpreted naturally in the structure $(u,(v,o))$ by either restricting to u or v. The origin symbol o is interpreted as o over $\mathrm{dom}(v)$ and as the identity function[2] over $\mathrm{dom}(u)$. An origin-graph is called *non-erasing* if any input position is the origin of at least one output position. Given alphabets A, B an origin-graph over $\mathfrak{S}^A, \mathfrak{S}^B$ is called a *word-to-word origin-graph over* A, B if its input and output structures are words. In the following, the origin-graphs we consider will mostly be word-to-word. Note that we exclude the empty word as a possible input word but this does not weaken the models we consider, up to adding a special symbol at the beginning of words, for instance.

4.1.2 Transductions

Let $\mathfrak{S}^1, \mathfrak{S}^2$ be two signatures, an origin-free transduction (or just transduction) over $\mathfrak{S}^1, \mathfrak{S}^2$ is a set of pairs (u, v) such that u is a (non-empty) $\mathfrak{S}^1$-structure and v is a $\mathfrak{S}^2$-structure. The domain of a transduction R is the set $\mathrm{dom}(R) := \{u \mid \ (u,v) \in R\}$. A *transduction with origin* (or *origin-transduction*) over $\mathfrak{S}^1, \mathfrak{S}^2$ is a set τ of origin-graphs $(u,(v,o))$ such that u is a $\mathfrak{S}^1$-structure, v is a $\mathfrak{S}^2$-structure and $o : \mathrm{dom}(v) \to \mathrm{dom}(u)$ is the origin function. Given an origin-transduction τ, we define its *origin-free projection* $\mathfrak{p}(\tau) := \{(u,v) \mid \ \exists (u,(v,o)) \in \tau\}$. The *domain* of an origin-transduction τ is the set $\mathrm{dom}(\tau) := \mathrm{dom}(\mathfrak{p}(\tau))$. An origin-transduction τ is called *functional* if for any structure $u \in \mathrm{dom}(\tau)$ there exists a unique pair (v, o) such that $(u,(v,o)) \in \tau$. The composition of origin-transductions τ_1 over $\mathfrak{S}^1, \mathfrak{S}^2$ and τ_2 over $\mathfrak{S}^2, \mathfrak{S}^3$ is defined by $\tau_2 \circ \tau_1 := \{(u_1,(v_2, o_1 \circ o_2)) \mid \ (u_i,(v_i,o_i)) \in \tau_i, \ i \in \mathbf{2}, \ v_1 = u_2\}$. Note that we have

[2]The extension of the origin function to $\mathrm{dom}(u)$ is just a technicality so that o can be interpreted as a total function.

$\mathfrak{p}(\tau_2 \circ \tau_1) = \mathfrak{p}(\tau_2) \circ \mathfrak{p}(\tau_1)$, which justifies the notation. An origin-transduction is called *non-erasing* if all its origin-graphs are non-erasing. Given alphabets A, B a transduction over $\mathfrak{S}^A, \mathfrak{S}^B$ is called a *word-to-word transduction over A, B* if all its origin-graphs are word-to-word. We tend to write origin-transductions with Greek letters and origin-free transductions with capital Latin letters.

4.1.3 MSO-transducers

Courcelle MSO-transducers (see [Cou94]), generalize the order-preserving definition we gave in Ch. 1. In general they define transductions from relational structures over one signature to structures over a second signature, but we will mostly focus on word-to-word transductions. Let $\mathfrak{S}^1, \mathfrak{S}^2$ be two signatures without function symbols. We start by defining non-deterministic MSO-transducers and then we define the deterministic version, MSO-transducers, as a particular restriction. An NMSO-transducer over signatures $\mathfrak{S}^1$ and $\mathfrak{S}^2$ is a tuple

$$\mathcal{T} = \left(k, l, \phi_{\text{dom}}, \left(\phi^i_{\text{pos}}(x, X_1, \dots, X_l)\right)_{i \in \mathbf{k}}, \left(\phi^{i_1,\dots,i_m}_R(x_1, \dots, x_m, X_1, \dots, X_l)\right)_{\substack{i_1,\dots,i_m \in \mathbf{k} \\ \mathsf{ar}^2(R)=m}} \right)$$

where k is a positive integer (called the number of copies), l is an integer (called the number of parameters), ϕ_{dom} is an $\mathsf{MSO}[\mathfrak{S}^1]$-sentence, for $i \in \mathbf{k}$, ϕ^i_{pos} is an $\mathsf{MSO}[\mathfrak{S}^1]$-formula, for R a relation symbol of $\mathfrak{S}^2$ of arity m and for $i_1, \dots, i_m \in \mathbf{k}$, $\phi^{i_1,\dots,i_m}_R$ is an $\mathsf{MSO}[\mathfrak{S}^1]$-formula. From u a $\mathfrak{S}^1$-structure, and for $P_1, \dots, P_l \subseteq \text{dom}(u)$ we define a $\mathfrak{S}^2$ structure v by:

- $\text{dom}(v) := \left\{(i, j) \mid \ i \in \text{dom}(u), \ j \in \mathbf{k}, \ u \models \phi^j_{\text{pos}}(i, P_1, \dots, P_l)\right\}$
- $R^v := \left\{((i_1, j_1), \dots, (i_m, j_m)) \in \text{dom}(v)^m \mid \ u \models \phi^{j_1,\dots,j_m}_R(i_1, \dots, i_m, P_1, \dots, P_l)\right\}$
  where R is a $\mathfrak{S}^2$ relation symbol of arity m

We now define naturally an origin function $o : \text{dom}(v) \to \text{dom}(u)$ by $o(i, j) = i$. We say that the origin-graph $(u, (v, o))$ is *realized* by $\mathcal{T}$. We define two semantics for transductions: the origin-transduction *realized* by $\mathcal{T}$ is the set $[\![\mathcal{T}]\!]_o$ of origin-graphs realized by it, while the origin-free transduction *realized* by $\mathcal{T}$ is defined by $[\![\mathcal{T}]\!] := \mathfrak{p}([\![\mathcal{T}]\!]_o)$. An MSO-transduction is a particular case of NMSO-transduction with no parameters, *i.e.* $l = 0$. In that case, we will simply omit l in the tuple. Note that MSO-transductions (with or without origins) are functional by definition. A *regular word-to-word transduction* over alphabets A, B is an NMSO-transduction over signatures $\mathfrak{S}_A$, $\mathfrak{S}_B$ with the semantic restriction to word-to-word origin graphs (note that this restriction can be enforced in $\mathsf{MSO}[\leq]$). A word-to-word MSO-transduction is called a *regular function.*

Example 4.1.1. We give an example of an NMSO-transducer over words with alphabets A, A. We define an NMSO-transducer which copies some subword of the input twice. We define a transducer with one parameter variable X which will denote the positions of the copied subword. Let $\mathcal{T}_{\text{d}} := \left(2, 1, \top, \left(\phi^i_{\text{pos}}(x, X)\right)_{i \in \mathbf{2}}, (a(x))_{a \in A}, \left(\phi^{i,j}_{\leq}(x, y, X)\right)_{i,j \in \mathbf{2}}\right)$ with $\forall i \in \mathbf{2}$, $\phi^i_{\text{pos}}(x, X) := x \in X$, $\phi^{i,i}_{\leq}(x, y, X) := x \leq y$, $\phi^{1,2}_{\leq}(x, y, X) := \top$ and $\phi^{2,1}_{\leq}(x, y, X) := \bot$. The transduction we obtain is $[\![\mathcal{T}_{\text{d}}]\!] = \{(u, vv) \mid \ v \text{ subword of } u\}$.

Similarly, we define a NMSO-transducer which does the same but only when the subword has even length. Let $\text{even}(X)$ be formula saying that X contains an even number of positions, then $\mathcal{T}_{\text{d,e}} := \left(2, 1, \text{even}(X), \left(\phi^i_{\text{pos}}(x, X)\right)_{i \in \mathbf{2}}, (a(x))_{a \in A}, \left(\phi^{i,j}_{\leq}(x, y, X)\right)_{i,j \in \mathbf{2}}\right)$. Here we have $[\![\mathcal{T}_{\text{d,e}}]\!] = \{(u, vv) \mid \ v \text{ subword of } u, \ |v| = 0 \mod 2\}$.

The equivalence between MSO-transducers and deterministic two-way transducers is due to [EH01]. The equivalence between SSTs and MSO-transducers is due to [AC10]. As it was remarked in [Boj14], the equivalences carry over to origin semantics.

Theorem 4.1.2. *The following models characterize the regular word-to-word functions (with or without origins):*

- *deterministic two-way transducers*
- MSO*-transducers*
- *streaming string transducers*

It was also shown in [AD11] that one of the above equivalences generalizes to the non-deterministic case.

Theorem 4.1.3. *The following models characterize the regular word-to-word relations (with or without origins):*

- NMSO*-transducers*
- *non-deterministic streaming string transducers*

However, non-deterministic two-way (and even one-way) transducers are incomparable in expressiveness with NMSO-transducers.

4.2 Logics with origins

A logic over origin-graphs defines a class of transductions. We consider several fragments of MSO over word-to-word origin graphs, and study their respective expressiveness and the decidability of the problems mentioned above: emptiness, model-checking and synthesis of regular functions.

4.2.1 MSO over word-to-word origin graphs

We denote by $\mathsf{MSO}_o = \mathsf{MSO}[A, B, \leq_{\mathsf{out}}, \leq_{\mathsf{in}}, \mathsf{o}]$ the logic over word-to-word origin-graphs with $\leq_{\mathsf{in}}, \leq_{\mathsf{out}}$ the input and output linear orders and A, B the input and output alphabets, respectively and o the origin function. Note that, unless specified, the alphabets are implicitly known and we write $\mathsf{MSO}[\leq_{\mathsf{out}}, \leq_{\mathsf{in}}, \mathsf{o}]$, instead. Given an MSO_o-formula ϕ, we define by $[\![\phi]\!]_o$ the set of origin-graphs satisfying ϕ and by $[\![\phi]\!] := \mathfrak{p}([\![\phi]\!]_o)$ the origin-free semantics of ϕ.

Example 4.2.1. We define several macros for MSO_o, which will make notations easier. We define $\mathsf{in}(x) := x \leq_{\mathsf{in}} x$ and $\mathsf{out}(x) := x \leq_{\mathsf{out}} x$, the input and output predicates, respectively. We naturally define the predicates $=, <_{\mathsf{in}}, <_{\mathsf{out}}$. We also define quantifiers restricted to either input or output positions: let $\alpha \in \{\mathsf{in}, \mathsf{out}\}$ then $\exists^\alpha x\ \phi := \exists x\ \alpha(x) \wedge \phi$, $\forall^\alpha x\ \phi := \forall x\ \alpha(x) \rightarrow \phi$. Note that $\forall^\alpha x\ \phi = \neg\exists^\alpha x\ \neg\phi$.

We define several formulas with (hopefully) transparent names:

- $\phi_{\text{inj}} := \forall^{\mathsf{out}} x, y\ \mathsf{o}(x) = \mathsf{o}(y) \rightarrow x = y$, states that o is injective
- $\phi_{\text{surj}} := \forall^{\mathsf{in}} x\ \exists^{\mathsf{out}} y\ \mathsf{o}(y) = x$, states that o is surjective
- $\phi_{\text{lab}} := \forall^{\mathsf{out}} x\ \bigwedge_{a \in A} a(\mathsf{o}(x)) \rightarrow a(x)$, states that labels are preserved
- $\phi_{\text{shuffle}} := \phi_{\text{inj}} \wedge \phi_{\text{surj}} \wedge \phi_{\text{lab}}$

- $\phi_{\text{id}} := \phi_{\text{shuffle}} \wedge \forall^{\mathsf{out}} x, y\ \mathsf{o}(x) \leq_{\mathsf{in}} \mathsf{o}(y) \to x \leq_{\mathsf{out}} y$
- $\phi_{\text{mirror}} := \phi_{\text{shuffle}} \wedge \forall^{\mathsf{out}} x, y\ \mathsf{o}(x) \leq_{\mathsf{in}} \mathsf{o}(y) \to x \geq_{\mathsf{out}} y$

The formula ϕ_{shuffle} recognizes origin graphs where the output word is a permutation of the input word. ϕ_{id} recognizes the identity transduction while ϕ_{mirror} recognizes the mirror transduction, which reverses the linear order of words.

4.2.2 Model-checking

We state the following theorem from [BDGP17], which solves the model-checking problem in a quite general context. The theorem would actually hold for any device recognizing origin-graphs of bounded tree-width, thanks to the Courcelle Theorem [Cou90]. One should keep in mind however that model-checking *with* origins is a more restricted form of model-checking, where the specification has to say how the output is produced. The statement talks about NSSTs, a non-deterministic variant of the SSTs of [AC10], which was shown to be expressively equivalent to MSO-transducers in [AD11].

Theorem 4.2.2. *The model-checking of NSSTs against* MSO_o*-formulas is decidable, in the origin semantics.*

4.2.3 Satisfiability, validity, equivalence

We define several problems for logics in general. The problem which will interest us the most is the satisfiability problem since a decidable satisfiability problem for a fragment closed under boolean operations entails the decidability of the other two. Let F be a logical fragment of MSO over some signature and let $\mathcal{M}$ be a set of structures. The *satisfiability* (emptiness) problem asks, given as input an F-formula ϕ, whether there exists a structure $M \in \mathcal{M}$ such that $M \models \phi$. A logic with decidable satisfiability problem is sometimes called *decidable*. The *validity* (universality) problem asks, given as input an F-formula ϕ, whether for any structure $M \in \mathcal{M}$ it holds that $M \models \phi$. The *equivalence* problem asks, given as input two F-formulas ϕ_1, ϕ_2, whether for any structure $M \in \mathcal{M}$ it holds that $M \models \phi_1 \Leftrightarrow M \models \phi_2$.

4.2.4 Undecidable fragments

We exhibit some fragments of MSO_o for which satisfiability of formulas is undecidable. Since the results rely on the comparison between origin-graphs and data words we give the proofs in Sec. 4.5 In the following theorem we consider the logic $\mathsf{FO}^2[\leq_{\mathsf{out}}, S_{\mathsf{out}}, \leq_{\mathsf{in}}, \mathsf{o}]$ of first-order formulas with only two variables, where S_{out} denotes the successor predicate over the output word. The proof is inspired by the undecidability result of [BDM+11] over data words.

Theorem 4.2.3. *The satisfiability problem for* $\mathsf{FO}^2[\leq_{\mathsf{out}}, S_{\mathsf{out}}, \leq_{\mathsf{in}}, \mathsf{o}]$ *is undecidable.*

We now consider the logic $\mathsf{FO}^2[S_{\mathsf{out}}, S_{\mathsf{in}}, \mathsf{o}]$ with $S_{\mathsf{in}}, S_{\mathsf{out}}$ denoting the successor predicates over the input and output, respectively. It is well known that the successor predicate of a linear order is not definable in FO^2. The proof uses the result of [MSZ13] over data words.

Theorem 4.2.4. *The satisfiability problem for* $\mathsf{FO}^2[S_{\mathsf{out}}, S_{\mathsf{in}}, \mathsf{o}]$ *is undecidable.*

4.2.5 A new fragment

As we have just seen, even very weak fragments of MSO_o still have an undecidable satisfiability problem. Our objective is to find a fragment of MSO_o which is weak enough to be decidable, but still as expressive as possible. Our initial objective was to find a fragment expressive enough to capture *at least* the regular functions.

We define $\mathsf{L}o$, a fragment of MSO_o which does capture regular functions (and much more). We show in Sec. 4.4 that the satisfiability problem *is* decidable for $\mathsf{L}o$, and we even show a stronger result: an algorithm that takes an $\mathsf{L}o$-formulas as an input and outputs a regular function uniformizing it. Intuitively, $\mathsf{L}o$ is an extension of $\mathsf{FO}^2[A, B, \leq_{\mathsf{in}}, \leq_{\mathsf{out}}, \mathsf{o}]$, with additional binary predicates. These predicates are arbitrary predicates definable in $\mathsf{MSO}[A, \leq_{\mathsf{in}}]$, *i.e.* regular predicates which can only talk about the input.

Definition 4.2.5. We define over alphabets A, B the logic $\mathsf{L}o := \mathsf{FO}^2[B, \leq_{\mathsf{out}}, \mathsf{o}, \mathsf{MSO}[A, \leq_{\mathsf{in}}]]$

In order to make formulas more readable we write $\mathsf{MSO}[A, \leq_{\mathsf{in}}]$ predicates, called *input predicates*, in between chevrons $\langle, \rangle$. Note that from now on, the input predicates are viewed as atomic formulas. Furthermore, since we only have two first-order variables, we can assume without loss of generality that the input predicates have at most two free first-order variables.

Example 4.2.6. All the transductions given in Ex. 4.2.1 are actually definable in $\mathsf{L}o$. For instance we define equivalently $\phi_{\text{inj}} := \forall^{\mathsf{out}} x, y \ \langle z =_{\mathsf{in}} t\rangle(\mathsf{o}(x), \mathsf{o}(y)) \rightarrow x = y$. To simplify notations, we will often substitute the terms for the variables in the predicates $\langle\phi\rangle$. For instance we write $\phi_{\text{inj}} := \forall^{\mathsf{out}} x, y \ \langle \mathsf{o}(x) =_{\mathsf{in}} \mathsf{o}(y)\rangle \rightarrow x = y$. Let us give the other previous examples in this formalism.

- $\phi_{\text{surj}} := \forall^{\mathsf{in}} x \ \exists^{\mathsf{out}} y \ \langle \mathsf{o}(y) =_{\mathsf{in}} x\rangle$, states that o is surjective
- $\phi_{\text{lab}} := \forall^{\mathsf{out}} x \ \bigwedge_{a \in A} \langle a(\mathsf{o}(x))\rangle \rightarrow a(x)$, states that labels are preserved
- $\phi_{\text{shuffle}} := \phi_{\text{inj}} \wedge \phi_{\text{surj}} \wedge \phi_{\text{lab}}$
- $\phi_{\text{id}} := \phi_{\text{shuffle}} \wedge \forall^{\mathsf{out}} x, y \ \langle \mathsf{o}(x) \leq_{\mathsf{in}} \mathsf{o}(y)\rangle \rightarrow x \leq_{\mathsf{out}} y$
- $\phi_{\text{mirror}} := \phi_{\text{shuffle}} \wedge \forall^{\mathsf{out}} x, y \ \langle \mathsf{o}(x) \leq_{\mathsf{in}} \mathsf{o}(y)\rangle \rightarrow x \geq_{\mathsf{out}} y$

Let us now give another example which actually makes use of a third variable. Note that the use of the third variable is made inside an MSO predicate and so is allowed in $\mathsf{L}o$: $\exists^{\mathsf{out}} x, y \ a(x) \wedge a(y) \wedge \langle \mathsf{o}(x) <_{\mathsf{in}} \mathsf{o}(y)\rangle \wedge \langle \forall^{\mathsf{in}} z \ z \leq_{\mathsf{in}} \mathsf{o}(x) \vee \mathsf{o}(y) \leq_{\mathsf{in}} z\rangle$. This formula recognizes origin-graphs such that there are two output positions labeled by a whose respective origins are adjacent.

4.2.6 Expressing regular transductions

Regular functions We show how to express regular functions in $\mathsf{L}o$.

Theorem 4.2.7. *Any regular function can be expressed in* $\mathsf{L}o$.

Proof. We consider $\mathcal{T} := \left(k, \phi_{\text{dom}}, \left(\phi^i_{\text{pos}}(x)\right)_{i \in \mathbf{k}}, \left(\phi^i_a(x)\right)_{i \in \mathbf{k}, a \in B}, \left(\phi^{i,j}_{\leq}(x, y)\right)_{i,j \in \mathbf{k}}\right)$ a word-to-word MSO-transducer over A, B. Given an $\mathsf{MSO}[A, \leq]$-formula ϕ of $\mathcal{T}$, we will abuse notations and write $\langle\phi\rangle$ to denote the predicate where $\leq_{\mathsf{in}}$ and $\exists^{\mathsf{in}}$ have been substituted for $\leq$ and $\exists$, respectively.

Let us start by defining formulas which talk about the output produced by some input position x. Let $P \subseteq \mathbf{k}$. We define the formula saying which copies of x are used.

$$\phi_P(x) := \bigwedge_{i \in P} \phi^i_{\mathrm{pos}}(x) \bigwedge_{i \notin P} \neg\phi^i_{\mathrm{pos}}(x)$$

Let $i_1, \dots, i_l$ be a non-repeating sequence of integers in $\mathbf{k}$, we define a formula specifying the order of the copies of x.

$$\phi_{i_1,\dots,i_l}(x) := \phi_{\{i_1,\dots,i_l\}}(x) \bigwedge_{1 \le m < n \le l} \phi^{i_m,i_n}_{\le}(x,x)$$

Let $v \in B^l$, we define a formula saying that x produces v.

$$\phi_{i_1,\dots,i_l,v}(x) := \phi_{i_1,\dots,i_l}(x) \bigwedge_{j \in \mathbf{l}} \phi^{i_j}_{v(j)}(x)$$

Now we define inductively formulas to state that an output position has at least i positions to its left with the same origin for $i \ge 0$.

$$D_0(x) := \mathsf{out}(x)$$

$$D_{i+1}(x) := \exists^{\mathsf{out}} y\ y \le_{\mathsf{out}} x \wedge \langle \mathsf{o}(y) =_{\mathsf{in}} \mathsf{o}(x) \rangle \wedge D_i(y)$$

From this we define $C_{i+1}(x) := D_i(x) \wedge \neg D_{i+1}(x)$ which states that x is exactly the $i+1$st position with origin x.

We are now able to define the final formula defining the same origin-transduction as $\mathcal{T}$.

$$\begin{array}{ll}
\langle \phi_{\mathrm{dom}} \rangle \wedge & \big(\forall^{\mathsf{in}} x\ \langle \phi_{\varnothing}(x) \rangle \leftrightarrow \forall^{\mathsf{out}} y\ \langle \mathsf{o}(y) \neq_{\mathsf{in}} x \rangle\big) \\
\wedge \forall^{\mathsf{out}} x & \bigvee_{l,i_1,\dots,i_l,\in \mathbf{k}, a \in B, p \in \mathbf{l}} \\
 & \ C_p(x) \wedge \langle \phi_{i_1,\dots,i_l} \rangle(\mathsf{o}(x)) \wedge \langle \phi^{i_p}_a \rangle(\mathsf{o}(x)) \wedge a(x) \\
\wedge \forall^{\mathsf{out}} x, y & \bigwedge_{l,m,i_1,\dots,i_l,j_1,\dots,j_m \in \mathbf{k}, a,b \in B, p \in \mathbf{l}, q \in \mathbf{m}} \\
 & \ C_p(x) \wedge C_q(y) \wedge \langle \phi_{i_1,\dots,i_l} \rangle(\mathsf{o}(x)) \wedge \langle \phi_{j_1,\dots,j_m} \rangle(\mathsf{o}(y)) \\
 & \ \wedge \langle \phi^{i_p}_a \rangle(\mathsf{o}(x)) \wedge \langle \phi^{j_q}_b \rangle(\mathsf{o}(y)) \wedge \langle \phi^{i_p,j_q}_{\le}(\mathsf{o}(x), \mathsf{o}(y)) \rangle \\
 & \to a(x) \wedge b(y) \wedge (x \le_{\mathsf{out}} y)
\end{array}$$

The formula can seem scary but it can be decomposed into three main parts: in the first line we state that the input satisfies the domain formula and that an input position which does not produce anything must satisfy $\phi_{\varnothing}$. The big disjunction states that any output position must correspond to some copy of its origin. The big conjunction makes sure that the output positions do satisfy the properties enforced by $\mathcal{T}$. □

Existential logic We define a new logic over origin-graphs $\exists\mathsf{L}o$, which consists in formulas of the form $\exists X_1 \dots \exists X_n\ \phi$ where ϕ is an $\mathsf{L}o$-formula with additional unary predicates $X_1, \dots, X_n$ available both for input and output positions. We show that this extension captures NMSO-transductions.

Theorem 4.2.8. *Any regular transduction can be expressed in $\exists\mathsf{L}o$.*

Proof. The proof is very similar to the one for regular origin-functions. Indeed, an NMSO-transducer can be thought of as an MSO-transducer but over a larger alphabet. If $\mathcal{T}$ is an NMSO-transducer with parameters $X_1, \dots, X_n$, then we define an $\exists\mathsf{L}o$ formula $\exists X_1, \dots, X_n\ \phi$ where ϕ is an $\mathsf{L}o$-formula over a signature extended with new unary predicates. The $\mathsf{L}o$ formula ϕ is defined exactly as in the proof of Th. 4.2.7. □

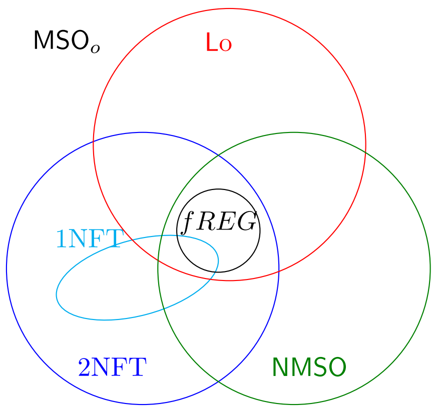


Figure 4.2: Classes of transductions.

Example 4.2.9. We give examples of transductions which belong to some classes of Fig. 4.2 but not others. A transduction which is not a function but in all other classes could be $\{(a,a),(a,b)\}$. A typical example of a transduction which cannot be done by a one-way transducer (1NFT) is the mirror function given by ϕ_{mirror} from Ex. 4.2.6. An example which is definable by an NMSO-transduction or in $\mathsf{L}o$ but cannot be done by a two-way transducer (2NFT) is given by $\mathcal{T}_{\mathrm{d}}$ from Ex. 4.1.1 which intuitively needs to make twice the same non-deterministic choice on each position. An origin-transduction which can be done by a 1NFT, or an NMSO-transducer but not in $\mathsf{L}o$ is keeping an even number of positions and erasing the others. The intuition behind this is that parity cannot be expressed in $\mathsf{FO}^2[\leq]$. Finally an example which cannot be realized by an NMSO-transduction is $\{a\} \times A^*$ since NMSO-transductions can only have a *linear size increase*, meaning that for any regular transduction there is a uniform constant k such that the output is at most k times the size of the input.

Combining these we can obtain a transduction in any part of the diagram of Fig. 4.2. Let us give an example of MSO_o-transduction that does not belong to the other classes. Let $\phi_{\mathrm{cfl}} := \phi_{\mathrm{shuffle}} \wedge \phi^{\mathsf{in}}_{a^*b^*} \wedge \phi^{\mathsf{out}}_{(ab)^*}$, where ϕ_{shuffle} is the formula from Ex. 4.2.1 which says that the output is a permutation of the input. The formula $\phi^{\mathsf{in}}_{a^*b^*}$ states that the input belongs to a^*b^* which can be expressed in first-order logic with two variables with either S_{in} or $\leq_{\mathsf{in}}$. On the other hand $\phi^{\mathsf{out}}_{(ab)^*}$ states that the output belongs to $(ab)^*$, which is famously not expressible in in first-order logic with two variables and $\leq_{\mathsf{out}}$ but is of course expressible with S_{out}. This means that the domain of the transduction defined by ϕ_{cfl} is the language $\{a^nb^n \mid n \geq 1\}$ which is famously not rational. However as we will see in the next sections, $\mathsf{L}o$-transductions always have a rational domain.

4.3 Reduction of the regular synthesis problem

The $\mathsf{L}o$ *regular synthesis problem* asks, for an origin transduction given as an $\mathsf{L}o$-formula if one can obtain a regular function (given as an MSO-transducer for instance) uniformizing it. Before tackling the regular synthesis problem for $\mathsf{L}o$-formulas, we start by reducing the problem several times to simpler problems. First we show how to restrict the problem to non-erasing transductions. This can easily be done by adding a copy of the input at the beginning of the output word. Second, we put formulas in Scott Normal Form (SNF), which is a classical method when dealing with first-order logic with two variables, allowing one to reduce the quantification rank of a formula to two (see [GO99]). Third, we show that over non-erasing origin-graphs, we

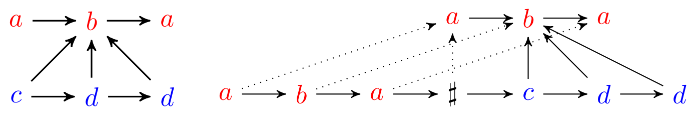


Figure 4.3: Transforming an origin-graph into a non-erasing origin-graph.

can restrict quantifications to output positions since any input position is the origin of some output position. Finally we replace these simple low quantifier rank formulas by sets of simple constraints called *MSO constraints*. From these reductions we can describe the uniformization algorithm. First we start by transforming the input formula following the previous steps. Then we describe a uniformization algorithm for MSO constraints. Finally we work backwards to retrieve a uniformization of the initial transduction.

4.3.1 Non-erasing transductions

Let $g = (u, (v, o))$ be an origin-graph over alphabets A, B, we define a non-erasing origin-graph over $A, B \uplus \{\sharp\} \uplus A$ by $g^{\text{ne}} := (u, (u\sharp v, o'))$ such that for $i \in \{1, \dots, |u|\}$, $o'(i) := i$, $o'(|u|+1) := 1$ and for $i > |u|+1$, $o'(i) := o(i - |u| - 1)$. Intuitively, we add on the left of v a copy of the input and separate the two by a $\sharp$ symbol. This is illustrated in Fig. 4.3. We now show that we only have to find a uniformization algorithm for non-erasing formulas.

Theorem 4.3.1. *Let ϕ be an* Lo*-formula. We can define a non-erasing* Lo*-formula ϕ^{ne} such that for any origin-graph g, $g \models \phi \Leftrightarrow g^{\text{ne}} \models \phi^{\text{ne}}$.*

Proof. Let ϕ be an Lo-formula over alphabets A, B. The main idea of the proof is to restrict quantifications to either positions before or after a special symbol $\sharp$. Then, we say that the output restricted to positions before the $\sharp$ satisfies the identity formula and the output restricted to positions after the $\sharp$ satisfies ϕ.

We start by defining $\psi_\sharp := \exists^{\mathsf{out}} x\ \sharp(x) \wedge \forall^{\mathsf{out}} x, y\ (\sharp(x) \wedge \sharp(y)) \rightarrow x = y$, stating that there exists exactly one output position labeled by $\sharp$. Then, we can define the predicates $<_\sharp(x) := \exists^{\mathsf{out}} y\ x <_{\mathsf{out}} y \wedge \sharp(y)$ and $>_\sharp(x) := \exists^{\mathsf{out}} y\ y <_{\mathsf{out}} x \wedge \sharp(y)$. We also define $\psi_{A\sharp B} := \psi_\sharp \wedge \forall^{\mathsf{out}} x <_\sharp(x) \leftrightarrow (\bigvee_{a \in A} a(x))$ to ensure that the alphabets are respected. Given an Lo-formula ψ, we define inductively $\psi^{<\sharp}$, by restricting the output quantifications to positions before the $\sharp$. If $\psi = \exists^{\mathsf{out}} x\ \theta(x)$ then we define $\psi^{<\sharp} := \exists^{\mathsf{out}} x <_\sharp(x) \wedge \theta^{<\sharp}(x)$. Predicates and boolean connectives are not affected by this. Similarly we define $\psi^{>\sharp}$. We can define the final formula:

$$\phi^{\text{ne}} := \psi_{A\sharp B} \wedge \phi_{\text{id}}^{<\sharp} \wedge \phi^{>\sharp}$$

where ϕ_{id} is the formula defining the identity origin-transduction given in Ex. 4.2.1.

□

A consequence of the previous result is that a uniformization algorithm for non-erasing Lo-formulas yields an algorithm for general Lo-formulas.

Lemma 4.3.2. *Let phi be an* Lo *formula, and let $\mathcal{T}$ be an* MSO*-transducer uniformizing $[\![\phi^{\text{ne}}]\!]_o$. Then we can obtain an* MSO*-transducer $\mathcal{T}'$ uniformizing $[\![\phi]\!]_o$*

Proof. Let *phi* be an Lo formula over alphabets A, B, and let $\mathcal{T}$ be an MSO-transducer uniformizing $[\![\phi^{\text{ne}}]\!]_o$. Let $\mathcal{T}_{\text{erase}}$ be an MSO-transducer defined over alphabets $B \uplus \{\sharp\} \uplus A$, B which simply erases all positions not labeled in B. According to Th. 4.3.1, we have that $[\![\mathcal{T}_{\text{erase}}]\!]_o \circ [\![\phi^{\text{ne}}]\!]_o = [\![\phi]\!]_o$.

Since $[\![\phi^{\mathrm{ne}}]\!]_o$ and $[\![\phi]\!]_o$ this means that if $\mathcal{T}$ uniformizes $[\![\phi^{\mathrm{ne}}]\!]_o$, then $[\![\mathcal{T}_{\mathrm{erase}}]\!]_o \circ [\![\mathcal{T}]\!]_o$ uniformizes $[\![\phi]\!]_o$. Furthermore, since MSO-transducers are closed under composition (see [Cou94]), we can define an MSO-transducer $\mathcal{T}'$ realizing $[\![\mathcal{T}_{\mathrm{erase}}]\!]_o \circ [\![\mathcal{T}]\!]_o$, which concludes the proof. □

4.3.2 Scott Normal Form

We now describe the classical method (see [GO99]) used for two-variable logic, the Scott Normal Form (SNF). The main idea is to reduce the quantification rank by augmenting the signature with new unary predicates. The number of additional unary predicates is linear in the size of the original formula. Let ϕ be an L*o*-formula, and let $\psi(x)$ be a subformula of ϕ with one quantifier. Let ϕ' be the formula ϕ where the predicate $P(x)$ has been substituted for every occurrence of $\psi(x)$. Then the formula $\phi' \wedge \forall x\ \psi(x) \leftrightarrow P(x)$ has the same models as ϕ, up to projecting away the predicate P. Hence we obtain a new formula with a shallower syntactic tree, and by repeating the process we obtain a formula of quantifier rank two.

Theorem 4.3.3. *Let ϕ be an L*o*-formula over alphabets A, B. One can obtain an L*o* formula ψ over alphabets $A, B \times B'$ such that:*

- *B' is finite*
- *Up to projection $\pi : B \times B' \to B$ of the output alphabet, ϕ and ψ have the same models*
- *ψ is of the form $\forall x, y\ \chi(x, y) \wedge \bigwedge_{i \in \mathbf{n}} \forall x\ \exists y\ \beta_i(x, y)$, where $\forall i \in \mathbf{n}\ \chi, \beta_i$ are quantifier-free.*

Proof. First, we assume without loss of generality that negations only occur at the level of atomic formulas. Let k denote the number of quantifiers in ϕ. We construct ψ iteratively, by defining for $i \leq k$ formulas θ_i and ψ_i such that ϕ is equivalent to $\theta_i \wedge \psi_i$ (up to projecting over B), and ψ_i has i less quantifiers than ϕ. We initialize by $\theta_0 := \top$ and $\psi_0 := \phi$. Let $i > 0$ and consider $\xi_i(x)$ a subformula of ψ_{i-1} with one quantifier. Then $\xi_i(x)$ is either of the form $\exists y\ \rho(x, y)$ or $\forall y\ \rho(x, y)$, where $\rho(x, y)$ is a quantifier-free formula. Either way let $\theta_i := \theta_{i-1} \wedge \forall x\ P_i(x) \leftrightarrow \xi_i(x)$, where P_i is a fresh unary predicate. The formula ψ_i is obtained by substituting $P_i(x)$ for $\xi_i(x)$ in ψ_{i-1}.

Since ϕ has k quantifiers, we know that ψ_k is quantifier-free. Furthermore, θ has two types of conjuncts, $\forall\forall$ or $\forall\exists$. All $\forall\forall$ conjuncts can be regrouped in one formula χ, and we obtain a formula in the advertised form.

Let us now argue that ϕ and ψ are equivalent, up to projection over B. Let $i \leq k$, we show by induction on i that $\theta_i \wedge \psi_i$ is equivalent to ϕ. Let $i < k$ and let us assume that the property holds for i. Let g_i be a model for $\theta_i \wedge \psi_i$, we define g_{i+1} by adding the predicate P_{i+1} interpreted by $\{p \in \mathrm{dom}(g_i) \mid\ g_i, p \models \xi_{i+1}(x)\}$. Of course $g_{i+1} \models \theta_{i+1}$ since P_{i+1} is interpreted exactly as the positions in which $\xi_{i+1}(x)$ holds. Similarly, since the positions satisfying $\xi_{i+1}(x)$ are exactly the ones satisfying $P_{i+1}(x)$ we have that $g_{i+1} \models \psi_{i+1}$.

Conversely, let $g_{i+1} \models \theta_{i+1} \wedge \psi_{i+1}$. We define g_i by projecting away the predicate P_{i+1}. We have *a fortiori* that $g_i \models \theta_i$. Furthermore, all positions of g_{i+1} satisfying $P_{i+1}(x)$ are exactly the positions satisfying $\xi_{i+1}(x)$. However the positions of g_i which satisfy $\xi_{i+1}(x)$ are exactly the same as for g_{i+1}, which means that $g_i \models \psi_i$, and concludes the proof. □

Remark 4.3.4. Note that the number of new predicates is linear in the size of the formula. However, the number of new *letters* is exponential. Indeed the new predicates are not exclusive, so we obtain 2^k new letters.

Given a uniformization algorithm for L*o*-formulas in SNF, we are thus able to retrieve an algorithm for general L*o*-formulas by projecting the output labels over the original alphabet.

Lemma 4.3.5. *Let ϕ be an* L*o-formula, let ψ be the obtained formula in SNF and let $\mathcal{T}$ be an* MSO*-transducer uniformizing $[\![\psi]\!]_o$. One can define an* MSO*-transducer $\mathcal{T}'$ uniformizing $[\![\phi]\!]_o$.*

Proof. Let ϕ be an L*o*-formula, let ψ be the obtained formula in SNF and let $\mathcal{T}$ be an MSO-transducer uniformizing $[\![\psi]\!]_o^{/} =$. According to Th. 4.3.3, ϕ and ψ have the same models up to projection $\pi : B \times B' \to B$. Hence we define $\mathcal{T}_\pi$ a word-to-word MSO-transducer replacing every label of the form $(b, b') \in B \times B'$ by b. Thus we have that $[\![T_\pi]\!]_o \circ [\![\psi]\!]_o = [\![\phi]\!]_o$. Since ϕ and ψ have the same domain, then we have that $[\![T_\pi]\!]_o \circ [\![\mathcal{T}]\!]_o$ uniformizes $[\![\phi]\!]_o$. Again since MSO-transducers are closed under composition ([Cou94]), we can define $\mathcal{T}'$ realizing $[\![T_\pi]\!]_o \circ [\![\mathcal{T}]\!]_o$. □

4.3.3 Output formulas

An output formula is a formula which only quantifies over output positions, except inside an input predicate. We show that over non-erasing origin-graphs, any L*o*-formula can be transformed into an output formula. Intuitively, since we consider non-erasing origin-graphs, we can virtually quantify over the input by quantifying over the output, through the origin function.

Theorem 4.3.6. *Given an* L*o-formula ϕ, one can define an output* L*o-formula ϕ^{out} which is equivalent, over non-erasing origin-graphs. Furthermore, if ϕ is in SNF, then so is ϕ^{out}.*

Proof. We replace any quantification $\exists x\ \phi(x)$ by $\exists^{\mathsf{out}} x\ \phi(x) \vee \phi(\mathsf{o}(x))$, and $\forall x\ \phi(x)$ by $\forall^{\mathsf{out}} x\ \phi(x) \wedge \phi(\mathsf{o}(x))$. Since we only consider non-erasing origin-graphs, any input position is the origin of some output position, which means that the new formula is equivalent to the old one (over non-erasing origin-graphs). Since we don't change the alternation of quantifiers, if ϕ is in SNF then so is ϕ^{out}. □

Remark 4.3.7. Given a non-erasing formula, we can obtain an output formula which defines the same origin-graphs (when restricted to non-erasing origin-graphs). Hence if we have a uniformization algorithm for output formulas, we also have one for non-erasing formulas. In the following we will write $[\![\phi]\!]_o^{\mathrm{ne}} := \{g \in [\![\phi]\!]_o \mid\ g \text{ non-erasing}\}$.

Corollary 4.3.8. *Let ϕ be an* L*o-formula, and let $\mathcal{T}$ be an* MSO*-transducer uniformizing $[\![\phi^{\mathsf{out}}]\!]_o^{\mathrm{ne}}$. Then $\mathcal{T}$ uniformizes $[\![\phi]\!]_o$*

Proof. This is a direct consequence of Th. 4.3.6. □

4.3.4 Sets of constraints

With the previous three subsections we have shown that we only need to obtain a uniformization algorithm for output formulas of quantifier rank two, restricted to non-erasing origin-graphs. In the vein of [SZ12], we transform such a formula into a set of constraints, which are easier to manipulate.

Let A, B be alphabets. An *existential constraint* is a pair (b, E) with $b \in B$, E a finite set of triples $(c, d, \langle\phi\rangle)$ where $c \in B$, $d \in \{<_{\mathsf{out}}, =_{\mathsf{out}}, >_{\mathsf{out}}\}$ and $\langle\phi\rangle$ is an input predicate. Let g be an origin graph and let p be an output position of g. We say that g, p *satisfies* the constraint (b, E) if whenever p is labeled by b there exists an output position q and a triple $(c, d, \langle\phi\rangle) \in E$ such that q is labeled by c, p, q are related by d and the origins of p, q satisfy $\langle\phi\rangle$. In that case, such a position q is called a *witness* of (b, E) for p. Formally, g, p satisfy the constraint if we have $g, p \models b(x) \to \exists^{\mathsf{out}} y\ \bigvee_{(c,d,\langle\phi\rangle)\in E} c(y) \wedge (x\ d\ y) \wedge \langle\phi\rangle(\mathsf{o}(x), \mathsf{o}(y))$. We say that g *satisfies* the constraint if for any position p, we have that g, p satisfies the constraint.

A *universal constraint* is a tuple $(b, c, d, \langle\phi\rangle)$ with $b, c \in B$, $d \in \{<_{\mathsf{out}}, =_{\mathsf{out}}, >_{\mathsf{out}}\}$ and $\langle\phi\rangle$ is an input predicate. We say that a non-erasing origin-graph g and two output positions p, q do *not* satisfy the constraint if p, q are labeled b, c, respectively, ordered by d and their origins satisfy $\langle\phi\rangle$. Intuitively, a universal constraint can be thought of as a forbidden pattern. Formally, a graph g and two output positions p, q satisfy the constraint if we have $g, p, q \models \neg\,(b(x) \wedge c(y) \wedge (x\ d\ y) \wedge \langle\phi\rangle(\mathsf{o}(x), \mathsf{o}(y)))$. The graph g *satisfies* the universal constraint if for all output positions p, q then g, p, q satisfy it.

An instance of the *MSO Constraint Problem* (MCP) is a pair $C = (C_\exists, C_\forall)$ of finite sets of existential and universal constraints, respectively. A non-erasing origin-graph g is said to *satisfy* (or model) C if it satisfies each of its constraints, and we denote it by $g \models C$. We write $[\![C]\!]_o$ to denote the set of non-erasing origin graphs satisfying C.

Theorem 4.3.9. *Let ϕ be an output $\mathsf{L}o$-formula in SNF. We can construct C an MCP equivalent to ϕ over non-erasing origin-graphs.*

Proof. Let $\phi = \forall^{\mathsf{out}}x, y\ \chi(x, y) \wedge \bigwedge_{i\in\mathbf{n}} \forall^{\mathsf{out}}x\ \exists^{\mathsf{out}}y\ \beta_i(x, y)$ be an output $\mathsf{L}o$-formula in SNF. A *atomic binary type* for x, y is a truth value for all unary predicates $P(x)$, $P(y)$ and binary predicates $Q(x, y)$. Hence the formula $\chi(x, y)$ can be seen as a disjunction of all the allowed atomic binary types. Thus we can define $C_\forall$ as the set of all *forbidden* atomic binary types. Thus satisfying $\forall^{\mathsf{out}}x, y\chi(x, y)$ is equivalent to satisfying all the constraints of $C_\forall$.

By a case study analysis, we can rewrite the rest of the formula $\bigwedge_{i\in\mathbf{n}} \forall^{\mathsf{out}}x\ \exists^{\mathsf{out}}y\ \beta_i(x, y)$ as:

$$\forall^{\mathsf{out}}x \bigwedge_{i\in\mathbf{m}} b_i(x) \to \exists^{\mathsf{out}}y \bigvee_{j\in\mathbf{n_i}} c_{i,j}(y) \wedge (x\ d_{i,j}\ y) \wedge \langle\phi_{i,j}\rangle(\mathsf{o}(x), \mathsf{o}(y))$$

where $m \in \mathbb{N}$, for all $i \in \mathbf{m}$, $n_i \in \mathbb{N}$ and for all $j \in \mathbf{n_i}$, $c_{i,j} \in B$, $d_{i,j} \in \{<_{\mathsf{out}}, =_{\mathsf{out}}, >_{\mathsf{out}}\}$, and $\langle\phi_{i,j}\rangle$ is an input predicate. For each $i \in \mathbf{m}$ we define an existential constraint (b_i, E_i) with $E_i := \{(c_{i,j}, d_{i,j}, \langle\phi_{i,j}\rangle) \mid j \in \mathbf{n_i}\}$. We define $C_\exists$ as the set of all such constraints and thus, satisfying the original formula is equivalent to satisfying $C_\exists$. ☐

Corollary 4.3.10. *Let ϕ be an output $\mathsf{L}o$-formula in SNF, let C be the obtained equivalent MCP, and let $\mathcal{T}$ be an MSO-transducer uniformizing $[\![C]\!]_o$. Then $\mathcal{T}$ uniformizes $[\![\phi]\!]_o^{\mathrm{ne}}$.*

Proof. This is a direct consequence of Th. 4.3.9. ☐

In the following lemma we sum up the reduction results of the section.

Lemma 4.3.11. *Given a regular uniformization algorithm for the MCP, we have a regular uniformization algorithm for $\mathsf{L}o$.*

Proof. This is obtained by combining the reductions of the previous subsections, Lem. 4.3.2, Lem. 4.3.5, Cor. 4.3.8 and Cor. 4.3.10. ☐

4.4 Uniformization algorithm

We have reduced the initial synthesis problem to the regular synthesis problem of MCPs. Indeed, according to Lem. 4.3.11, it suffices to exhibit a regular uniformization algorithm for MCPs in order to obtain a regular uniformization algorithm for $\mathsf{L}o$. The main ideas used to solve this problem are inspired from [SZ12] where the authors showed the decidability of first-order logic with two variables over data words with the linear order and the data order. We extend these ideas to our setting, and refine them to obtain not only decidable satisfiability but also effective

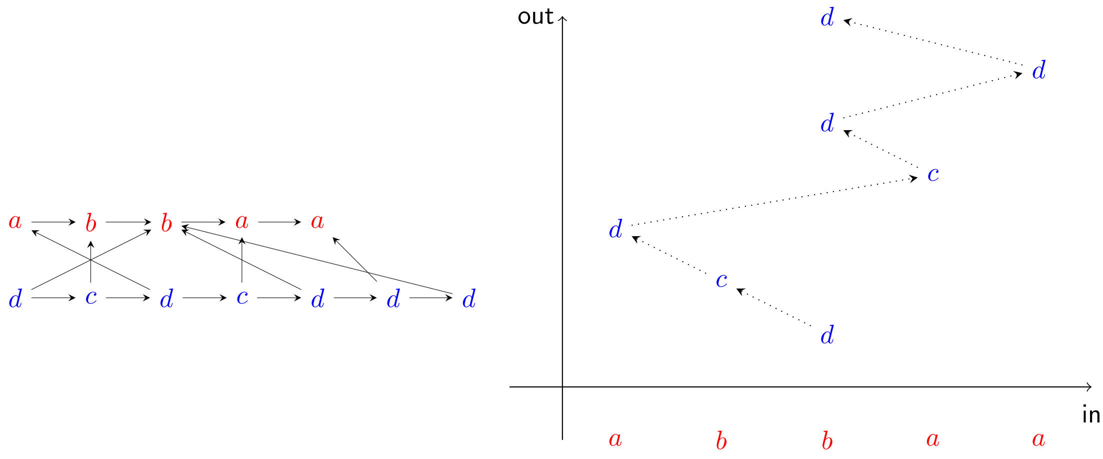


Figure 4.4: An origin-graph (left) seen as a two-dimensional structure (right).

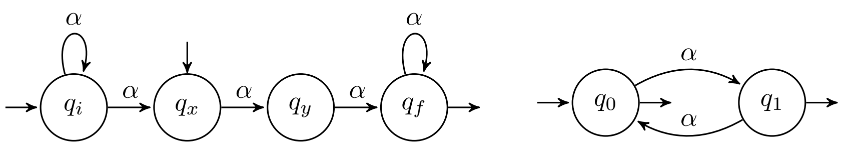


Figure 4.5: Two predicate automata. The left one recognizes the successor predicate with selecting pair (q_x, q_y). The right one recognizes the even distance predicate with selecting pairs (q_0, q_0) and (q_1, q_1).

regular synthesis for Lo. MSO binary predicates are replaced by *binary predicate automata* recognizing words with two distinguished positions x, y. We enrich origin-graphs with extra information, and we call these new objects *full profile graphs*. A full profile graph contains for each input position a copy of the output, with additional information about the binary predicates. From a full profile graph we define its *reduced profile graph*, which only keeps a bounded number of output positions for each input position. We show however that if a full graph satisfies an existential or universal constraint, then its reduced graph also does. Finally we show that if a reduced graph is valid (with respect to an MCP) and locally consistent (defined later) then one can construct an associated full graph. Since these properties are regular, we can recognize the set of valid and consistent reduced graphs, and by constructing the associated origin-graph, we obtain a uniformization of the original transduction.

A very useful tool will be to visualize an origin graph as a two dimensional structure with the input in the abscissa and the output in the ordinate, just as seen in Fig. 4.4.

4.4.1 Predicate automata

Before starting solving the problem, we replace MSO-formulas by automata, which are easier to deal with. We define the notion of predicate automata, a common tool to study MSO queries (see *e.g.* [NPTT05]).

A (binary) *predicate automaton* over an alphabet A is a pair $(\mathcal{A}, SP)$ where $\mathcal{A}$ is an automaton and $SP \subseteq Q \times Q$ is the set of *selecting pairs*, with Q the set of states of $\mathcal{A}$. Given a word u and

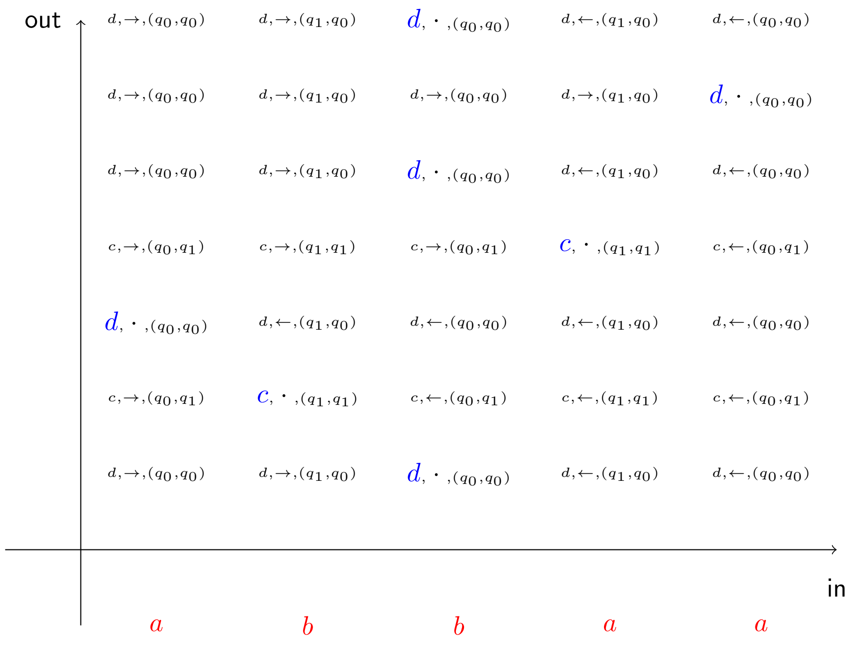


Figure 4.6: The full profile sequence of an origin-graph.

two positions $i, j \leq |u|$, we say that the triple u, i, j is recognized by $(\mathcal{A}, SP)$ if there exists an accepting run r of $\mathcal{A}$ over u such that $(r(i), r(j)) \in SP$. Given $\phi(x, y)$ a binary $\mathsf{MSO}[\leq]$-definable predicate over an alphabet A, we can define an equivalent predicate automaton $(\mathcal{A}_\phi, S_\phi)$ over A. Given $\Phi = \{\phi_1, \dots, \phi_n\}$ a set of binary $\mathsf{MSO}[\leq]$ predicates, we can take the union of all the predicate automata of each ϕ_i, $i \in \mathbf{n}$, and thus we obtain a unique automaton $\mathcal{A}_\Phi$ and one set of selecting pairs for each predicate, $\{SP_{\phi_1}, \dots, SP_{\phi_n}\}$.

Example 4.4.1. We give in Fig. 4.5 on the preceding page a predicate automaton for the successor predicate, where the only selecting pair is $\{(q_x, q_y)\}$. We also give an automaton for the even distance predicate.

4.4.2 Profiles

The notion of profile is a bit intricate but it is crucial to our approach. The profile of an input position in an origin-graph keeps information about the output positions it produces and also about other output positions.

Clauses Let A, B be alphabets, let C be an MCP instance, let $\mathcal{A}_\Phi$ be the automaton of the MSO input predicates appearing in C, let Q_Φ be the state space of $\mathcal{A}_\Phi$ and let $S_\Phi := \mathbf{2}^{Q_\Phi \times Q_\Phi}$. A *clause* for C is an element of the alphabet $B \times \{\cdot, \rightarrow, \leftarrow\} \times S_\Phi$. Clauses of the form $(b, \cdot, P)$ are called *local clauses* and talk about the label of an output position, and the MSO type of its origin (in a local clause P will only contain pairs of the form (p, p)). A clause of the form $(b, \rightarrow, P)$ (resp. $(b, \leftarrow, P)$) is called a *consistency clause* and states that there is an output position labeled b whose origin is greater (resp. smaller) than the current position and such that the binary

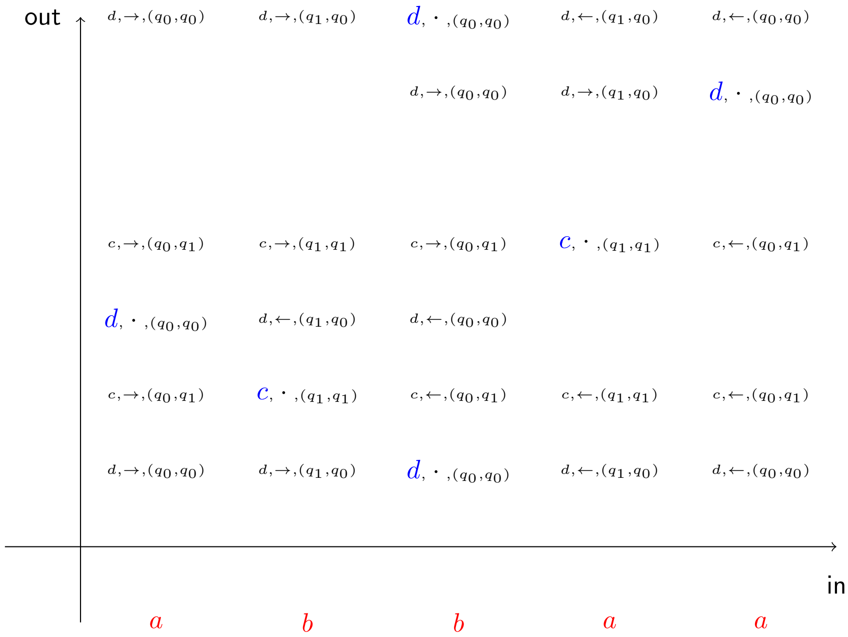


Figure 4.7: The reduced profile sequence of an origin-graph.

MSO type of these two positions corresponds to the pairs of states P. The number of clauses is bounded by $N_C := |B| \cdot 3 \cdot 2^{|Q_\Phi|^2}$.

Profile A C-*profile* (or just profile for short) is a sequence of the form $\lambda = (a, S, A_1, \dots, A_k)$ where $a \in A$, $S \subseteq Q_\Phi$ is a subset of states of the predicate automaton, and A_j is a clause for any $j \in \mathbf{k}$. A clause occurrence A_j in such a profile λ is called *extremal* if either for all clauses $A_{j'} = A_j$ we have $j' \leq j$ (A_j is maximal) or for all clauses $A_{j'} = A_j$ we have $j' \geq j$ (A_j is minimal). In the following, for $j \in \mathbf{k}$ we will often say *the clause* A_j to refer to the occurrence of the clause corresponding to the jth clause of λ.

A profile is called *reduced* if any type of clause occurs at most two times. The *reduction* of a profile λ is the reduced profile $\rho(\lambda)$ obtained by removing all non extremal clauses, *i.e.* for each type of clause only the two outermost occurrences are kept.

The *input* of a profile sequence $s = \lambda_1, \dots, \lambda_n$, with $\lambda_i = \left(a_i, S_i, A_1^i, \dots, A_{k_i}^i\right)$ is the word $\mathrm{in}(s) = a_1 \cdots a_n$. Note that the number of reduced profiles is less than $|A| \cdot 2^{|Q_\Phi|} \cdot N_C^{2N_C+1}$.

Let $g = (u, (v, o))$ be an origin-graph. The *full profile* of an input position $i \in \{1, \dots, |u|\}$ of g is the sequence $\lambda_i := (a, S, A_1, \dots, A_{|v|})$, where $a = u(i)$, and S is the set of states p such that there is an accepting run r of $\mathcal{A}_\Phi$ over u with $r(i) = p$. In short, the set S characterizes the unary MSO-type of position i of u. For $j \in \{1, \dots, |v|\}$, we have $A_j := (v(j), s_j, P_j)$, where s_j is $\cdot$ if $o(j) = i$, $\rightarrow$ if $o(j) > i$ and $\leftarrow$ if $o(j) < i$. The set P_j is defined as the set of pairs of states (p, q) such that there is an accepting run r of $\mathcal{A}_\Phi$ over u where $r(i) = p$ and $r(o(j)) = q$. Intuitively, the clause A_j keeps track of the label of position j as well as the binary predicates satisfied at positions $(i, o(j))$. Note that if A_j is local, then $P_j = \{(p, p) \mid p \in S\}$. The *reduced profile* (sometimes just profile) of an input position is the reduction of its full profile. The *full profile sequence* of $g = (u, (v, o))$ is the sequence $\mathrm{fSeq}(g) = \lambda_1, \dots, \lambda_{|u|}$ where λ_i is the full profile sequence of the ith position of u. The *(reduced) profile sequence* of q is the sequence $\mathrm{Seq}(g) = \rho(\lambda_1), \ldots, \rho(\lambda_{|u|})$.

Example 4.4.2. In Fig. 4.6 and 4.7 we give the full profile sequence and the reduced profile sequence, respectively, of the origin graph from Fig. 4.4 on page 104. The profiles contain the information relative to the even distance predicate automaton from Fig. 4.5 on page 104. In this two-dimensional representation, profiles are given vertically.

4.4.3 Validity

We define validity of profiles and of profile sequences with respect to an MCP instance C, and show that origin-graphs satisfying C have valid full profile sequences, which in turn have valid reduced profile sequences.

Let C be an MCP, and let $\lambda = (a, S, A_1, \ldots, A_k)$ be a profile. Let (b, E) be an existential constraint of C. We say that λ satisfies (b, E) if for all $j \in \{1, \ldots, k\}$ where A_k is a local clause of the form $(b, \cdot)$ there exist $(c, d, \langle\phi\rangle) \in E$, and $j' \in \{1, \ldots, k\}$ such that $j\ d\ j'$, $A_{j'} = (c, s, P)$, and $P \cap SP_\phi \neq \varnothing$. The clause $A_{j'}$ is called a *witness* of (b, E) for A_j. Let $(b, c, d, \langle\phi\rangle)$ be a universal constraint. The profile λ satisfies $(b, c, d, \langle\phi\rangle)$ if for all $j, j' \in \{1, \ldots, k\}$, it is *not* the case that A_j is a local clause with label b, $A_{j'} = (c, s, P)$, $j\ d\ j'$ and $P \cap SP_\phi \neq \varnothing$. The profile λ is *valid* for C if it satisfies each of its constraints. A profile sequence is called *valid* for C (or just valid when C is clear from context) if each of its profiles is valid.

We now show the equivalence between being a model and having a valid full profile sequence.

Proposition 4.4.3. *Let C be an MCP, then $g \models C$ if and only if* $\mathrm{fSeq}(g)$ *is valid.*

Proof. Let $g = (u, (v, o))$ be an origin-graph and let $\mathrm{fSeq}(g) = \lambda_1, \ldots, \lambda_{|u|}$ be its full profile sequence. We first show that g satisfies an existential constraint if and only if $\mathrm{fSeq}(g)$ does, and then we do the same for universal constraints.

Let (b, E) be an existential constraint of C satisfied by $\mathrm{fSeq}(g)$ and let $j \in \{1, \ldots, |v|\}$ be such that $v(j) = b$ and $o(j) = i \in \{1, \ldots, |u|\}$. Let $\lambda_i = (a, S, A_1, \ldots, A_{|v|})$ be the ith full profile of g, then we have $A_j = (b, \cdot, P)$. Since λ_i satisfies the constraint, there exists $A_{j'} = (c, s, P')$, $j' \in \{1, \ldots, |v|\}$, a witness of the constraint for A_j. This means that there exists $(c, d, \langle\phi\rangle) \in E$ with $j\ d\ j'$ and $P' \cap SP_\phi \neq \varnothing$. Hence, by definition of the predicate automaton, we have $u \models \phi(i, \mathsf{o}(j'))$. Hence we have that g satisfies (b, E).

Conversely let us assume that g satisfies the constraint (b, E), let $i \in \{1, \ldots, |u|\}$, let $\lambda_i = \big(a, S, A_1, \ldots, A_{|v|}\big)$ and let $A_j = (b, \cdot, P)$. Since g satisfies C, there is a witness j' of (b, E) for j. Thus we know that there is $(c, d, \langle\phi\rangle) \in E$ such that $j\ d\ j'$, $v(j') = c$ and $u \models \phi(\mathsf{o}(j), \mathsf{o}(j'))$. By definition we have $A_{j'} = (c, s, P)$ such that $P \cap SP_\phi \neq \varnothing$. Hence, λ_i satisfies (b, E), and thus $\mathrm{fSeq}(g)$ satisfies (b, E).

Let $(b, c, d, \langle\phi\rangle)$ be a universal constraint of C, and let us assume that λ_i violates the constraint for some $i \in \{1, \ldots, |u|\}$. This means that there exist j, j' such that $A_j = (b, \cdot, P)$, $A_{j'} = (c, s, P')$ such that $j\ d\ j'$ and $P' \cap SP_\phi \neq \varnothing$. By definition of λ, this means that $v(j) = b$, $v(j') = c$, $j\ d\ j'$ and $u \models \phi(\mathsf{o}(j), \mathsf{o}(j'))$ and thus g also violates the constraint.

Let us finally assume that g violates a universal constraint. Let $(b, c, d, \langle\phi\rangle)$ be a universal constraint and let $j, j' \in \{1, \ldots, |v|\}$ such that $v(j) = b$, $v(j') = c$, $j\ d\ j'$ and $u \models \phi(\mathsf{o}(j), \mathsf{o}(j'))$. This means, by definition of the full profile of $i = \mathsf{o}(j)$ that it has two clauses $A_j = (b, \cdot, P)$ and $A_{j'} = (c, s, P')$ where $P' \cap SP_\phi \neq \varnothing$ and thus λ_i violates the constraint.

□

We now show that reduction of profiles preserves validity.

Proposition 4.4.4. *Let C be an MCP, and let λ be a valid profile. The profile $\rho(\lambda)$ is valid.*

Proof. Let C be an MCP and let $\lambda = (a, S, A_1, \ldots, A_k)$ be a valid profile. Since $\rho(\lambda)$ is a subsequence of λ, we have that $\rho(\lambda)$ satisfies any universal constraint satisfied by λ.

Let $l \leq k$ and let $1 \leq i_1 \leq \ldots \leq i_l \leq k$ such that $\rho(\lambda) = (a, S, A_{i_1}, \ldots, A_{i_l})$. Let (b, E) be an existential constraint of C, and let $j \in \{1, \ldots, l\}$ such that $A_{i_j} = (b, \cdot, P)$. Since λ is valid, there exists $A_{j'}$ a witness of (b, E) for A_{i_j} with $i_j\ d\ j'$ (with $d \in \{=, <, >\}$). However, the reduction of a profile does not remove extremal clauses, thus there exists $j'' \in \{1, \ldots, l\}$ such that $A_{j'} = A_{i_{j''}}$ either $j' = i_{j''}$ or $j''\ d\ i_{j''}$. Either way we have that $i_j\ d\ i_{j''}$ and thus $A_{i_{j''}}$ is a witness of (b, E) for A_{i_j}. □

2-bounded origin-graphs We define a notion of small models which we call *2-bounded.* An origin-graph over A, B is called *2-bounded* if for any input position and any output letter $b \in B$, the input position produces at most two output positions labeled by b. Reduced profiles have at most two occurrences of each local clause. Thus we will show that the profile sequence of any origin-graph is equal to the profile sequence of some 2-bounded origin-graph. We show that a 2-bounded origin-graph satisfies an MCP C if and only if its reduced profile sequence is valid.

Proposition 4.4.5. *Let C be an MCP and let g be a 2-bounded origin-graph. Then $g \models C$ if and only if* $\mathrm{Seq}(g)$ *is valid.*

Proof. Let C be an MCP and let $g = (u, (v, o))$ be a 2-bounded origin-graph. If $g \models C$ then $\mathrm{fSeq}(g)$ is valid, according to Prop. 4.4.3. Then from Prop. 4.4.4, $\mathrm{Seq}(g) = \rho(\mathrm{fSeq}(g))$ is also valid.

We have left to show that if $\mathrm{Seq}(g)$ is valid then so is $\mathrm{fSeq}(g)$. Let $i \in \{1, \ldots, |u|\}$ let $\lambda_i = \big(a, S, A_1, \ldots, A_{|v|}\big)$ be the full profile of position i. Let $1 \leq i_1 \leq \ldots \leq i_l \leq k$ be such that $\rho(\lambda_i) = (a, S, A_{i_1}, \ldots, A_{i_l})$.

Let (b, E) be an existential constraint satisfied by $\rho(\lambda_i)$, and let $A_j = (b, \cdot, P)$. Since g is 2-bounded, reducing a profile cannot erase a local clause, thus we have $A_j = A_{i_m}$ for some $m \in \mathbf{l}$. Since $\rho(\lambda_i)$ satisfies (b, E) there exists A_{i_n} a witness of it for A_j. Thus A_{i_n} is also a witness of (b, E) for A_j, since λ_i is a supersequence of $\rho(\lambda_i)$. Hence λ_i also satisfies (b, E).

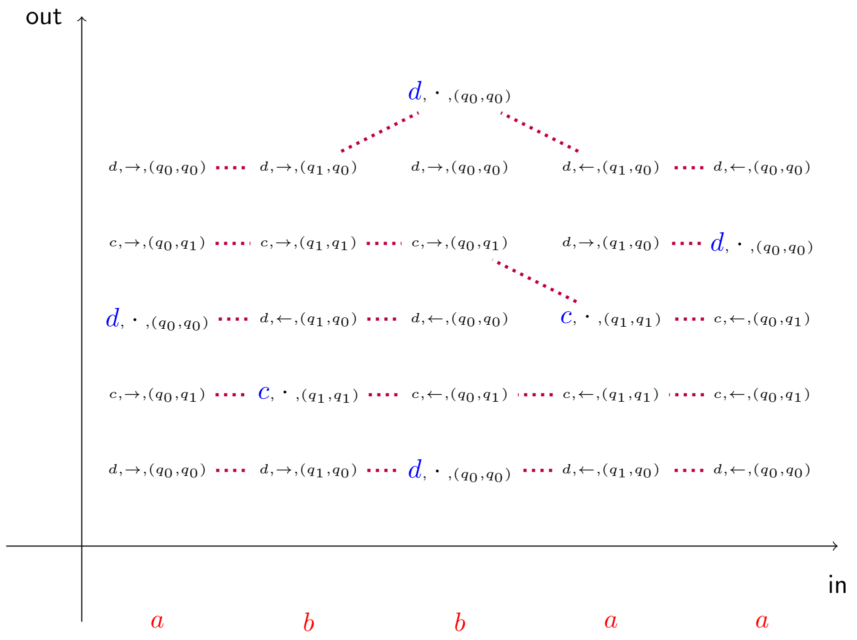
Let $(b, c, d, \langle\phi\rangle)$ be a universal constraint violated by λ_i, this means that there are two clauses $A_j = (b, \cdot, P)$ and $A_{j'} = (c, s, P')$ such that $j\ d\ j'$ and $P' \cap SP_\phi \neq \varnothing$. Since A_j is a local clause, it is not removed in $\rho(\lambda_i)$. Furthermore, we can assume without loss of generality that $A_{j'}$ is an extremal occurrence of the clause and thus it is not removed either in $\rho(\lambda_i)$. Hence $\rho(\lambda_i)$ also violates the constraint, which concludes the proof.

□

4.4.4 Consistency

Consistency is a property of profile sequences that ensures compatibility of adjacent profiles. Before defining consistency, we need the key notion of successor clauses.

Successor clauses We define the (unique) successor of a clause which is either local or pointing to the left. Similarly, the predecessor of a clause either local or pointing to the right is also unique. Let C be an MCP over A, B, let $\mathcal{A}_\Phi$ be the associated predicate automaton with state space Q_Φ. Let $a, a' \in A$, let $S, S' \subseteq Q_\Phi$ and let K be a clause of the form $(b, \cdot, P)$ or $(b, \leftarrow, P)$. Then *the successor of* K with respect to a, a', S, S' is the clause $K' = (b, \leftarrow, P')$ with $P' = \left\{(p', q) \in S' \times S \mid \exists (p, q) \in P,\ p \xrightarrow{a}_{\mathcal{A}_\Phi} p'\right\}$. Let $a, a' \in A$, let $S, S' \subseteq Q_\Phi$ and let K' be a



out
$d, \cdot, (q_0,q_0)$
$d,\rightarrow,(q_0,q_0)$ $d,\rightarrow,(q_1,q_0)$ $d,\rightarrow,(q_0,q_0)$ $d,\leftarrow,(q_1,q_0)$ $d,\leftarrow,(q_0,q_0)$
$c,\rightarrow,(q_0,q_1)$ $c,\rightarrow,(q_1,q_1)$ $c,\rightarrow,(q_0,q_1)$ $d,\rightarrow,(q_1,q_0)$ $d, \cdot, (q_0,q_0)$
$d, \cdot, (q_0,q_0)$ $d,\leftarrow,(q_1,q_0)$ $d,\leftarrow,(q_0,q_0)$ $c, \cdot, (q_1,q_1)$ $c,\leftarrow,(q_0,q_1)$
$c,\rightarrow,(q_0,q_1)$ $c, \cdot, (q_1,q_1)$ $c,\leftarrow,(q_0,q_1)$ $c,\leftarrow,(q_1,q_1)$ $c,\leftarrow,(q_0,q_1)$
$d,\rightarrow,(q_0,q_0)$ $d,\rightarrow,(q_1,q_0)$ $d, \cdot, (q_0,q_0)$ $d,\leftarrow,(q_1,q_0)$ $d,\leftarrow,(q_0,q_0)$
in
a b b a a

Figure 4.8: The reduced profile sequence of an origin-graph, with linked clauses (dotted).

clause of the form $(b, \cdot, P')$ or $(b, \rightarrow, P')$. Then *the predecessor of* K' with respect to a, a', S, S' is the clause $K = (b, \rightarrow, P)$ with $P = \left\{(p, q) \in S \times S' \mid \exists (p', q) \in P',\ p \xrightarrow{a}_{\mathcal{A}_\Phi} p'\right\}$. A pair of profiles K, K' such that either K' is a successor of K or K is a predecessor of K' is called an *adjacent* pair of clauses.

Example 4.4.6. In particular all pairs of clauses linked by a dotted line in Fig. 4.8 are adjacent.

Linked clauses Before defining consistency, we define the notion of *linked* clauses in two profiles. Let $\lambda = (a, S, A_1, \dots, A_k)$ and $\mu = (b, T, B_1, \dots, B_l)$ be two profiles. Let $i \in \mathbf{k}, j \in \mathbf{l}$ such that A_i, B_j is an adjacent pair. We say that A_i and B_j are *linked*, with respect to λ, μ, if one of the two following cases occurs:

- the index i is maximal among clauses of λ adjacent to B_j and the index j is maximal among clauses of μ such that A_i is adjacent to them.
- the index i is minimal among clauses of λ adjacent to B_j and the index j is minimal among clauses of μ such that A_i is adjacent to them.

We denote that A_i and B_j are linked by $A_i\ \ell_{\lambda,\mu}\ B_j$ (the profiles λ, μ are often left implicit). Given a profile sequence $s = \lambda_1, \dots, \lambda_n$, we define the *link closure* of s by ℓ_s^*, the symmetric and transitive closure of $\bigcup_{1 \le i < n} \ell_{\lambda_i, \lambda_{i+1}}$.

Example 4.4.7. The linked clauses are represented by dotted lines in Fig. 4.8.

Consistency of profiles Let us define the notion of *consistent profiles*. Intuitively, two profiles are consistent if they are the reductions of two profiles of same size such that for any i the ith clause of the first profile and the ith clause of the second are adjacent. Formally, let $\lambda = (a, S, A_1, \dots, A_k)$ and $\mu = (b, T, B_1, \dots, B_l)$ be two profiles. The pair λ, μ is called *consistent* if 1) each clause of λ is adjacent to some clause of μ and *vice versa*, 2) each clause is linked to at most one clause, and 3) there are no *crossing links i.e.* links A_i, B_j and $A_{i'}, B_{j'}$ such that $i < i'$ and $j > j'$, or such that $i > i'$ and $j < j'$. A profile sequence $\lambda_1, \dots, \lambda_n$ is *consistent* if for all $i \in \{1, \dots, n-1\}$, the pair λ_i, λ_{i+1} is consistent.

Proposition 4.4.8. *Let C be an MCP and let g be an origin-graph. The sequence* $\mathrm{Seq}(g)$ *is consistent.*

Proof. Let $g = (u, (v, o))$ be an origin-graph, and let us first show that its full profile sequence is consistent. Let $i \in \{1, \dots, |u| - 1\}$ and let $\lambda_i = (a, S, A_1, \dots, A_{|v|})$ and $\lambda_{i+1} = (b, T, B_1, \dots, B_{|v|})$ be the full profiles of positions i and $i + 1$, respectively. By definition, each pair A_j, B_j is an adjacent pair, for $j \in \{1, \dots, |v|\}$. Since we never erase all occurrences of a clause, we have that each clause of $\rho(\lambda_i)$ is adjacent to some clause of $\rho(\lambda_{i+1})$, and *vice versa.* Furthermore, by definition of linked clauses, non-extremal clauses can never be part of a linked pair for λ_i, λ_{i+1}. Thus we obtain that linked pairs for λ_i, λ_{i+1} are necessarily of the form A_j, B_j where both A_j and B_j are both minimal or both maximal occurrences of their respective clause. Thus, by removing non-extremal clauses, we preserve the linked pairs, and we don't create new ones, which means that each clause is linked to at most one other clause, and hence λ_i, λ_{i+1} is consistent, which concludes the proof. □

4.4.5 Complete profile sequences

The notion of completeness pertains to profile sequences and ensures that all the set of states and pairs of states in the profiles contain all possible states of accepting runs. A profile is called *initial* if it does not contain a clause pointing to the left and *final* if it does not contain clauses pointing to the right.

Link closure Before defining completeness, we need some properties of the link closure of reduced and consistent profile sequences. Given a profile sequence s, we call *the graph of l_s^** the graph of clause occurrences of s, with edges of l_s^*.

Lemma 4.4.9. *Let s be a consistent profile sequence. The following holds: each connected component of ℓ_s^* is a path with at most one local clause.*

Proof. By definition of consistency, a clause is linked at most once on the left and once on the right, which means that connected components of ℓ_s^* are paths. A clause pointing to the right can only be followed by a clause pointing to the right or a local clause, a local clause can only be followed by a clause pointing to the left and clauses pointing to the left must be followed by a clause pointing to the left. This means that each path of ℓ_s^* contains at most one local clause. □

We consider the link closure of consistent reduced profile sequences and show that non-local clauses are linked in the direction they point to.

Lemma 4.4.10. *Let $s = \lambda_1, \dots, \lambda_n$ be a consistent reduced profile sequence, with $n \geq 2$. For any $i \in \{1, \dots, n-1\}$, any clause of type $(b, \rightarrow, P)$ of λ_i is linked to a clause of λ_{i+1}. Symmetrically, for any $i \in \{2, \dots, n\}$, any clause of type $(b, \leftarrow, P)$ of λ_i is linked a clause of λ_{i-1}.*

Proof. Let $s = \lambda_1, \dots, \lambda_n$ be a consistent reduced profile sequence, with $n \geq 2$. Let $i \in \{1, \dots, n-1\}$, let $\lambda_i = (a, S, A_1, \dots, A_k)$ and let $A_j = (b, \rightarrow, P)$. Since λ_i is reduced, A_j is either the maximal or minimal (or both) occurrence of the clause. Let us assume without loss of generality that A_j is the maximal occurrence. If we consider all clauses of λ_{i+1} whose predecessor is A_j, then A_j is linked to the maximal of these clauses, by consistency. The symmetric case is proved in the same way. □

Combining the two previous lemmas we obtain the following property of consistent and reduced profile sequences.

Corollary 4.4.11. *Let s be a consistent reduced profile sequence beginning with an initial profile and ending with a final one. Each connected component of ℓ^*_s is a path with exactly one local clause.*

Proof. The result follows from Lem. 4.4.9 and 4.4.10. □

Example 4.4.12. In Fig. 4.8 on page 109, we consider the reduced profile sequence of the origin-graph from Fig. 4.4 on page 104 where we added the links between clauses. Note that each linked component is a path and that each path contains exactly one local clause.

Complete sequences We now have the tools to define complete sequences. Intuitively, a complete sequence does not conceal any predicate information in the clauses. Let $s = \lambda_1, \ldots, \lambda_n$ be a consistent reduced profile sequence beginning with an initial profile and ending with a final one, and let $u = \text{in}(s)$.

Let $A_j = (b, d, P)$ be a clause of λ_i, let i' be the index of the profile containing the local clause linked to A_j by ℓ^* (which exists and is unique by Cor. 4.4.11). The clause A_j is *complete* with respect to s if P is equal to the set of pairs (p, q) such that there is an accepting run r of $\mathcal{A}_\Phi$ over u with $r(i) = p$ and $r(i') = q$. Then the sequence s is called *complete* if it satisfies the following properties:

- For all $i \in \mathbf{n}$, S_i is equal to the set of states p such that there is an accepting run r of $\mathcal{A}_\Phi$ over u with $r(i) = p$.
- For all $i \in \mathbf{n}$, all clauses of λ_i are complete with respect to s.

We now show that the profile sequence of an origin-graph is complete.

Theorem 4.4.13. *Let C be an MCP, and let g be an origin-graph. The sequence* $\text{Seq}(g)$ *is complete.*

Proof. Let C be an MCP, and let $g = (u, (v, o))$ be an origin-graph. By definition, the first profile of $\text{Seq}(g)$ is initial and the last one is final. According to Prop. 4.4.8, $\text{Seq}(g)$ is consistent. Let $\text{Seq}(g) = \lambda_1, \ldots, \lambda_n$, let $i \in \mathbf{n}$ and let $\lambda_i = (a, S, A_1, \ldots, A_k)$. Again, by definition, we have that S is complete.

As we have seen in the proof of Prop. 4.4.8, links in $\text{Seq}(g)$ coincide with the links in $\text{fSeq}(g)$ thus the local clause linked to a clause of $\text{Seq}(g)$ is the same clause in $\text{fSeq}(g)$, which is complete by definition of full profiles. This means that all clauses of $\text{Seq}(g)$ are complete. □

4.4.6 Soundness

We have shown that a model of an MCP has a valid (Prop. 4.4.3 and 4.4.4) and a complete (Th. 4.4.13) reduced profile sequence. Our goal is now to obtain the converse result that any complete and valid profile sequence is the profile sequences of some model of C. Before that we define a partial order on local clauses, the linearization of which will yield an origin graph.

Let $s = \lambda_1, \ldots, \lambda_n$ be a consistent profile sequence. Let $j < k$ be two integers such that A_j and B_k are two distinct local clauses of some profiles of s. If there exists a profile $\lambda_i = (a, S, C_1, \ldots, C_m)$ with two clauses $C_{j'}$, $C_{k'}$ such that $A_j \ \ell^* \ C_{j'}$, $B_k \ \ell^* \ C_{k'}$ and $j' < k'$ then we set $A_j \rightarrow_s B_k$.

Example 4.4.14. Let us consider the profile sequence s of Fig. 4.8 on page 109, which is consistent by Prop. 4.4.8. Let C_2 be the local clause of the second profile and let C_5 be the local clause of the last profile. Since C_2 is linked to a clause of the last profile, below C_5, then we have $C_2 \rightarrow_s C_5$. Let C_1 be the local clause of the first profile, then C_1 and C_5 are incomparable with respect to $\rightarrow_s$ because their link paths never intersect the same profile.

Proposition 4.4.15. *Let s be a consistent profile sequence, then $\rightarrow_s$ over occurrences of local clauses of s is acyclic.*

Proof. Let $s = \lambda_1, \ldots, \lambda_n$ be a consistent profile sequence. We map clause occurrences into the two-dimensional plane such that if we have $A \rightarrow_s B$ then A has a strictly lower ordinate than B, which will conclude the proof. Let $\lambda_i = (a_i, S_i, A^i_1, \ldots, A^i_{k_i})$ for $i \in \mathbf{n}$. We start by setting $\mu(A^1_j) := (1, j)$, for $j \in \mathbf{i_1}$, and then we define μ by induction. For any clause A^{i+1}_j, if it is linked to some clause $A^i_{j'}$ with $\mu(A^i_{j'}) = (i, q)$ then $\mu(A^{i+1}_j) := (i+1, q)$. Let $1 \leq j_1 < \ldots < j_k \leq k_{i+1}$ be the indexes of the linked clauses and let $q_1 < \ldots < q_k$ be their respective ordinates (they are ordered the same, since there are no crossing links by consistency). Let $j < j_1$, then $\mu(A^{i+1}_j) := (i+1, q_1 - (j_1 - j))$. Let $j_l < j < j_{l+1}$, then $\mu(A^{i+1}_j) := (i+1, q_l + \frac{j - j_l}{j_{l+1} - j_l}(q_{l+1} - q_l)$. Let $j > j_k$, then $\mu(A^{i+1}_j) := (i+1, q_k + (j - j_k))$. Hence by definition, linked pairs have the same ordinate, while two clauses where $A \rightarrow_s B$ implies that the ordinate of A is strictly smaller than that of B, concluding the proof. $\square$

From complete sequences to origin-graphs We define a construction from a complete profile sequence to an origin-graph and show some of its properties. Let $s = \lambda_1, \ldots, \lambda_n$ be a profile sequence with $\lambda_i = (a_i, S_i, A^i_1, \ldots, A^i_{k_i})$ and let $\leq_s$ be a linear order over local clauses, compatible with $\rightarrow_s$ (*i.e.* $A \rightarrow_s B \Rightarrow A \leq_s B$). Let us define $g = (u, (v, o))$ the *origin-graph of s induced by $\leq_s$*, by $u := a_1 \cdots a_n$, $\mathrm{dom}(v) := \left\{ A^i_j \text{ local} \mid \ i \in \mathbf{n}, j \in \mathbf{k_i} \right\}$ with $\leq^v := \leq_s$, $b^v := \left\{ A^i_j \in \mathrm{dom}(v) \mid \ A^i_j = (b, \cdot, P) \right\}$ for $b \in B$, and $o(A^i_j) := i$, for $i \in \mathbf{n}, j \in \mathbf{k_i}$.

Lemma 4.4.16. *Let s be a complete profile sequence, let $\leq_s$ be a linear order compatible with $\rightarrow_s$ and let g be the origin-graph of s induced by $\leq_s$. Then* $\mathrm{Seq}(g) = s$.

Proof. Let $s = \lambda_1, \ldots, \lambda_n$ be a complete sequence, let $\leq_s$ be a linear order compatible with $\rightarrow_s$ and let $g = (u, (v, o))$ be the origin-graph of s induced by $\leq_s$. Let $i \in \mathbf{n}$, and let $\lambda_i = (a_i, S_i, A^i_1, \ldots, A^i_{k_i})$, then we have $a_1 \cdots a_n = u$. Let $s' = \mathrm{fSeq}(g) = \lambda'_1, \ldots, \lambda'_n$, let $i \in \mathbf{n}$, and let $\lambda'_i = (a_i, S'_i, A'^i_1, \ldots, A'^i_{k'_i})$. The sets S'_i only depend on u and i, and thus by completeness of s we have $S'_i = S_i$. The local clauses of s' are obtained from the local clauses of s and thus for each profile, the number of occurrences as well as the respective order of the local clauses is preserved.

Let us first show that s is a subsequence of s', the full profile sequence of g. Let $i \in \mathbf{n}$, let $1 \leq j < j' \leq k_i$, then A^i_j and $A^i_{j'}$ both point to some local clause in other profiles (Cor. 4.4.11). We have that these positions must be ordered by $\rightarrow^*_s$ (the transitive closure of $\rightarrow_s$) and thus by $\leq_s$, which means that these two clauses appear in the same order in the ith full profile of g.

Let us assume that $\lambda_i \neq \rho(\lambda'_i)$. This means that there is an extremal (let's assume maximal) occurrence of a clause of λ'_i which does not appear in λ_i. If the clause does not appear at all, then by consistency we obtain a contradiction. If this clause does appear in λ_i but lower, then this would imply crossing links with a clause that appears above in λ_i but below in λ'_i, contradicting consistency. Hence we have $\lambda_i \neq \rho(\lambda'_i)$, and thus $s = \mathrm{Seq}(g)$. □

We now have all the tools to characterize complete and valid profile sequences.

Lemma 4.4.17. *Let C be an MCP. We have $\{\mathrm{Seq}(g) \mid g \models C\} = \{s \mid$ s is complete and valid$\}$.*

Proof. The left-to-right inclusion comes from Prop. 4.4.3 and 4.4.4 concerning validity and from Th. 4.4.13 for completeness. The right to left inclusion comes from Lem. 4.4.16 and Prop. 4.4.5. □

4.4.7 Profile automaton

We show that the language of complete and valid profile sequences is effectively rational. Moreover, we have shown that all models of an MCP have complete and valid profile sequences and conversely that any complete and valid profile sequence is the profile sequence of some model. Thus we obtain a way to compute the domain of an MCP. As a consequence, using the reductions from Sec. 4.3, we can compute an automaton recognizing the domain of an Lo-transduction.

profile automaton

Lemma 4.4.18. *Let C be an MCP. The set of complete and valid profile sequences is effectively rational.*

Proof. Let C be an MCP. We consider P_C the alphabet of reduced profiles, and we show that the set of complete and valid profile sequences is rational. To this end we show that consistency, completeness and validity are all rational properties of a reduced profile sequence. First one can easily see that validity of a sequence only depends on the alphabet of profiles, so by reducing the alphabet to valid reduced profiles, we only consider valid sequences. Similarly, consistency is a local property, and a profile sequence is consistent if any pair of two consecutive profiles is consistent, which is a rational property. It is easy to check that the first profile of a sequence is initial and that the last one is final.

We have left to show that completeness can be checked rationally. Actually we show that non-completeness can be checked rationally. Let $s = \lambda_1, \ldots, \lambda_n$ be a profile sequence where $\lambda_i = \left(a_i, S_i, A^i_1, \ldots, A^i_{k_i}\right)$ for $i \in \mathbf{n}$. The sequence s is non-complete if either 1)there is a position $i \in \mathbf{n}$ such that S_i is not equal to the set of states reached in position i of an accepting run of $\mathcal{A}_\Phi$ over $a_1 \cdots a_n$ or 2) a clause of λ_i is not complete. To check non-completeness, we define an automaton that guesses a position i satisfying 1) or 2) and checks that the guess was correct. □

Rational domain As a consequence, we obtain that the domain of an MCP is also rational.

Corollary 4.4.19. *The domain of an MCP instance is effectively rational.*

Proof. Let C be an MCP instance over alphabets A, B. According to Lem. 4.4.17, we have $\{\mathrm{Seq}(g) \mid g \models C\} = \{s \mid s \text{ is complete and valid}\}$. Furthermore we have for any origin graph $g = (u, (v, o))$ that $u = \mathrm{dom}(\mathrm{Seq}(g))$. Thus $\mathrm{dom}(C) = \{\mathrm{dom}(s) \mid s \text{ is complete and valid}\}$, which is rational by Lem. 4.4.18, by projection of profiles over A (since rational languages are closed under projection). □

By the previous reductions we have that the domain of an Lo-transduction is rational.

Theorem 4.4.20. *The domain of an Lo-transduction is effectively rational.*

Proof. Each of the reductions of Sec. 4.3 preserves the domain of a transduction thus, according to Cor. 4.4.19, the domain of an Lo-transduction is rational. □

Satisfiability, validity and equivalence As a corollary, we obtain the decidability of Lo.

Theorem 4.4.21. *The satisfiability, validity and equivalence problems for* Lo*-formulas over origin-graphs are decidable.*

Proof. The decidability of the satisfiability problem is a consequence of Th. 4.4.20. The other problems are solved using the closure of Lo under boolean combinations. Given a formula ϕ, it is valid if and only if $\neg\phi$ is satisfiable. Similarly, given ϕ_1, ϕ_2, they are equivalent if and only if $\phi_1 \leftrightarrow \phi_2$ is valid. □

Remark 4.4.22. Note that the equivalence problem for Lo transductions is only decidable *with origins*. Indeed it is well known that equivalence of transductions up to origins is undecidable.

Since we translate the input MSO-predicates into predicate automata, we obtain an unavoidable complexity in TOWER for deciding satisfiability of Lo formulas. However when the input predicates are given directly as predicate automata, we obtain a tight elementary complexity.

Theorem 4.4.23. *The satisfiability of* Lo *is* EXPSPACE*-complete if the input predicates are given as predicate automata.*

Proof. The hardness result comes from [SZ12], where it is shown that satisfiability of $\mathsf{FO}^2[\leq, \preccurlyeq]$ over data words is PSPACE-hard. Equivalently (see Sec. 4.5) one can say that the satisfiability of $\mathsf{FO}^2[\leq_{\mathsf{out}}, \leq_{\mathsf{in}}, \mathsf{o}]$ over non-erasing origin-graphs with a unary input alphabet is PSPACE-hard.

To show that the problem can be decided in EXPSPACE, we use the fact that one can decide emptiness of the profile automaton in NLOGSPACE, with respect to the size of the profile automaton. Let ϕ be an Lo-formula over alphabets A, B with input predicates given by the predicate automaton $\mathcal{A}_\Phi$ with state space Q_Φ. Let us show that the profile automaton of the associated MCP instance has size exponential in $|\phi| + |Q_\Phi|$. Transforming ϕ into a non-erasing formula is linear, then putting it into SNF is also linear, however as we have seen, we obtain an output alphabet $B \times B'$ where B' has size exponential in ϕ. Finally going from a formula to an output formula is also linear and doesn't change the alphabet.

In Subsec. 4.4.2 we have seen that the number of different clauses is $N_C := |B \times B'| \cdot 3 \cdot 2^{|Q_\Phi|^2}$ which is exponential. We have also seen that the number of reduced profiles is upper bounded by $|A| \cdot 2^{|Q_\Phi|} \cdot N_C^{2N_C+1}$ which is doubly exponential. Since the set of states of the profile automaton is just the set of reduced profile we obtain that the profile automaton has size doubly exponential in ϕ and Q_Φ, which means that checking for emptiness can be done in EXPSPACE. □

4.4.8 Synthesis

The rationality of the language of complete and valid profile sequences actually gives us more than rationality of the domain of Lo-transductions. We can indeed use this to effectively uniformize Lo-transductions, and in particular solve the regular synthesis problem for Lo.

Theorem 4.4.24. *Given an MCP instance, one can synthesize a regular function uniformizing it.*

Proof. Let C be an MCP instance, we define several MSO-transductions whose composition uniformizes $[\![C]\!]_o$. We define $R := f_{\text{lin}} \circ f_{\text{DAG}} \circ f_{\text{link}} \circ R_{\text{Seq}}$, where R_{Seq} relates words u with valid and complete profile sequences s such that $\text{in}(s) = u$, f_{link} maps any consistent profile sequence to the graph of clauses with links, f_{DAG} maps a graph of clauses to the directed acyclic graph of local clauses from Prop. 4.4.15 and f_{lin} linearizes a DAG into a word.

A transducer realizing R_{Seq} is simply obtained from the profile automaton of Lem. 4.4.18, by replacing a transition of the form (p, λ, q) where $\text{in}(\lambda) = a$, by a transition reading a and outputting λ. To realize f_{link}, we define an MSO-transduction, which reads input words over P_C (the alphabet of profiles), copies each input position the number of its clauses times and links linked clauses. From the graph of clauses, we can define an MSO-transduction that erases all non local clauses and adds the $\to_s$ relation which is MSO-definable. According to Prop. 4.4.15, the graph of local clauses with $\to_s$ is a DAG. Finally we use a result of [Cou96] which states that there is an NMSO-transduction that takes a DAG as an input and produces a linearization of it. Additionally the DAG needs to be *locally ordered*, meaning that the successors of a given vertex must be ordered. This is the case for the graphs of $\to_s$ over local clauses since local clauses can be ordered by the input order and then the profile order. Thus we can define an MSO-transduction f_{lin} taking as input the graph of local clauses with $\to_s$, and linearizing it.

The equality of the domain of R and $[\![C]\!]_o$ comes from Cor. 4.4.19 where we show that input words of models of C are exactly the domain words of complete valid sequences. Furthermore, the fact that this transduction uniformizes $[\![C]\!]_o$, comes from Lem. 4.4.16 which states that the origin-graph obtained from a linearization of a valid and complete sequence is a model of C. Finally since NMSO-transductions are closed under composition [Cou90], we obtain an NMSO-transduction realizing R. Thus R can be uniformized by an MSO-transduction. □

We finally obtain our main result, by reducing synthesis for Lo-transductions to synthesis for MCP instances.

Theorem 4.4.25. *Given an* Lo*-formula, one can synthesize a regular function which uniformizes it.*

Proof. This is a simple consequence of the uniformization from Th. 4.4.24, and the properties of the reductions from Sec. 4.3. Indeed we have seen in Lem. 4.3.11 that having a regular uniformization algorithm for MCP instances yields a regular uniformization algorithm for Lo. □

As a consequence we can decide if an Lo-transduction is functional.

Corollary 4.4.26. *Given an* Lo*-formula* ϕ*, one can decide if* $[\![\phi]\!]_o$ *is functional.*

Proof. Let ϕ be an Lo-formula, and let $\mathcal{T}$ be an MSO-transducer uniformizing ϕ, which can be obtained by Th. 4.4.25. According to Th. 4.2.7, one can define an Lo-formula $\phi_{\mathcal{T}}$ such that $[\![\phi_{\mathcal{T}}]\!]_o = [\![\phi]\!]_o$. Thus the formula $\phi \leftrightarrow \phi_{\mathcal{T}}$ is valid if and only ϕ is functional, which is decidable from Th. 4.4.21. □

4.5 Words with ordered data

Data words have been studied recently as an extension of words over an infinite alphabet. Usually the alphabet is split into a finite part and an infinite part where the infinite part is restricted to only equality tests (see [Bou02]). Several attempts have been made to extend the expressiveness with additional predicates talking about the data, for instance [BDM+11, SZ12] have given decidable logics over data words with a linearly ordered data domain. However these extensions come at a price and the logics have to be weakened to keep decidability. We make a bijective connection between origin-graphs and *typed data words* which are just data words with ordered data and a two-sorted finite alphabet. We then introduce a new logic, Ld over data words whose decidability can be reduced to decidability of Lo. This logic extends the one of [SZ12] by adding arbitrary MSO predicates that talk only about the data.

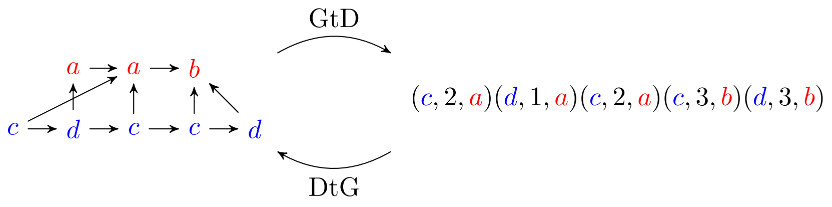


Figure 4.9: An origin graph and its encoding as a data word.

4.5.1 A logic for data words

Typed data words We consider *typed data words* over an ordered data domain where each datum carries a label (type) from a finite alphabet. Formally a typed data word of length n and *data size* m over alphabets A, B is a word over the alphabet $B \times \mathbb{N} \times A$: $w = (b_1, d_1, a_1) \cdots (b_n, d_n, a_n)$ such that $\{d_1, \ldots, d_n\} = \mathbf{m}$[3] and for each $i, j \in \mathbf{n}$, if $d_i = d_j$ then $a_i = a_j$. The letter a_i is called the *type* (sometimes label) of the data value d_i.

The data of a typed data word w induces a total pre-order over positions of w, defined by $i \preccurlyeq j$ if $d_i \leq d_j$. We also define $i \prec j$ by $d_i < d_j$ and $i \sim j$ if $d_i \leq d_j$ and $d_i \geq d_j$. Hence a typed data word can be seen as a logical structure with one linear order and one total pre-order.

Logics for data words We know from [BDM+11] that MSO over (untyped) data words (*i.e.* $|A| = 1$) is undecidable, even its first-order fragment. However, the fragment $\mathsf{FO}^2[\leq, \preccurlyeq, S_{\preccurlyeq}]$ *is* decidable, according to [SZ12]. The logic we define, called $\mathsf{L}d$, can be thought of as an extension of $\mathsf{FO}^2[\leq]$, with any binary predicates which only talk about the data, generalizing the result of [SZ12]. More precisely we define the binary predicates $\mathsf{MSO}[A, \preccurlyeq]$ with the semantic restriction that second-order quantification is restricted to sets closed under $\sim$[4]. Because of this restriction, the binary MSO predicates can be thought of as formulas that quantify over data and sets of data, see Rk. 4.5.2. For instance, the formula $\forall y\ x \preccurlyeq y$ states that the data value of x is the smallest of all data values.

Definition 4.5.1. We define the logic $\mathsf{L}d := \mathsf{FO}^2[B, \leq, \mathsf{MSO}[A, \preccurlyeq]]$

Remark 4.5.2. As we have said, $\mathsf{MSO}[A, \preccurlyeq]$ predicates can only talk about data, let us make this remark more explicit. Let $w = (b_1, d_1, a_1) \cdots (b_n, d_n, a_n)$ be a data word over A, B of data size m. The total preorder $\preccurlyeq$ induces a word over A, that is $u = a_{i_1} \cdots a_{i_m}$ where $d_{i_k} = k$, for any $k \in \mathbf{m}$. By definition of typed data words, we have that $d_i = d_j$ implies $a_i = a_j$, which means that u is well-defined. We call this word the *data quotient* of w. Then, the MSO predicates can talk about properties of the word u, for instance "Positions with even pieces of data are labeled by a" can be expressed in the logic.

4.5.2 From transductions to data words and back

Transforming models We describe encodings from non-erasing origin-graphs to typed data words and back. Let $g = (u, (v, o))$ be a non-erasing origin-graph over alphabets A, B, we

[3]This assumption is made without loss of generality since our logic will only be able to talk about the relative order of data values.

[4]This can actually be enforced syntactically by guarding quantifications $\exists X\ \phi$ by $\exists X\ (\forall x, y\ (x \in X \wedge x \sim y) \rightarrow y \in X) \wedge \phi$.

define $w = \mathrm{GtD}(g)$ a data word of length $|v|$ and data size $|u|$, by $w(i) = (v(i), o(i), u(o(i)))$ for $i \in \{1, \ldots, |v|\}$.

Let $w = (b_1, d_1, a_1) \cdots (b_n, d_n, a_n)$ be a data word of data size m over alphabets A, B. We define the non-erasing origin-graph $\mathrm{DtG}(w) = (u, (v, o))$ by $v = b_1 \cdots b_n$, u is the data quotient of w and $o(i) = d_i$ for $i \in \mathbf{n}$.

Proposition 4.5.3. *The transformations* GtD *and* DtG *are inverses of one another.*

Proof. Let $g = (u, (v, o))$ be a non-erasing origin-graph over alphabets A, B, let $m = |u|$, and $n = |v|$. Let $w = (b_1, d_1, a_1) \cdots (b_n, d_n, a_n)$ be a typed data word of data size m.

Let us assume that $\mathrm{GtD}(g) = w$. Then $w(i) = (v(i), o(i), u(o(i)))$, and let $g' = \mathrm{DtG}(w) = (u', (v', o'))$. By definition, $v' = v(1) \cdots v(|v|) = v$, $u' = u(1) \cdots u(|u|) = u$ and $o'(i) = o(i)$ for any $i \in \{1, \ldots, |u|\}$.

Conversely, let us assume that $\mathrm{DtG}(w) = g$. Then $v = b_1 \cdots b_n$, $u = a_{i_1} \cdots a_{i_m}$ where $d_{i_k} = k$ for $k \in \mathbf{m}$ and $o(i) = d_i$ for $i \in \mathbf{n}$. Let $w' = \mathrm{GtD}(g) = (b'_1, d'_1, a'_1) \cdots (b'_n, d'_n, a'_n)$, then for any $i \in \mathbf{n}$, $(b'_i, d'_i, a'_1) = (v(i), o(i), u(o(i))) = (b_i, d_i, a_i)$ and thus $w' = w$. $\square$

Transforming formulas We define a syntactic translation from $\mathsf{L}d$ to $\mathsf{L}o$ formulas that preserves models up to the encodings GtD and DtG. First, an $\mathsf{MSO}[A, \preccurlyeq]$ binary predicate χ is transformed syntactically into an $\mathsf{MSO}[A, \leq_{\mathsf{in}}]$ binary predicate χ' by substituting $\leq_{\mathsf{in}}$ for $\preccurlyeq$. Let ϕ be an $\mathsf{L}d$-formula over A, B. We define an output $\mathsf{L}o$-formula ϕ^o inductively, by substitutions of atomic formulas and by replacing quantifications by output quantifications. If:

- $\phi = \langle \chi \rangle(x, y)$ then $\phi^o := \langle \chi' \rangle(\mathsf{o}(x), \mathsf{o}(y))$
- $\phi = b(x)$ with $b \in B$ then $\phi^o := b(x)$
- $\phi = x \leq y$ then $\phi^o := x \leq_{\mathsf{out}} y$
- $\phi = \exists x\ \psi(x)$ then $\phi^o := \exists^{\mathsf{out}} x\ \psi^o(x)$.

The boolean connectives are left unchanged.

Lemma 4.5.4. *Let ϕ be an $\mathsf{L}d$-formula, and let w be a typed data word. Then $w \models \phi$ if and only if* $\mathrm{DtG}(w) \models \phi^o$.

Proof. We show this by induction over $\mathsf{L}d$-formulas. Let $w = (b_1, d_1, a_1) \cdots (b_n, d_n, a_n)$ be a data word of data size m over alphabets A, B and let $g = \mathrm{DtG}(w) = (u, (v, o))$. To simplify the induction, the domain of g is denoted by $\{1, \ldots, |v|\} \cup \{(1, 1), \ldots, (1, |u|)\}$. We show for ϕ an $\mathsf{L}d$-formula and for $i, j \in \mathbf{n}$ that if $w \models \phi(i, j)$ then $g \models \phi^o(i, j)$. The converse is done in the exact same way. Let $\phi = \langle \chi \rangle(x, y)$, and let $i, j \in \mathbf{n}$ such that $w \models \langle \chi \rangle(i, j)$. Then we have that $u \models \chi'(d_i, d_j)$ where u can be seen as the word over A induced by $\preccurlyeq$ over w. Thus we have $g \models \phi^o(i, j) = \langle \chi' \rangle(\mathsf{o}(i), \mathsf{o}(j))$. Let $\phi = b(x)$ and let $i \in \mathbf{n}$ such that $w \models b(i)$. Then clearly $v(i) = b$ and we have $g \models \phi^o(i) = b(i)$ since we produce an output formula and we only quantify over output positions. Let $\phi = x \leq y$ and let $i, j \in \mathbf{n}$ such that $w \models i \leq j$, then $i \leq j$ and we have naturally $g \models i \leq_{\mathsf{out}} j$. Let $\phi = \exists x\ \psi(x, y)$ and let $j \in \mathbf{n}$ be such that $w \models \exists x\ \psi(x, j)$. This means that there is a position $i \in \mathbf{n}$ such that $w \models \psi(i, j)$. By induction hypothesis we thus have that $g \models \psi^o(i, j)$, which concludes the proof. $\square$

Decidability As a corollary, we obtain the decidability of $\mathsf{L}d$, by reduction to $\mathsf{L}o$-formulas.

Theorem 4.5.5. *The satisfiability problem for* $\mathsf{L}d$ *over typed data words is decidable.*

Proof. Let ϕ be an $\mathsf{L}d$-formula, and let ϕ^o be the obtained output $\mathsf{L}o$-formula. Since DtG is a bijection between data words and non-erasing origin-graphs (Prop. 4.5.3), we have according to Lem. 4.5.4 that $\mathrm{DtG}(\llbracket\phi\rrbracket) = \llbracket\phi^o\rrbracket_o^{\mathrm{ne}}$. Thus, using Th. 4.4.21, satisfiability is decidable. □

4.5.3 Undecidable fragments

We recall two undecidable logics over data words, and show as a consequence the undecidability of two fragments of MSO_o over origin-graphs. The main idea is to use the tight correspondence between origin-graphs and data words, to show that the decidability of some fragment over origin-graphs would entail the decidability of a fragment over data words, known to be undecidable. The first one is the logic $\mathsf{FO}^2[\leq, S_\leq, \preccurlyeq]$ over data words with the linear order predicate, the successor predicate for the linear order and the data order predicate, which was shown to be undecidable in [BDM+11]. As a consequence we obtain the undecidability of $\mathsf{FO}^2[\leq_{\mathsf{out}}, S_{\mathsf{out}}, \leq_{\mathsf{in}}, \mathsf{o}]$ over origin-graphs.

Proof of Theorem 4.2.3. We consider the previous construction which transforms an $\mathsf{L}d$ formula over typed data words into an output $\mathsf{L}o$-formula over non-erasing origin-graphs. We add to the transformation by defining that if $\phi = S_\leq(x,y)$ then $\phi^o := S_{\mathsf{out}}(x,y)$. In this context, the proof of Lem. 4.5.4 still applies, which means that the decidability of $\mathsf{FO}^2[\leq_{\mathsf{out}}, S_{\mathsf{out}}, \leq_{\mathsf{in}}, \mathsf{o}]$ over origin-graphs would entail the decidability of $\mathsf{FO}^2[\leq, S_\leq, \preccurlyeq]$ yielding a contradiction. Hence the logic $\mathsf{FO}^2[\leq_{\mathsf{out}}, S_{\mathsf{out}}, \leq_{\mathsf{in}}, \mathsf{o}]$ is undecidable over origin-graphs. □

The second logic is $\mathsf{FO}^2[S_\leq, S_\preceq]$ over data words with the successor for the linear order and the successor for the data order which was shown to be undecidable in [MSZ13]. With the same technique we obtain the undecidability of $\mathsf{FO}^2[S_{\mathsf{out}}, S_{\mathsf{in}}, \mathsf{o}]$ over origin-graphs.

Proof of Theorem 4.2.4. We consider the same construction from $\mathsf{L}d$-formulas to $\mathsf{L}o$-formulas. We add again to the transformation that if $\phi = S_\leq(x,y)$ then $\phi^o := S_{\mathsf{out}}(x,y)$. We also add that if $\phi = S_\preccurlyeq(x,y)$ then $\phi^o := S_{\mathsf{in}}(\mathsf{o}(x), \mathsf{o}(y))$. Again, the proof of Lem. 4.5.4 means that the decidability of $\mathsf{FO}^2[S_{\mathsf{out}}, S_{\mathsf{in}}, \mathsf{o}]$ over origin-graphs would entail the decidability of $\mathsf{FO}^2[S_\leq, S_\preceq]$ yielding a contradiction. □

4.6 Decidable extensions of $\mathsf{L}o$

We present two different extensions of $\mathsf{L}o$ for which our uniformization techniques still work. This gives in particular the decidability of the two extensions and shows the robustness of $\mathsf{L}o$. Furthermore these extensions can be translated in terms of data word logics and so give more expressive decidable data word logics.

The first extension $\exists\mathsf{L}o$ is a classical extension for logics with arbitrary unary predicates. The logic $\exists\mathsf{L}o$ is the set of formulas of the form $\exists X_1, \dots, X_n \phi$ where ϕ is an $\mathsf{L}o$-formula with additional unary predicates $X_1, \dots, X_n$.

The second extension $\mathsf{L}o^{\mathrm{so}}$, adds new unary predicates called *single-origin predicates*. Given a rational language $L \subseteq B^*$, the semantics of a single-origin predicate $L(x)$ is that the word of positions with origin x belongs to L.

In Fig. 4.10 on the facing page we give a summary of the different transduction classes, their relative inclusions and the frontier for regular synthesis and decidability of satisfiability and equivalence.

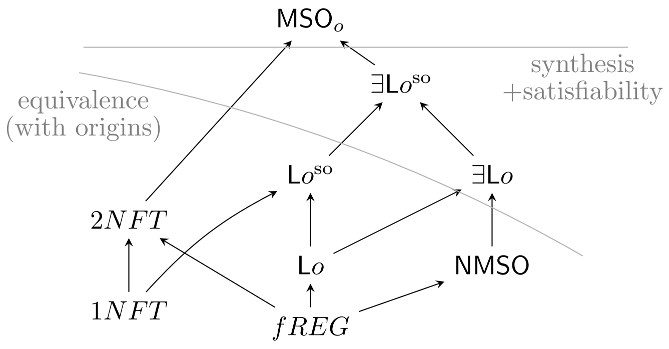


Figure 4.10: Summary of models for transductions and their relative inclusions. The gray lines represent the decidability frontiers.

4.6.1 Existential extension

The fragment ∃Lo allows us to capture NMSO-transducers (Th. 4.2.8) while preserving the decidability, and even the regular uniformization of the logic.

Theorem 4.6.1. *Given an* ∃Lo*-formula, one can synthesize a regular transduction uniformizing it.*

Proof. Let $\phi = \exists X_1 \dots X_n \psi$ be an ∃Lo-formula over alphabets A, B. Then the formula ψ can be seen as an Lo-formula over the alphabets $A \times \mathbf{2}^{\{X_1,\dots,X_n\}}, B \times \mathbf{2}^{\{X_1,\dots,X_n\}}$. Thus using Th. 4.4.25 we can obtain a two-way transducer $\mathcal{T}$ uniformizing the transduction of ψ over the extended alphabets. Let $\overline{\mathcal{T}}$ denote the transducer obtained from $\mathcal{T}$ by projecting the enriched alphabets to the original alphabets A, B. Thus we have that $[\![\overline{\mathcal{T}}]\!]_o \subseteq [\![\phi]\!]_o$ and $\text{dom}([\![\overline{\mathcal{T}}]\!]_o) = \text{dom}([\![\phi]\!]_o)$. However the transduction realized by $\overline{\mathcal{T}}$ may not be functional. This can be easily solved by a disambiguation procedure ([dS13]) which yields a transducer $\mathcal{T}'$ uniformizing $\overline{\mathcal{T}}$ and thus uniformizing ϕ as well. □

As a consequence we obtain a more expressive decidable logic over data words: ∃Ld defined by formulas of the form $\exists X_1, \dots, X_n \phi$ where ϕ is an Ld-formula with additional unary predicates $X_1, \dots, X_n$. The same translation from formulas over data words to formulas over origin-graphs works, just by considering extended alphabets.

Corollary 4.6.2. *The logic* ∃Ld *is decidable over typed data words.*

The previous results again give us the decidability of ∃Lo, but since the logic is not closed under boolean operations, we cannot conclude that validity and equivalence are also decidable. In fact it turns out that both problems are undecidable for ∃Lo.

Theorem 4.6.3. *The validity and equivalence problems for* ∃Lo*-formulas over origin-graphs are undecidable.*

Proof. We prove that the validity problem is undecidable. In particular, the equivalence with the formula $\top$ is thus undecidable. Since Lo is syntactically closed under negation, we actually show that the satisfiability problem is undecidable for the logic ∀Lo, which consists in formulas of the form $\forall X_1 \dots \forall X_n \phi$ where ϕ is an Lo-formula.

The proof works by reduction from the Post Correspondence Problem. Let A be an alphabet, let $n \in \mathbb{N}$ and let $u_i, v_i \in A^*$ for any $i \in \mathbf{n}$. We define a formula ϕ in ∀Lo such that ϕ is satisfiable if and only if there exists a sequence of indices $i_1, \dots, i_k \in \mathbf{n}$ such that $u_{i_1} \cdots u_{i_k} = v_{i_1} \cdots v_{i_k}$. The formula ϕ is defined over alphabets $A, A \times \mathbf{2}$, such that the projection of the output over $A \times \{j\}$ is the identity, for $j \in \mathbf{2}$. Furthermore, the output word must be of the form $(+_{i \in \mathbf{n}} 1(u_i)2(v_i))^*$, where the morphism $j : A^* \to (A \times \{j\})^*$ is defined by $j(a) = (a, j)$ for $a \in A$ and $j \in \mathbf{2}$.

We only have left to define the formula ϕ to conclude the proof. We define the formula $\phi := \forall^{\mathsf{out}} X \ \phi_{\mathrm{well-formed}}(X) \wedge \phi_{\mathrm{id},1} \wedge \phi_{\mathrm{id},2}$. The formulas $\phi_{\mathrm{id},j}$, for $j \in \mathbf{2}$, stating that over each copy of A the origin-graph belongs to the identity, are easily defined as in Ex. 4.2.6:

- $j(x) := \bigvee_{a \in A} (a, j)(x)$
- $\phi_{\mathrm{inj},j} := \forall^{\mathsf{out}} x, y \ \ (j(x) \wedge j(y) \wedge \langle \mathsf{o}(x) =_{\mathsf{in}} \mathsf{o}(y) \rangle) \to x = y$
- $\phi_{\mathrm{surj},j} := \forall^{\mathsf{in}} x \ \exists^{\mathsf{out}} y \ \langle \mathsf{o}(y) =_{\mathsf{in}} x \rangle \wedge j(y)$
- $\phi_{\mathrm{lab},j} := \forall^{\mathsf{out}} x \ \ \bigwedge_{a \in A} (a, j)(x) \to \langle a(\mathsf{o}(x)) \rangle$
- $\phi_{\mathrm{shuffle},j} := \phi_{\mathrm{inj},j} \wedge \phi_{\mathrm{surj},j} \wedge \phi_{\mathrm{lab},j}$
- $\phi_{\mathrm{id},j} := \phi_{\mathrm{shuffle},j} \wedge \forall^{\mathsf{out}} x, y \ \ (j(x) \wedge j(y) \wedge \langle \mathsf{o}(x) \leq_{\mathsf{in}} \mathsf{o}(y) \rangle) \to x \leq_{\mathsf{out}} y$

We now have left to define $\phi_{\mathrm{well-formed}}(X)$ such that $\forall^{\mathsf{out}} X \ \phi_{\mathrm{well-formed}}(X)$ will ensure that the output is indeed in the language $L := (+_{i \in \mathbf{n}} 1(u_i)2(v_i))^*$. We first define a formula stating that X is a contiguous set of positions:

$$\mathrm{cont(X)} := \neg \left(\exists^{\mathsf{out}} x \ x \in X \wedge (\exists^{\mathsf{out}} y \ x <_{\mathsf{out}} y \wedge y \notin X \wedge (\exists^{\mathsf{out}} x \ \wedge y <_{\mathsf{out}} x \wedge x \in X)) \right)$$

Let $m := \max \{ |u_i| + |v_i| \mid \ i \in \mathbf{n} \} + 1$, then the words in L can be characterized by their factors, prefix and suffix of length $\leq m$. Let $w \in (A \times \mathbf{2})^*$ we define a predicate $w(X, x)$ which states that w is a subword of X starting at position x. The predicates are defined by induction on the length of w. Let $a \in A \times \mathbf{2}$, then $a(X, x) := x \in X \wedge a(x)$, $aw(X, x) := a(X, x) \wedge \exists^{\mathsf{out}} y \ x <_{\mathsf{out}} y \wedge w(X, y)$. With the same technique we can define $|X| = p$ for any integer $p \in \mathbb{N}$. Let $F \subseteq (A \times \mathbf{2})$ be the set of factors of length m of L. Similarly, let P and S be the set of prefixes and suffixes of length m in L, respectively. Let $\min(X)$ and $\max(X)$ denote that the minimum position and the maximum position of the word belongs to X, respectively.

$$\begin{array}{rcl} \phi_{\text{well-formed}}(X) & := & \mathrm{cont}(X) \wedge |X| = m \\ & \to & \bigvee_{w \in F} w(X) \\ & & \wedge \bigvee_{w \in P} \min(X) \to w(X) \\ & & \wedge \bigvee_{w \in S} \max(X) \to w(X) \end{array}$$

Note that the formula does not consider the small PCP solutions, *i.e.* of the form $u_i = v_i$ for instance. Of course this is not a problem because if there exists a solution then there is a solution of any arbitrary size. □

4.6.2 Single-origin predicates

One problem with $\mathsf{L}o$ is the impossibility to express arbitrary rational properties of the output independently of the input. Of course as we have seen, just adding this ability without any restriction would in fact cause undecidability of the logic (see the undecidability proof from [BDM+11]). We find a middle ground and add predicates that talk about rational properties of the output word, but restricted to positions with a single origin. For instance if the input word contains only one position then $\mathsf{L}o$ can only talk about $\mathsf{FO}^2[\leq_{\mathsf{out}}]$-definable properties. However in that case $\mathsf{L}o^{\mathrm{so}}$ can define any rational property of the output. As we have seen, the class of rational transductions (*i.e.* realized by 1NFTs) is incomparable with $\mathsf{L}o$-transductions. However these new single-origin predicates allow us to subsume the rational transductions

In terms of data words, the extension allows one to talk about any rational property of a subword induced by a single piece of data. In particular, over data words with data size 1, we recover the rational languages instead of $\mathsf{FO}^2[\leq]$-definable languages.

Let A, B be alphabets, let L be a rational language over B^* we define a new unary input predicate $L(x)$ with the semantic interpretation that the subword of positions with origin x belongs to L. We define over alphabets A, B the logic with single-origin predicates, $\mathsf{L}o^{\mathrm{so}} := \mathsf{FO}^2[B, \leq_{\mathsf{out}}, \mathsf{o}, \mathsf{MSO}[A \uplus \{L(x) \mid L \text{ rational}\}, \leq_{\mathsf{in}}]]$

Example 4.6.4. Let A, B be alphabets. Let $L = (ab)^*$, we define the $\mathsf{L}o^{\mathrm{so}}$-formula $\phi := \forall^{\mathsf{out}} x\ a(x) \to \langle \mathsf{even}(\mathsf{o}(x)) \wedge L(\mathsf{o}(x))\rangle$ which states that an output position labeled by a must have an even origin and that the output word of positions with the same origin must belong to L.

We show that any rational transduction can be expressed by an $\mathsf{L}o^{\mathrm{so}}$-formula. This is not the case for $\mathsf{L}o$: for instance the transduction $\{a\} \times (aa)^*$ cannot be expressed in the logic since the input is trivial and the output is not $\mathsf{FO}^2[\leq_{\mathsf{out}}]$-definable. Another example of such a transduction is given in Ex. 4.2.9.

Theorem 4.6.5. *Any rational transduction is $\mathsf{L}o^{\mathrm{so}}$-definable.*

Proof. Let $\mathcal{T}$ be a one-way transducer, our goal is to define an equivalent $\mathsf{L}o^{\mathrm{so}}$-formula. We see $\mathcal{T}$ as an *automaton* over the alphabet $A \uplus B$, which we can assume to be deterministic, with state space Q, initial state $q_0 \in Q$ and set of final states $F \subseteq Q$. Given a letter $a \in A \uplus B$ and a state $q \in Q$, we denote by $q.a$ the state reached from q after reading a. In order to simplify the proof and without loss of generality, we assume that we can only read letters of A from q_0. We define the origin semantics of $\mathcal{T}$: Let $a_1 v_1 \dots a_n v_n$ be a word accepted by $\mathcal{T}$, such that for $i \in \mathbf{n}$, $a_i \in A$ and $v_i \in B^*$. We define the origin graph $g := (u, (v, o))$ with $u := a_1 \cdots a_n$, $v := v_1 \cdots v_n$ and for $j \in \{1, \dots, |v|\}$ such that $|v_1 \cdots v_{j-1}| < i \leq |v_1 \cdots v_j|$ we set $o(j) := i$. Then we say that $\mathcal{T}$ realizes g which we denote by $g \in [\![\mathcal{T}]\!]_o$. Let $q, r \in Q$ and let $L_{q,r} \subseteq B^*$ be the set of output words that can go from q to r in $\mathcal{T}$ (without reading any input letter).

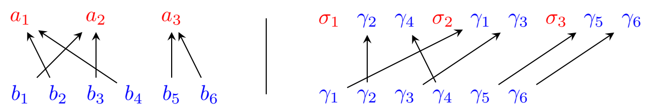
We define the $\mathsf{L}o^{\mathrm{so}}$-formula $\phi_{\mathrm{pres}} \wedge \langle \exists_{p,q,r \in Q} X^p_{q,r} \phi\rangle$ with $\phi_{\mathrm{pres}} := \forall^{\mathsf{out}} x, y\ \mathsf{o}(x) \leq_{\mathsf{in}} \mathsf{o}(y) \to x \leq_{\mathsf{out}} y$ states that the input order is preserved in the output. The formula ϕ is a conjunction of four formulas: $\phi_{var} \wedge \phi_{succ} \wedge \phi_{min} \wedge \phi_{max}$. The formula ϕ_{var} will encode that the input positions in $X^p_{q,r}$ can go from p to q by reading the letter and produce output words in $L_{q,r}$. The formula ϕ_{succ} ensures that two successive positions must belong to some sets $X^p_{q,r}$ and $X^r_{s,t}$, respectively. The formulas ϕ_{min} and ϕ_{max} encode that the first position starts in the initial state and that the last one ends up in a final state, respectively.

$$
\begin{aligned}
\phi_{var} &= \forall^{\mathsf{in}} x \textstyle\bigwedge_{p,q,r \in Q} X^p_{q,r}(x) \to \big(L_{q,r}(x) \bigwedge_{a \in A} a(x) \to p.a = q\big) \\
\phi_{succ} &= \forall^{\mathsf{in}} x, y \textstyle\bigwedge_{p,q,r \in Q} \Big(S_{\mathsf{in}}(x, y) \wedge X^p_{q,r}(x) \to \bigvee_{s,t} X^r_{s,t}(x)\Big) \\
\phi_{min} &= \forall^{\mathsf{in}} x\ \ \min(x) \to \textstyle\bigvee_{p,q \in Q} X^{q_0}_{p,q}(x) \\
\phi_{max} &= \forall^{\mathsf{in}} x\ \ \max(x) \to \textstyle\bigvee_{p,q \in Q, r \in F} X^p_{q,r}(x)
\end{aligned}
$$

By construction we have obtained an $\mathsf{L}o^{\mathrm{so}}$-formula realizing $[\![\mathcal{T}]\!]_o$. □

Our regular uniformization algorithm can be extended to the logic $\mathsf{L}o^{\mathrm{so}}$.

Theorem 4.6.6. *Given an $\mathsf{L}o^{\mathrm{so}}$-formula, one can synthesize a regular transduction uniformizing it.*



a_1 a_2 a_3
b_1 b_2 b_3 b_4 b_5 b_6
σ_1 γ_2 γ_4 σ_2 γ_1 γ_3 σ_3 γ_5 γ_6
γ_1 γ_2 γ_3 γ_4 γ_5 γ_6

Figure 4.11: An origin-graph of ϕ and its translated version as an origin-graph of τ_2.

Proof. Let ϕ be an $\mathsf{L}o^{\text{so}}$-formula over alphabets A, B. We view $[\![\phi]\!]_o$ as the composition of two transductions, $\tau_2 \circ \tau_1$ where τ_1 is a rational transduction (*i.e.* given by a 1NFT) and τ_2 is an $\mathsf{L}o$-transduction. From Th. 4.4.25, we can obtain a regular function f_2 uniformizing τ_2. Before defining τ_1 and τ_2 let us show how one can use such a decomposition to prove the statement. We define τ_1', a restriction of the codomain of τ_1 to $\text{dom}(f_2)$. Formally, $\tau_1' := \{(u,(v,o)) \in \tau_1 \mid v \in \text{dom}(f_2)\}$. Since f_2 is regular, its domain is a rational language and we can define a 1NFT realizing τ_1'. Then we uniformize τ_1', which can be done by an unambiguous 1NFT ([EM65]), and obtain a rational function f_1, such that $f_2 \circ f_1$ uniformizes $[\![\phi]\!]_o$.

The transduction τ_1 is a rational transduction over $A, A \uplus B$ which after reading a letter $a \in A$ produces an arbitrary word in aB^*. The idea is that each letter produces its subword so that τ_2 can talk about the input instead of the output. Let $g = (u,(v,o))$ be an origin-graph over A, B, we define $g' = (u',(v,o'))$ over $A \uplus B, B$: let $V_i := \{j \in \text{dom}(v) \mid o(j) = i\}$, for $i \in \text{dom}(u)$. Let $v_i = \prod_{j \in V_i} v(j)$, then $u' := u(1)v_1 \cdots u(|u|)v_{|u|}$. Finally we set $o'(j) := i + j$ for $j \in V_i$. We illustrate this in Fig. 4.11.

We now transform ϕ syntactically into an $\mathsf{L}o$-formula ϕ' over alphabets $A \uplus B, B$. Our goal is then to show that $[\![\phi']\!]_o := \{g' \mid g \models \phi\}$. We define a binary input predicate that relates an input position y labeled in B to the previous input position x labeled in A, *i.e.* its "origin" with respect to τ_1 and which we call the *virtual origin*:

$$\text{vo}(x,y) := A(x) \wedge B(y) \wedge x <_{\mathsf{in}} y \wedge \forall^{\mathsf{in}} z \ \ (x <_{\mathsf{in}} z <_{\mathsf{in}} y) \to B(z)$$

Let L be a regular language over B, we denote by ϕ_L an $\mathsf{MSO}[\leq]$-formula recognizing it. Our syntactic transformations only modifies the input predicates and is done in three steps.

1) We guard all quantifications so that they only talk about input positions labeled in A: $(\exists^{\mathsf{in}} x \ \psi(x))' := \exists^{\mathsf{in}} x \ A(x) \wedge \psi'(x)$.

2) The binary predicates $\langle \phi(x,y) \rangle$ are replaced by $\langle \exists^{\mathsf{in}} z, t \ \text{vo}(z,x) \wedge \text{vo}(t,y) \wedge \phi'(z,t) \rangle$. One can see that $g \models \langle \phi(x,y) \rangle(\mathsf{o}(i), \mathsf{o}(j))$ if and only if $g' \models \langle \exists^{\mathsf{in}} z, t \ \text{vo}(z,x) \wedge \text{vo}(t,y) \wedge \phi'(z,t) \rangle(\mathsf{o}(i), \mathsf{o}(j))$.

3) Finally a single origin predicate is replaced by an input predicate which talk about the input positions with a fixed virtual origin. A predicate $L(x)$ is replaced by a formula $\phi_L^{\text{vo}(x,\_)}$ defined from ϕ_L by restricting quantifications to positions with virtual origin x.

The final formula is $\phi' \wedge \phi_{\text{well-formed}}$ where $\phi_{\text{well-formed}}$ is the conjunction of:

- input positions labeled in A do not produce anything: $\forall^{\mathsf{in}} x \ \langle A(x) \rangle \to \forall^{\mathsf{out}} y \ x \neq \mathsf{o}(y)$
- input positions labeled in B produce exactly one output position with the same label: $\forall^{\mathsf{in}} x \ \left(\bigwedge_{b \in B} b(x) \to \exists^{\mathsf{out}} y \ b(y) \wedge \mathsf{o}(y) = x \right) \wedge (\forall^{\mathsf{out}} x, y \ \mathsf{o}(x) = \mathsf{o}(y) \to x = y)$
- over a fixed virtual origin, the linear order is preserved which translates into the $\mathsf{L}o$-formula: $\forall^{\mathsf{out}} x, y \ \langle \exists^{\mathsf{in}} z \ \text{vo}(z, \mathsf{o}(x)) \wedge \text{vo}(z, \mathsf{o}(y) \wedge x <_{\mathsf{in}} y \rangle \to x <_{\mathsf{out}} y$.

By construction $\tau_2 := [\![\phi' \wedge \phi_{\text{well-formed}}]\!]_o = \{g' \mid g \models \phi\}$. Since we have indeed that $\tau_2 \circ \tau_1 = [\![\phi]\!]_o$, then this concludes the proof. $\square$

Remark 4.6.7. The same proof would actually work for $\exists\mathsf{L}o^{\mathrm{so}}$. Moreover, let $\mathcal{C}_1, \mathcal{C}_2$ be classes of transducers such that regular uniformization is possible for both classes and $\mathcal{C}_1$ is closed under rational restriction of the codomain. Then one can uniformize any transduction of $\tau_2 \circ \tau_1$ with τ_i given by a $\mathcal{C}_i$ transducer, for $i \in \mathbf{2}$. For instance τ_1 can be given by a two-way non-deterministic transducer and τ_2 by an $\mathsf{L}o$-formula.

As a consequence we obtain that the satisfiability problem for $\mathsf{L}o^{\mathrm{so}}$ is decidable and, since $\mathsf{L}o^{\mathrm{so}}$ is closed under boolean operations, the validity and equivalence problems are also decidable.

Theorem 4.6.8. *The satisfiability, validity and equivalence problems for* $\mathsf{L}o^{\mathrm{so}}$ *over origin-graphs are decidable.*

Proof. This is a direct consequence of Th. 4.6.5. Since one can uniformize an $\mathsf{L}o^{\mathrm{so}}$-transduction, in particular one can decide the emptiness of the domain. Furthermore, by closure under boolean combinations, we obtain the decidability of the validity and equivalence problems. □

Conclusion

In this manuscript we have studied definability and synthesis problems for word-to word transductions. In the first part, we tackled definability questions for classes of rational word functions using algebraic and computational minimization techniques. In the second part, we introduced an expressive logic for transductions from words to words and solved the regular uniformization problem for this logic.

Rational functions and canonical models

Canonical models have proven to be very useful to study computational objects. In the case of automata over finite words, having a canonical object is interesting for several reasons. The first one is of course that equivalent automata correspond to the same minimal automaton. The second reason is the size, *i.e.* the *minimal* automaton of a language has the least number of states among deterministic automata recognizing the same language, hence the name. Another property of the minimal automaton is that it can be obtained efficiently (in PTime) from any deterministic automaton. Finally, the minimal automaton carries intrinsic algebraic properties of the language, which has proven particularly useful in definability problems, and in particular thanks to the many logic-algebra correspondences which have been established over the years (see [Str94, DGK08] *e.g.*).

As it has been shown (see [Cho03]), almost all these good properties of the minimal automaton carry over to the minimal sequential transducer for *sequential* functions. A first step had been made in that direction with [RS91], but rational functions still lacked a good description of minimal devices with the aforementioned good properties. In this manuscript we have shown that rational functions have not one but a finite number of minimal bimachines. The difficulty comes from the fact the we need to minimize two things at once: the look-ahead and the automaton. In particular any bimachine minimal in the number of states is minimal in the algebraic sense that its transition congruence is minimal. Furthermore, as in the language case, minimization of bimachines can be done in PTime. We also have established a transfer theorem from logic-algebra correspondences over languages to logic-algebra correspondences over functions. In particular this yields an algorithm to decide definability of a rational function in first-order logic. In fact we show that FO-definability is PSpace-complete for rational functions given as bimachines.

Open problems Some problems still remain open. We know (Ex. 2.2.36) that a minimal bimachine can be exponentially larger than another bimachine realizing the same function. We also know (Th. 2.2.35) that a minimal bimachine has size at most doubly exponential with respect to any equivalent bimachine. The open question is what is the exact worst-case size of a minimal bimachine? Another question is the complexity of deciding aperiodicity when the function is given by a transducer instead of a bimachine. We know that the problem is PSpace-hard and in ExpTime, but whether it can be done in PSpace remains open. Another direction of study, already mentioned in [RS91], would be to characterize the functions which are both sequential and right sequential. This would, we believe, help us to better understand the relationships between the different minimal bimachines of a function.

Over infinite words we have more open questions. While the case of sequential functions has been treated satisfactorily, it remains open whether one can minimize right-sequential functions. Such a procedure would probably solve most of the remaining problems and yield a complete description of minimal bimachines, as in the finite case. Another question is whether the ultimate congruence has some minimality property, like the delay congruence. Answering these questions would help us characterize more logical fragments, as in the case of finite words. Finally, the complexity of computing a canonical bimachine is probably much lower than the one of our current algorithm.

Future work Our approach has provided effective characterizations for classes of logical transductions. However the cases we cover are restricted to logical fragments with access to the linear order predicate. One extension that would require new techniques would be to characterize the transductions definable in $\mathsf{FO}[+1]$.

Over infinite words, it seems that continuity plays an important role, and we should be able, for continuous rational functions, to obtain a unique minimal look-ahead congruence.

Obtaining a canonical device for regular functions seems to be a very difficult problem. However restricting to *sweeping* transductions, *i.e.* functions realized by two-way transducers that can only can direction at the ends of a word, it seems like some of the ideas of [RS91] could be applied. A step in that direction has actually been made in [LLN$^+$11]. In this article the authors give a normal from for sequential transducers from *trees* to words. In particular, over unary trees, their model correspond to a deterministic transducer with one forward pass and one backward pass. It looks like this normal form should preserve aperiodicity. This would give good hopes of defining a normal form for aperiodic transductions definable by two-way transducers with two sweeps, and thus deciding $\mathsf{FO}[\leq]$-definability for these functions.

Another possible extension, using the same kind of normal form for tree-to-word transducers [LLN$^+$11, Boi16], would be to decide if a certain transducer is definable in some logical fragment of MSO. Indeed, over tree languages several effective characterizations of logical fragments have been obtain *via* an algebraic characterization (see [Boj08]).

Specification and synthesis of transductions

A computational model that takes inputs, performs some computation and then produces outputs defines a transduction. In that framework natural verification questions arise. The model-checking problem asks whether the model satisfies a given specification, which can itself be seen as a (not necessarily functional) transduction of all acceptable acceptable input/output pairs, and the problem amounts to deciding inclusion. The synthesis problem asks, given a specification, whether one can produce a model in a given class which satisfies the specification. In both cases a good specification language would be somewhat close to the natural language. Indeed, one would want to express high-level properties of a model and check them or synthesize a model from it. Good examples in the case of regular languages would be MSO, regular expressions, or LTL which has some good algorithmic properties.

In the case of transductions, no such specification language existed. Of course the Courcelle NMSO-transducer do express logical properties but they are very limited in what they can express. One of the big limitations of NMSO-transductions is that they cannot *underspecify* properties, meaning that once an interpretation has been chosen for the parameters, the transduction becomes functional. For instance the transduction $A^+ \times B^*$ cannot be expressed as an

NMSO-transducer. However any origin-graph satisfies the formula $\top$ over origin-graphs which realizes the (origin-free) transduction $A^+ \times B^*$.

We have introduced a new logic Lo for transductions, which we believe is well-suited as a specification language for transductions. We have been able to provide a regular uniformization algorithm for Lo-transductions, and as a consequence have obtained many results, such as decidability of satisfiability, equivalence, functionality, *etc.* We have also shown that the logic can be extended by additional predicates while preserving the uniformization capabilities.

Open problems We have been able to define decidable extensions of Lo that subsume the classes of transductions definable by 1NFTs and by NMSO-transducers. However we have not been able to come up with an extension that subsumes the transductions definable by 2NFTs, but still has decidable satisfiability. This problem looks challenging, in particular the transduction $R = \{(u, u^n) \mid u \in A^*, n \in \mathbb{N}\}$ seems difficult to express with only two variables.

In [BDGP17] the authors managed to characterize the origin-transductions definable by NMSO-transducers. This characterization relies on a property they call *bounded crossing*. Given the profile automaton of an Lo-transduction, one may be able to decide if the correspond transduction has bounded crossing which would give an algorithm to decide if an Lo-transduction can be realized by an NMSO-transducer.

Future work Other extensions of the logic Lo should be possible but the link with data words shows how close we are to the undecidability frontier (see [MZ13]), and one should tread lightly.

This link between data words and origin-graphs, although surprising at first, has allowed us to guide our search for an expressive and still decidable logic. While most models of computation of data words allow full power on the word without data and restrict heavily the possibility to compare data, our approach is quite different. We restrict the expressiveness on the word but allow one to talk about any property that only involves the data. We think this link should prove fruitful in the study of both data words and transductions.

We have high expectations that our method can be transfered to tree-to-word transductions using tree-automata that recognize profile trees instead of profile sequences. The generalization to tree-to-tree transductions (or even word-to-tree) transductions seems difficult. Indeed if one wants to be able to express MSO-transductions, one quickly obtains transductions where the domain is non-regular which would require a completely different approach. However, using the Courcelle/Bojańczyk-Pilipczuk Theorem ([BP16]) we should obtain a regular synthesis-like result (we would obtain a not necessarily functional NMSO-transduction in the end) for transductions from graphs of bounded tree-width to words. Finally, in terms of data words, a tree-to-word origin-graph can be seen as a data word where the data has more structure than a linear order, *i.e.* a tree-structure. In [Tan14] the author mentions such a model, where data values are strings ordered by the prefix order which gives a ranked tree structure to the data.

In our approach we manage to solve the problem of equivalence of transductions *with* origins. Of course equivalence of transductions without origins is undecidable even for rational transductions. It may be interesting to investigate a notion of equivalence of transductions up to “small” origin changes, similar to what was done in the case of rational transductions by [FJLW16].

Bibliography

[AC10] Rajeev Alur and Pavol Cerný, *Expressiveness of streaming string transducers*, IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India, 2010, pp. 1–12. [Cited on pages vi, 2, 91, 95, and 96.]

[AD11] Rajeev Alur and Jyotirmoy V. Deshmukh, *Nondeterministic streaming string transducers*, Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part II, 2011, pp. 1–20. [Cited on pages 95 and 96.]

[ADR15] Rajeev Alur, Loris D'Antoni, and Mukund Raghothaman, *Drex: A declarative language for efficiently evaluating regular string transformations*, Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015, 2015, pp. 125–137. [Cited on page 91.]

[AFR14] Rajeev Alur, Adam Freilich, and Mukund Raghothaman, *Regular combinators for string transformations*, Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014, 2014, pp. 9:1–9:10. [Cited on page 91.]

[Arn85] André Arnold, *A syntactic congruence for rational omega-language*, Theor. Comput. Sci. **39** (1985), 333–335. [Cited on pages 15 and 16.]

[BC00] Marie-Pierre Béal and Olivier Carton, *Computing the prefix of an automaton*, ITA **34** (2000), no. 6, 503–514. [Cited on pages 41 and 63.]

[BC02] ———, *Determinization of transducers over finite and infinite words*, Theor. Comput. Sci. **289** (2002), no. 1, 225–251. [Cited on pages 28, 33, 34, and 46.]

[BC04] ———, *Determinization of transducers over infinite words: The general case*, Theory Comput. Syst. **37** (2004), no. 4, 483–502. [Cited on pages 23, 58, 62, 65, 66, 67, 77, and 78.]

[BCPS00] Marie-Pierre Béal, Olivier Carton, Christophe Prieur, and Jacques Sakarovitch, *Squaring transducers: An efficient procedure for deciding functionality and sequentiality of transducers*, LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings, 2000, pp. 397–406. [Cited on page 23.]

[BDGP17] Mikołaj Bojańczyk, Laure Daviaud, Bruno Guillon, and Vincent Penelle, *Which classes of origin graphs are generated by transducers*, 44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland, 2017, pp. 114:1–114:13. [Cited on pages xi, 6, 92, 96, and 127.]

[BDM$^+$11] Mikołaj Bojańczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin, *Two-variable logic on data words*, ACM Trans. Comput. Log. **12** (2011), no. 4, 27:1–27:26. [Cited on pages 92, 96, 115, 116, 118, and 120.]

[Ber79] Jean Berstel, *Transductions and context-free languages*, Teubner Studienbücher : Informatik, vol. 38, Teubner, 1979. [Cited on pages v, vi, 1, 2, and 23.]

[BGMP16] Félix Baschenis, Olivier Gauwin, Anca Muscholl, and Gabriele Puppis, *Minimizing resources of sweeping and streaming string transducers*, 43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy, 2016, pp. 114:1–114:14. [Cited on pages vii and 3.]

[BGMP17] ———, *Untwisting two-way transducers in elementary time*, 32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017, 2017, pp. 1–12. [Cited on pages vii and 3.]

[BL69] Julius Richard Büchi and Lawrence H. Landweber, *Solving sequential conditions by finite-state strategies*, Transactions of the American Mathematical Society **138** (1969), 295–311. [Cited on pages viii, 4, and 92.]

[BLN12] Adrien Boiret, Aurélien Lemay, and Joachim Niehren, *Learning rational functions*, Developments in Language Theory - 16th International Conference, DLT 2012, Taipei, Taiwan, August 14-17, 2012. Proceedings, 2012, pp. 273–283. [Cited on pages 57 and 79.]

[Boi16] Adrien Boiret, *Normal form on linear tree-to-word transducers*, Language and Automata Theory and Applications - 10th International Conference, LATA 2016, Prague, Czech Republic, March 14-18, 2016, Proceedings, 2016, pp. 439–451. [Cited on page 126.]

[Boj08] Mikołaj Bojańczyk, *Effective characterizations of tree logics*, Proceedings of the Twenty-Seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2008, June 9-11, 2008, Vancouver, BC, Canada, 2008, pp. 53–66. [Cited on page 126.]

[Boj14] ———, *Transducers with origin information*, Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II, 2014, pp. 26–37. [Cited on pages ix, x, 4, 6, 91, and 95.]

[Boj15] ———, *Recognisable languages over monads*, Developments in Language Theory - 19th International Conference, DLT 2015, Liverpool, UK, July 27-30, 2015, Proceedings., 2015, pp. 1–13. [Cited on page 25.]

[Bou02] Patricia Bouyer, *A logical characterization of data languages*, Inf. Process. Lett. **84** (2002), no. 2, 75–85. [Cited on page 115.]

[BP16] Mikołaj Bojańczyk and Michal Pilipczuk, *Definability equals recognizability for graphs of bounded treewidth*, Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016, 2016, pp. 407–416. [Cited on page 127.]

[BR18] Nicolas Baudru and Pierre-Alain Reynier, *From two-way transducers to regular function expressions*, Developments in Language Theory - 22nd International Conference, DLT 2018, Tokyo, Japan, September 10-14, 2018, Proceedings, 2018. [Cited on page 91.]

[Bü60] Julius Richard Büchi, *Weak second-order arithmetic and finite automata*, Mathematical Logic Quarterly - MLQ **6** (1960), 66–92. [Cited on page 21.]

[Bü62] ———, *On a decision method in restricted second order arithmetic*, Proc. International Congress on Logic, Method, and Philosophy of Science, Stanford University Press, 1962, pp. 1–12. [Cited on page 21.]

[Car10] Olivier Carton, *Right-sequential functions on infinite words*, Computer Science - Theory and Applications, 5th International Computer Science Symposium in Russia, CSR 2010, Kazan, Russia, June 16-20, 2010. Proceedings, 2010, pp. 96–106. [Cited on pages 23, 24, 57, 58, and 76.]

[CCP17] Michaël Cadilhac, Olivier Carton, and Charles Paperman, *Continuity and rational functions*, 44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland, 2017, pp. 115:1–115:14. [Cited on pages viii and 4.]

[CD15] Olivier Carton and Luc Dartois, *Aperiodic two-way transducers and fo-transductions*, 24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany, 2015, pp. 160–174. [Cited on pages viii and 4.]

[CH91] Sang Cho and Dung T. Huynh, *Finite-automaton aperiodicity is PSpace-complete*, Theor. Comput. Sci. **88** (1991), no. 1, 99–116. [Cited on pages 19 and 51.]

[Cho77] Christian Choffrut, *Une caracterisation des fonctions sequentielles et des fonctions sous-sequentielles en tant que relations rationnelles*, Theor. Comput. Sci. **5** (1977), no. 3, 325–337. [Cited on pages 23 and 49.]

[Cho03] ———, *Minimizing subsequential transducers: a survey*, Theor. Comput. Sci. **292** (2003), no. 1, 131–143. [Cited on pages vii, 3, 22, 27, 28, 31, 32, 58, and 125.]

[Chu63] Alonzo Church, *Application of recursive arithmetic to the problem of circuit synthesis*, Journal of Symbolic Logic **28** (1963), no. 4, 289–290. [Cited on pages viii, 4, and 92.]

[CKLP15] Michaël Cadilhac, Andreas Krebs, Michael Ludwig, and Charles Paperman, *A circuit complexity approach to transductions*, Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part I, 2015, pp. 141–153. [Cited on pages viii and 4.]

[CL15] Arnaud Carayol and Christof Löding, *Uniformization in automata theory*, Logic, Methodology and Philosophy of Science - Proceedings of the 14th International Congress, Nancy, 2011, College Publications, 2015. [Cited on pages viii, 4, and 92.]

[CM03] Olivier Carton and Max Michel, *Unambiguous Büchi automata*, Theor. Comput. Sci. **297** (2003), no. 1-3, 37–81. [Cited on pages 14, 19, 23, 57, 85, and 86.]

[Cou90] Bruno Courcelle, *The monadic second-order logic of graphs. i. recognizable sets of finite graphs*, Inf. Comput. **85** (1990), no. 1, 12–75. [Cited on pages xi, 6, 92, 96, and 115.]

[Cou94] ———, *Monadic second-order definable graph transductions: A survey*, Theor. Comput. Sci. **126** (1994), no. 1, 53–75. [Cited on pages 24, 94, 101, and 102.]

[Cou96] ———, *The monadic second-order logic of graphs X: linear orderings*, Theor. Comput. Sci. **160** (1996), no. 1&2, 87–143. [Cited on page 115.]

[CP17] Thomas Colcombet and Daniela Petrisan, *Automata minimization: a functorial approach*, 7th Conference on Algebra and Coalgebra in Computer Science, CALCO 2017, June 12-16, 2017, Ljubljana, Slovenia, 2017, pp. 8:1–8:16. [Cited on pages vii and 3.]

[CPP08] Olivier Carton, Dominique Perrin, and Jean-Éric Pin, *Automata and semigroups recognizing infinite words*, Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]., 2008, pp. 133–168. [Cited on pages 14, 15, and 17.]

[CPS06] Laura Chaubard, Jean-Eric Pin, and Howard Straubing, *Actions, wreath products of* C*-varieties and concatenation product*, Theor. Comput. Sci. **356** (2006), no. 1-2, 73–89. [Cited on pages vii, 3, and 15.]

[DFJL17] Luc Dartois, Paulin Fournier, Ismaël Jecker, and Nathan Lhote, *On reversible transducers*, 44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland, 2017, pp. 113:1–113:12. [Cited on page 91.]

[DFL18] Luc Dartois, Emmanuel Filiot, and Nathan Lhote, *Logics for word transductions with synthesis*, Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018, 2018, pp. 295–304. [Cited on page 93.]

[DG08] Volker Diekert and Paul Gastin, *First-order definable languages*, Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]., 2008, pp. 261–306. [Cited on pages 18, 19, and 51.]

[DGK08] Volker Diekert, Paul Gastin, and Manfred Kufleitner, *A survey on small fragments of first-order logic over finite words*, Int. J. Found. Comput. Sci. **19** (2008), no. 3, 513–548. [Cited on pages 17, 21, 56, and 125.]

[DGK18] Vrunda Dave, Paul Gastin, and Shankara Narayanan Krishna, *Regular transducer expressions for regular transformations over infinite words*, Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '18, Oxford, United Kingdom, July 9-12, 2018, 2018. [Cited on page 91.]

[DJRV17] Laure Daviaud, Ismaël Jecker, Pierre-Alain Reynier, and Didier Villevalois, *Degree of sequentiality of weighted automata*, Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, 2017, pp. 215–230. [Cited on pages vii and 3.]

[DK09] Volker Diekert and Manfred Kufleitner, *Fragments of first-order logic over infinite words*, 26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, February 26-28, 2009, Freiburg, Germany, Proceedings, 2009, pp. 325–336. [Cited on pages vii, 3, and 21.]

[DKT16] Vrunda Dave, Shankara Narayanan Krishna, and Ashutosh Trivedi, *Fo-definable transformations of infinite strings*, 36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016, December 13-15, 2016, Chennai, India, 2016, pp. 12:1–12:14. [Cited on page 19.]

[DRT16] Laure Daviaud, Pierre-Alain Reynier, and Jean-Marc Talbot, *A generalised twinning property for minimisation of cost register automata*, Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016, 2016, pp. 857–866. [Cited on pages vii and 3.]

[dS13] Rodrigo de Souza, *Uniformisation of two-way transducers*, Language and Automata Theory and Applications - 7th International Conference, LATA 2013, Bilbao, Spain, April 2-5, 2013. Proceedings, 2013, pp. 547–558. [Cited on page 119.]

[EF95] Heinz-Dieter Ebbinghaus and Jörg Flum, *Finite model theory*, Perspectives in Mathematical Logic, Springer, 1995. [Cited on page 20.]

[EH01] Joost Engelfriet and Hendrik Jan Hoogeboom, *MSO definable string transductions and two-way finite-state transducers*, ACM Trans. Comput. Log. **2** (2001), no. 2, 216–254. [Cited on pages vi, 2, 24, 91, and 95.]

[Eil74] Samuel Eilenberg, *Automata, languages, and machines. A*, Pure and applied mathematics, Academic Press, 1974. [Cited on pages ix, 5, 23, 28, and 35.]

[Eil76] ———, *Automata, languages, and machines., B*, Pure and applied mathematics, Academic Press, 1976. [Cited on pages vii, 3, and 17.]

[Elg61] Calvin C. Elgot, *Decision problems of finite automata design and related arithmetics*, Transactions of the American Mathematical Society **98** (1961), no. 1, 21–51. [Cited on page 21.]

[EM65] Calvin C. Elgot and Jorge E. Mezei, *On relations defined by generalized finite automata*, IBM Journal of Research and Development **9** (1965), no. 1, 47–68. [Cited on pages 24, 44, and 122.]

[FGL16a] Emmanuel Filiot, Olivier Gauwin, and Nathan Lhote, *First-order definability of rational transductions: An algebraic approach*, Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016, 2016, pp. 387–396. [Cited on page 28.]

[FGL16b] ———, *Aperiodicity of rational functions is PSpace-complete*, 36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016, December 13-15, 2016, Chennai, India, 2016, pp. 13:1–13:15. [Cited on page 28.]

[Fil15] Emmanuel Filiot, *Logic-automata connections for transformations*, Logic and Its Applications - 6th Indian Conference, ICLA 2015, Mumbai, India, January 8-10, 2015. Proceedings, 2015, pp. 30–57. [Cited on pages vi, ix, 2, 4, 24, and 25.]

[FJLW16] Emmanuel Filiot, Ismaël Jecker, Christof Löding, and Sarah Winter, *On equivalence and uniformisation problems for finite transducers*, 43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy, 2016, pp. 125:1–125:14. [Cited on pages viii, 4, and 127.]

[FKT14] Emmanuel Filiot, Shankara Narayanan Krishna, and Ashutosh Trivedi, *First-order definable string transformations*, 34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India, 2014, pp. 147–159. [Cited on pages viii and 4.]

[FMR18] Emmanuel Filiot, Nicolas Mazzocchi, and Jean-François Raskin, *A pattern logic for automata with outputs*, Developments in Language Theory - 22nd International Conference, DLT2018, Tokyo, Japan, Septemeber 10-14, 2018, Proceedings, 2018. [Cited on pages 48 and 81.]

[FR16] Emmanuel Filiot and Pierre-Alain Reynier, *Transducers, logic and algebra for functions of finite words*, SIGLOG News **3** (2016), no. 3, 4–19. [Cited on pages vi and 2.]

[GI83] Eitan M. Gurari and Oscar H. Ibarra, *A note on finitely-valued and finitely ambiguous transducers*, Mathematical Systems Theory **16** (1983), no. 1, 61–66. [Cited on page 23.]

[Gir86] Françoise Gire, *Two decidability problems for infinite words*, Inf. Process. Lett. **22** (1986), no. 3, 135–140. [Cited on page 23.]

[GO99] Erich Grädel and Martin Otto, *On logics with two variables*, Theor. Comput. Sci. **224** (1999), no. 1-2, 73–113. [Cited on pages 99 and 101.]

[Hop71] John E. Hopcroft, *An n log n algorithm for minimizing states in a finite automaton*, Tech. report, Stanford University, Stanford, CA, USA, 1971. [Cited on pages vii, 3, and 16.]

[Lad77] Richard E. Ladner, *Application of model theoretic games to discrete linear orders and finite automata*, Information and Control **33** (1977), no. 4, 281–303. [Cited on page 21.]

[LLN$^+$11] Grégoire Laurence, Aurélien Lemay, Joachim Niehren, Slawek Staworko, and Marc Tommasi, *Normalization of sequential top-down tree-to-word transducers*, Language and Automata Theory and Applications - 5th International Conference, LATA 2011, Tarragona, Spain, May 26-31, 2011. Proceedings, 2011, pp. 354–365. [Cited on page 126.]

[LW17] Christof Löding and Sarah Winter, *Synthesis of deterministic top-down tree transducers from automatic tree relations*, Inf. Comput. **253** (2017), 336–354. [Cited on pages viii and 4.]

[McN66] Robert McNaughton, *Testing and generating infinite sequences by a finite automaton*, Information and Control **9** (1966), no. 5, 521–530. [Cited on pages 14 and 21.]

[Moo56] Edward F. Moore, *Gedanken-experiments on sequential machines*, Automata studies, Annals of mathematics studies, no. 34, Princeton University Press, Princeton, N. J., 1956, pp. 129–153. MR 0078059 [Cited on pages vii, 3, 16, and 32.]

[MP71] Robert McNaughton and Seymour Papert, *Counter-free automata (M.I.T. research monograph no. 65)*, The MIT Press, 1971. [Cited on pages vii, 3, 18, 21, and 56.]

[MSZ13] Amaldev Manuel, Thomas Schwentick, and Thomas Zeume, *A short note on two-variable logic with a linear order successor and a preorder successor*, CoRR **abs/1306.3418** (2013). [Cited on pages 96 and 118.]

[MZ13] Amaldev Manuel and Thomas Zeume, *Two-variable logic on 2-dimensional structures*, Computer Science Logic 2013 (CSL 2013), CSL 2013, September 2-5, 2013, Torino, Italy, 2013, pp. 484–499. [Cited on page 127.]

[Ner63] Anil Nerode, *Linear automaton transformations*, Journal of Symbolic Logic **28** (1963), no. 2, 173–174. [Cited on pages vii, 2, and 15.]

[NPTT05] Joachim Niehren, Laurent Planque, Jean-Marc Talbot, and Sophie Tison, *N-ary queries by tree automata*, Database Programming Languages, 10th International Symposium, DBPL 2005, Trondheim, Norway, August 28-29, 2005, Revised Selected Papers, 2005, pp. 217–231. [Cited on page 104.]

[Per84] Dominique Perrin, *Recent results on automata and infinite words*, Mathematical Foundations of Computer Science 1984 (Berlin, Heidelberg) (M. P. Chytil and V. Koubek, eds.), Springer Berlin Heidelberg, 1984, pp. 134–148. [Cited on page 21.]

[Pin95] Jean-Éric Pin, *A variety theorem without complementation.*, Russian Mathematics (iz. VUZ) **39** (1995), 80–90. [Cited on pages vii and 3.]

[Pin18] ———, *Mathematical foundations of automata theory*, 2018. [Cited on pages 11 and 17.]

[PP04] Dominique Perrin and Jean-Eric Pin, *Infinite Words, Automata, Semigroups, Logic and Games*, vol. 141, Elsevier, 2004. [Cited on page 22.]

[PR89] Amir Pnueli and Roni Rosner, *On the synthesis of a reactive module*, Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, January 11-13, 1989, 1989, pp. 179–190. [Cited on pages viii and 4.]

[Pri02] Christophe Prieur, *How to decide continuity of rational functions on infinite words*, Theor. Comput. Sci. **276** (2002), no. 1-2, 445–447. [Cited on page 23.]

[PW95] Jean-Éric Pin and Pascal Weil, *Polynomial closure and unambiguous product*, Automata, Languages and Programming, 22nd International Colloquium, ICALP95, Szeged, Hungary, July 10-14, 1995, Proceedings, 1995, pp. 348–359. [Cited on page 21.]

[RS59] Michael O. Rabin and Dana S. Scott, *Finite automata and their decision problems*, IBM Journal of Research and Development **3** (1959), no. 2, 114–125. [Cited on page 14.]

[RS91] Christophe Reutenauer and Marcel Paul Schützenberger, *Minimization of rational word functions*, SIAM J. Comput. **20** (1991), no. 4, 669–685. [Cited on pages ix, x, 5, 22, 27, 36, 38, 40, 47, 48, 49, 79, 125, and 126.]

[RS95] ______, *Variétés et fonctions rationnelles*, Theor. Comput. Sci. **145** (1995), no. 1&2, 229–240. [Cited on pages 24, 35, 37, 38, and 42.]

[Sak09] Jacques Sakarovitch, *Elements of automata theory*, Cambridge University Press, 2009. [Cited on page 11.]

[Sch61] Marcel-Paul Schützenberger, *A remark on finite transducers*, Information and Control **4** (1961), no. 2-3, 185–196. [Cited on pages ix, 5, 28, and 35.]

[Sch65] ______, *On finite monoids having only trivial subgroups*, Information and Control **8** (1965), no. 2, 190–194. [Cited on pages vii, 3, 21, and 56.]

[Sch76a] ______, *Sur le produit de concatenation non ambigu*, Semigroup Forum **13** (1976), no. 1, 47–75. [Cited on page 21.]

[Sch76b] ______, *Sur les relations rationnelles entre monoides libres*, Theor. Comput. Sci. **3** (1976), no. 2, 243–259. [Cited on page 23.]

[Sim75] Imre Simon, *Piecewise testable events*, Automata Theory and Formal Languages, 2nd GI Conference, Kaiserslautern, May 20-23, 1975, 1975, pp. 214–222. [Cited on pages 21 and 56.]

[Str94] Howard Straubing, *Finite automata, formal logic, and circuit complexity*, Progress in Computer Science and Applied Series, Birkhäuser, 1994. [Cited on pages vii, 3, 11, 17, and 125.]

[SZ12] Thomas Schwentick and Thomas Zeume, *Two-variable logic with two order relations*, Logical Methods in Computer Science **8** (2012), no. 1. [Cited on pages xi, 7, 92, 93, 102, 103, 114, 115, and 116.]

[Tan14] Tony Tan, *Extending two-variable logic on data trees with order on data values and its automata*, ACM Trans. Comput. Log. **15** (2014), no. 1, 8:1–8:39. [Cited on page 127.]

[Tho79] Wolfgang Thomas, *Star-free regular sets of omega-sequences*, Information and Control **42** (1979), no. 2, 148–156. [Cited on page 21.]

[Tho81] ______, *A combinatorial approach to the theory of omega-automata*, Information and Control **48** (1981), no. 3, 261–283. [Cited on page 19.]

[Tho09] ______, *Facets of synthesis: Revisiting church's problem*, Foundations of Software Science and Computational Structures, 12th International Conference, FOSSACS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings, 2009, pp. 1–14. [Cited on pages viii and 4.]

[Tra61] Boris A. Trakhtenbrot, *Finite automata and the logic of monadic predicates*, Doklady Akademii Nauk SSSR **140** (1961), 326–329 (Russian). [Cited on page 21.]

[TW98] Denis Thérien and Thomas Wilke, *Over words, two variables are as powerful as one quantifier alternation*, Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998, 1998, pp. 234–240. [Cited on pages 21 and 56.]

[WK94] Andreas Weber and Reinhard Klemm, *Economy of description for single-valued transducers*, STACS 94, 11th Annual Symposium on Theoretical Aspects of Computer Science, Caen, France, February 24-26, 1994, Proceedings, 1994, pp. 607–618. [Cited on page 23.]