I'm not robot
reCAPTCHA
I'm not robot!

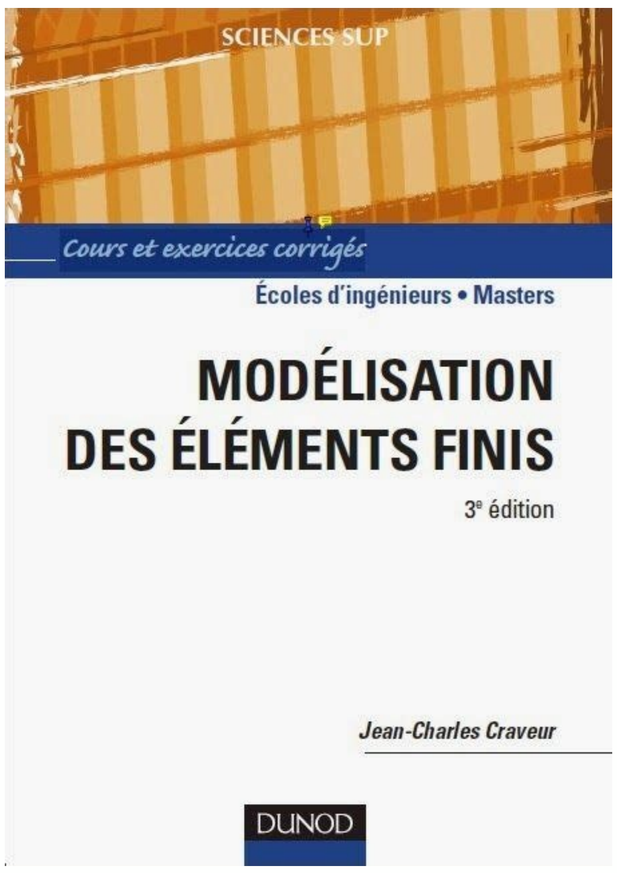
Exercices corrigés sur les automates finis pdf

Bienvenue dans cet article dont l’unique but est de vous aider à progresser avec des exercices sig corrigés du chapitre l’analyse de l’exploitation de la matière Gestion Opérationnelle du BTS MCO.Si vous souhaitez dans un premier temps voir ou revoir le cours sur le même thème, je vous invite à lire mon article L’analyse de l’exploitation : Les 4 piliers à maîtriserLes 16 exercices corrigés sur les soldes intermédiaires de gestion de cette page portent principalement sur l’étude du compte de résultat, la création du tableau des soldes intermédiaires de gestion, les fameux SIG.Vous trouverez également des exercices corrigés sur les notions suivantes : la capacité d’autofinancement c’at à dire la CAF. Nous allons voir les deux méthodes de résolution : la méthode additive et la méthode soustractive.Voici la liste des 16 exercices sig corrigés :Exercices SIG corrigés #1 : Calcul du résultat – Savoir le placerÉnoncéL’unité commerciale Cadastral fabrique des registres à destination de notaires et de mairies.Elle vous remet quelques éléments comptables afin d’effectuer afin de calculer son résultat pour l’exercice N :Achats de papier (ramettes) : 68 000 €Ventes de registres : 145 000 €Frais postaux : 870 €Impôts et taxes : 14 600 €Dotation aux amortissements : 11 000 €Charges de personnel : 25 000 €Frais de publicité : 4 500 €Travail à faireDéterminez le montant du résultat de l’unité commerciale Cadastral.Précisez sa place exacte au sein du compte de résultat.Proposition de correctionDéterminez le montant du résultat de l’unité commerciale Cadastral.Pour calculer le résultat de l’unité commerciale, il faut distinguer les charges des produits.Calcul des produits :Il s’agit des ventes de registres : 145 000 €Calculs des charges :Il s’agit de tous les éléments restants, soit au total : 123 970 €Le résultat se calcule en faisant la différence entre le total des produits et le total des charges, soit donc :Résultat : 145 000 – 123 970 = 21 030 €Le résultat de l’unité commerciale est donc de 21 030 €, et il s’agit d’un bénéfice.Précisez sa place exacte au sein du compte de résultat.Il s’agit d’un bénéfice donc il se situera du côté des charges après la ligne « Total des charges ».ÉnoncéL’unité commerciale Cadavre fabrique et distribue des cercueils à destination de pompes funèbres dans la région de Nantes.Son responsable Monsieur Levivant souhaite connaitre la répartition de son résultat.C’est la raison pour laquelle il vous sollicite pour le calcul des soldes intermédiaires de gestion.Pour cela, il vous remet son compte de résultat en annexe 1.Annexe 1 : Compte de résultat de l’UC CadavreAnnexe 1 (suite) : Compte de résultat de l’UC CadavreAnnexe 1 (suite et fin) : Compte de résultat de l’UC CadavreTravail à faireCalculez les SIG (Annexe 2 à compléter).Proposition de correctionTableau des soldes intermédiaires de gestion – SIG(1) : Il s’agit du poste ventes de marchandises(2) : le coût d’achat des marchandises vendues est égal à la différence entre les achats de marchandises et la variation des stocks de marchandises.(3) : la marge commerciale est égale à la différence entre les ventes de marchandises et le coût d’achat des marchandises vendues.(4) : le montant correspond à la rubrique ventes de produits finis.(5) : il n’y a pas de déstockage donc le montant correspond à la rubrique Production vendue.(6) : il s’agit de la somme des soldes déjà calculés : la marge commerciale et la production de l’exercice.(7) : la rubrique consommations de l’exercice en provenance des tiers est la somme des achats stockés des matières premières, de la variation des stocks de matières premières, l’entretien et réparations, la prime d’assurance, les rémunérations d’intermédiaires et honoraires, le poste Publicité, Transport de biens, les frais postaux et enfin les services bancaires.(8) : valeur ajoutée = Marge commerciale + production de l’exercice – consommations de l’exercice en provenance des tiersTableau des soldes intermédiaires de gestion corrigé – SIG (suite)(9) : uniquement le poste valeur ajoutée car il n’y a pas de subventions d’exploitation.(10) : il s’agit des postes impôts et taxes, et charges du personnel.(12) : (9) – (10)(13) : il faut recopier le solde que l’on vient de calculer l’excédent brut d’exploitation (EBE)(14) : Dotations aux amortissements + dotations aux provisions(15) : (13) – (14)(16) : vous devez reprendre le résultat d’exploitation auquel vous ajoutez les produits d’exploitation(17) : charges d’intérêts + escomptes accordés(18) : (16) – (17)(19) : le total des produits exceptionnels(20) : le total des charges exceptionnelles(21) : la différence entre (19) et (20)(22) : vous devez reprendre le résultat courant avant impôt auquel vous devez ajouter le résultat exceptionnel. Ces deux soldes ont déjà été calculés.(23) : il n’y a rien à mettre car les soldes sont positifs.(24) : (22) – (23)Tableau des soldes intermédiaires de gestion corrigé – SIG (fin)(25) : il faut indiquer l’équivalent de la rubrique produit des cessions d’éléments d’actif : il s’agit du poste « sur opérations en capital » côté Produits exceptionnels.(26) : il faut indiquer l’équivalent de la rubrique valeur comptable des éléments d’actif cédés : il s’agit du poste « sur opérations en capital » côté Charges exceptionnelles.ÉnoncéL’unité commerciale Cadeau confectionne et vend des gâteaux pour les entreprises mais aussi pour les particuliers dans la région de Marseille.Vous êtes chargé(e) d’analyser la composition de son compte de résultat.Vous disposez pour cela de son compte d’exploitation au 31/12/N-1Annexe 1 : Compte d’exploitation au 31/12/N-1Travail à faireCalculez les SIG (Annexe 2 à compléter).Proposition de correctionTableau des soldes intermédiaires corrigé(1) : Achats de matières premières + Autres achats et charges externes(2) : Impôts et taxes + salaires + charges socialesTableau des soldes intermédiaires corrigé (suite et fin)(3) : reprise du solde déjà calculé Résultat d’exploitation auquel vous ajoutez les produits financiers(4) : il s’agit du résultat exceptionnel négatif déjà calculé auquel vous ajoutez le poste Impôts sur les bénéfices.Exercices SIG corrigés #4 : Calculez une CAFÉnoncéL’unité commerciale Cadenas fabrique et vend des cadenas destinés à des galeries marchandes situées à l’intérieur des aéroports.Afin d’analyser sa situation financière, vous êtes chargé(e) de calculer sa capacité d’autofinancement.Pour cela, vous disposez du compte de résultat (annexe 1) et du tableau des SIG (annexe 2) de l’unité commerciale.Annexe 1 : Compte de résultatTravail à faireCalculez le montant de la CAF.Proposition de correctionPour répondre à la question on peut utiliser la méthode additive.Résultat net : 33 885 €+ Charges non décaissées :Dotation aux amortissements et provisions (charges d’exploitation) : 1 300 €Dotation aux amortissements et provisions (charges financières) : 0 €Dotation aux amortissements et provisions (charges exceptionnelles) : 0 €– Produits non encaissés :Reprises sur amortissements et provisions (produits d’exploitation) : 0 €Reprises sur amortissements et provisions (produits financiers) : 0 €Reprises sur amortissements et provisions (produits exceptionnels) : 0 €Produits exceptionnels (sur opération en capital) : 120 €Capacité d’autofinancement : 35 065 €La capacité d’autofinancement est donc de 35 065 €.Elle permettra de financer d’éventuels renouvellements d’immobilisations ou de rémunérer les actionnaires.Exercices SIG corrigés #5 : Calculez une CAFÉnoncéL’unité commerciale Cadran fabrique et vend des pièces détachées pour montres et horloges dans la région de Bordeaux.Afin d’analyser sa situation financière, vous êtes chargé(e) de calculer sa capacité d’autofinancement.Pour cela, vous disposez du compte de résultat (annexe 1) et du tableau des SIG (annexe 2) de l’unité commerciale.Annexe 1 : Compte de résultatTravail à faireCalculez le montant de la CAF.Proposition de correctionPour répondre à la question on peut utiliser la méthode soustractive.Produits donnant lieu à encaissements :Ventes de produits finis : 35 000 €Ventes de marchandises : 15 000 €Escomptes obtenus : 10 €– Charges donnant lieu à décaissements :Achats stockés de matières premières : 5 000 €Variation des stocks de matières premières : 1 000 €Achats de marchandises : 90 €Variation des stocks de marchandises : 20 €Entretien et réparations : 2 000 €Prime d’assurance : 1 500 €Rémunérations d’intermédiaires et honoraires : 240 €Publicité, publications, relations publiques : 160 €Transports de biens et transports collectifs : 120 €Frais postaux de télécommunications : 50 €Services bancaires et assimilés : 70 €Impôts, taxes et versements assimilés : 1 500 €Charges de personnel : 3 000 €Charges d’intérêts : 150 €Escomptes accordés : 20 €Pénalités, amendes fiscales et pénales : 25 €Capacité d’autofinancement : 35 065 €La capacité d’autofinancement est donc de 35 065 €.Elle permettra de financer d’éventuels renouvellements d’immobilisations ou alors de rémunérer les actionnairesExercices SIG corrigés #6 : Calculez une CAF – Méthode additiveÉnoncéOn donne le compte de résultat suivant :Travail à faireCalculez la CAF selon la méthode additive.Proposition de correctionLa méthode additive prend en compte les postes Dotation aux amortissements et la rubrique Bénéfice.CAF = Bénéfice + Dotation aux amortissementsCAF = 3 300 + 200 = 3 500 €Exercices SIG corrigés #7 : Calculez une CAF – Méthode soustractiveÉnoncéOn donne le compte de résultat suivant :Travail à faireCalculez la CAF selon la méthode soustractive.Proposition de correctionLa méthode soustractive consiste « globalement » en une soustraction entre les produits et les charges.CAF = somme des produits (hormis les reprises sur amortissements) – somme des charges (hormis les amortissements)CAF = (14 000 – 2 000) – (8 700 – 200)CAF = 3 500 €Exercices SIG corrigés #8 : Calcul du Résultat de l’exercice – BénéficeÉnoncéOn donne le tableau suivant :Travail à faireCalculez le résultat de l’exercice.Proposition de correctionNous allons appliquer la formule suivante :Résultat de l’exercice = Total des produits – Total des chargesRésultat de l’exercice = (12 000 + 2 000) – (4 000 + 1 500 + 2 500 + 500 +200)Résultat de l’exercice = 5 300 €Interprétation :L’entreprise réalise un bénéfice de 5 300 €.Exercices SIG corrigés #9 : Calcul du Résultat de l’exercice – PerteÉnoncéOn donne le tableau suivant :Travail à faireCalculez le résultat de l’exercice.Proposition de correctionNous allons appliquer la formule suivante :Résultat de l’exercice = Total des produits – Total des chargesRésultat de l’exercice = 12 000 – (6 000 + 2 500 + 4 500 + 1 500 +2 200)Résultat de l’exercice = – 4 700 €Interprétation :L’entreprise réalise une perte de 4 700 €.ÉnoncéOn donne le compte de résultat suivant :Travail à fairePrésentez les soldes intermédiaires de gestion.Proposition de correctionÉnoncéOn donne le compte de résultat suivant :Travail à fairePrésentez les soldes intermédiaires de gestion.Proposition de correctionÉnoncéOn donne le compte de résultat suivant :Travail à fairePrésentez le tableau des soldes intermédiaires de gestion.Proposition de correctionÉnoncéOn donne le compte de résultat suivant :Travail à fairePrésentez le tableau des soldes intermédiaires de gestion.Proposition de correctionExercices SIG corrigés #14 : Calcul d’un chiffre d’affairesÉnoncéOn donne un extrait du compte de résultat.Travail à faireDéterminez le montant du chiffre d’affaires.Proposition de correctionDans un compte de résultat, le montant du chiffre d’affaires peut correspondre à la rubrique « Ventes de marchandises ».C’est le cas dans cet exercice. Le montant du chiffre d’affaires est donc de 4 000 €.Exercices SIG corrigés #15 : Calcul d’un chiffre d’affairesÉnoncéOn donne l’extrait de compte de résultat suivant.Travail à faireDéterminez le montant du chiffre d’affaires.Proposition de correctionDans un compte de résultat, le montant du chiffre d’affaires peut correspondre à la rubrique « Prestation de services ».C’est le cas dans cet exercice. Le montant du chiffre d’affaires est donc de 12 000 €.Exercices SIG corrigés #16 : Calcul d’un chiffre d’affairesÉnoncéOn donne l’extrait de compte de résultat suivant.Travail à faireDéterminez le montant du chiffre d’affaires.Proposition de correctionDans un compte d’exploitation, on peut trouver les rubriques « Ventes de marchandises » et « Prestation de services », qui correspondent au montant total du chiffre d’affaires de l’entreprise.Dans ce cas précis, l’entreprise a une activité dite « mixte ».Dans cet exercice, le montant du chiffre d’affaires est donc de 23 500 € (= 12 000 € + 11 500 €). Cet article concerne les automates finis. Pour une présentation plus formelle, voir Automate fini non déterministe. Pour les autres significations, voir Automate. Pour les articles homonymes, voir FSM. Fig. 1 : Une hiérarchie d'automates. Un automate fini ou automate avec un nombre fini d'états (en anglais finite-state automaton ou finite state machine ou FSM) est un modèle mathématique de calcul, utilisé dans de nombreuses circonstances, allant de la conception de programmes informatiques et de circuits en logique séquentielle aux applications dans des protocoles de communication, en passant par le contrôle des processus, la linguistique et même la biologie. Un automate fini est une construction mathématique abstraite, susceptible d'être dans un nombre fini d'états, mais étant un moment donné dans un seul état à la fois ; l'état dans lequel il se trouve alors est appelé l'« état courant ». Le passage d'un état à un autre est activé par un événement ou une condition ; ce passage est appelé une « transition ». Un automate particulier est défini par l'ensemble de ses états et l'ensemble de ses transitions. On rencontre couramment des automates finis dans de nombreux appareils qui réalisent des actions déterminées en fonction des événements qui se présentent. Un exemple est un distributeur automatique de boissons qui délivre l'article souhaité quand le montant introduit est approprié. D'autres exemples sont les ascenseurs qui savent combiner les appels successifs pour s'arrêter aux étages intermédiaires, les feux de circulation qui savent s'adapter aux voitures en attente, les digicodes qui analysent la bonne suite de chiffres. Les automates finis peuvent modéliser un grand nombre de problèmes, parmi lesquels la conception assistée par ordinateur pour l'électronique, la conception de protocoles de communication, l'analyse syntaxique de langages. Dans la recherche en biologie et en intelligence artificielle, les automates finis ou des hiérarchies de telles machines ont été employés pour décrire des systèmes neurologiques. En linguistique, ils sont utilisés pour décrire les parties simples de grammaires de langues naturelles. En vérification de programmes (model checking), des automates finis, avec parfois un nombre très important d'états (des milliards), sont employés. Vus comme un modèle de calcul les automates finis ont un potentiel faible ; ils ont bien moins de puissance de calcul qu'une machine de Turing. En d'autres termes, il y a des tâches qu'un automate fini ne peut pas accomplir alors qu'un automate à pile ou une machine de Turing le pourront. Ceci est principalement dû au fait qu'un automate fini a une mémoire limitée à son nombre d'états. L'étude des automates finis est une branche de la théorie des automates. Exemples Un premier exemple : un portillon Fig. 2 : Diagramme d'état d'un portillon. Un portillon. Un exemple très simple d'un mécanisme que l'on peut modéliser par un automate fini est un portillon d'accès[1],[2]. Un portillon, utilisé dans certains métros ou dans d'autres établissements à accès contrôlés, est une barrière avec trois bras rotatifs à hauteur de la taille. Au début, les bras sont verrouillés et bloquent l'entrée, et empêchent les usagers de passer. L'introduction d'une pièce de monnaie ou d'un jeton dans une fente du portillon (ou la présentation d'un ticket ou d'une carte) débloque les bras et permet le passage d'un et un seul usager à la fois. Une fois l'usager entré, les bras sont à nouveau bloqués jusqu'à ce qu'un nouveau jeton soit inséré. Un portillon, vu comme un automate fini, a deux états : verrouillé (« locked » en anglais) et déverrouillé (« unlocked » en anglais)[1]. Deux « entrées » peuvent modifier l'état : la première si l'on insère un jeton dans la fente (entrée jeton) et la deuxième si l'on pousse le bras (entrée pousser). Dans l'état verrouillé, l'action de pousser n'a aucun effet : quel que soit le nombre de fois que l'on pousse, l'automate reste verrouillé. Si l'on insère un jeton, c'est-à-dire si l'on effectue une « entrée » jeton, on passe de l'état verrouillé à l'état déverrouillé. Dans l'état déverrouillé, ajouter des jetons supplémentaires n'a pas d'effet, et ne change pas l'état. Mais dès qu'un usager tourne le bras du portillon, donc fournit un pousser, la machine retourne à l'état verrouillé. L'automate d'un portillon peut être représenté par une table de transition d'états qui montre, pour chaque état, le nouvel état et la sortie (l'action) pour une entrée donnée. État courant Entrée État suivant Sortie verrouillé jeton déverrouillé Déverrouille le portillon pour qu'un usager puisse passer pousser verrouillé Rien déverrouillé jeton déverrouillé Rien pousser verrouillé Quand l'usager est passé, verrouille le portillon On peut aussi représenter l'automate par un graphe orienté, appelé un diagramme états-transitions, comme donné ci-dessus. Chaque état est représenté par un sommet (visualisé par un cercle). Les arcs (représentés par des flèches) montrent les transitions d'un état à un autre. Chaque flèche porte une entrée qui déclenche la transition. Les données qui ne provoquent pas de changement d'état, comme un jeton pour l'état déverrouillé, sont représentées par des arcs circulaires (des boucles) qui tournent autour de l'état. La flèche qui entre dans l'état verrouillé depuis le point noir sert à indiquer que cet état est l'état initial, au début de la modélisation. Un autre exemple : le loup, la chèvre et le chou Le loup, la chèvre et le chou. Chaque état représente ce que le passeur a déjà transporté (le chou est noté « S »). L'exemple que voici[3] illustre les possibilités qui s'offrent à un passeur qui doit faire traverser, d'une rive à l'autre, un loup, une chèvre et un chou (c'est une variante des nombreux problèmes de passage de rivière). Sa barque ne lui permet que d'emporter un seul des trois objets à la fois et, bien entendu, il ne peut laisser ensemble loup et chèvre ni chèvre et chou. Dans le diagramme ci-contre, un état représente ce que le passeur a déjà pu transporter sur l’autre rive (« P » représente le passeur, « L » le loup, « C » la chèvre, et le chou a été noté par « S ») : au début rien, à la fin tout. Sur les flèches, les objets transportés (et lui-même). Une des deux séquences de transport est CPLCSPC, l'autre est CPSCLPC. Bien entendu, on néglige les aller-retour-aller inutiles.
Concepts et terminologie Un état est la description de la configuration d'un système en attente d'exécuter une transition. Une transition est un ensemble d'actions à exécuter lorsqu'une condition est remplie ou lorsqu'un événement est reçu. Par exemple, une chaîne audio peut être dans un état « radio » et, quand elle reçoit un stimulus du genre « suivant », passe à la station radio suivante. Si le système est dans l'état « CD » et reçoit le stimulus « suivant », il passe à la piste suivante du CD en cours. Les mêmes stimuli peuvent donc déclencher des actions différentes, si l'état courant n'est pas le même.
Dans certaines représentations de machines finies, il est possible d'associer des actions à un état : action d'entrée : réalisée lorsque l'on « entre » dans l'état ; action de sortie : réalisée lorsque l’on « quitte » l'état ; action de transition : réalisée lors d'une transition Représentations Table états-transitions Article détaillé : Diagramme états-transitions. Plusieurs types de tables de transition d'état sont utilisées. Ces diagrammes sont très populaires en UML notamment.
La représentation la plus courante est illustrée ci-dessous ; la combinaison de l'état courant (par exemple B) et d'une entrée (par exemple Y) montre l'état suivant (dans l’exemple C).

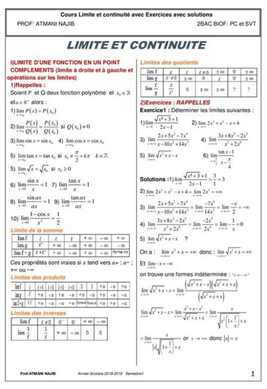
Cartouche du document
Année : ING 1 - Matière : Théorie des langages - Activité : Travail dirigé
Objectifs
Ce travail dirigé a pour but d'étudier la bijection entre les langages hors-contexte et les automates à pile.
Un langage hors contexte est aussi appelé langage algébrique
Une grammaire hors-contexte (ou algébrique) est un quadruplet T,N,S,P où :
• T : ensemble des éléments terminaux
• N : ensemble des éléments non terminaux
• S : élément non terminal initial (axiome)
• P : ensemble de règles de la forme :
• X —> a où a ∈ T et X ∈ N
• X —> Y où Y ∈ (N ∪ T)* et X ∈ N
Sommaire des exercices
1 - Langages algébriques et automates à piles
Corps des exercices
1 - Langages algébriques et automates à piles
Enoncé :
Dans ces exercices, nous chercherons à montrer qu'un langage est algébrique en trouvant une grammaire algébrique le représentant. Puis, le langage étant de type 2, nous chercherons un automate à pile pour le représenter.
Question 1)
Énoncé de la question
Soit l'alphabet A = {a, b} et le langage L_1 = {$a^n b^n$}. Ecrire la grammaire de ce langage et montrer que c'est un langage algébrique.
Solution de la question
G = {
T = {a,b}
N = {S}
S = S
P = {
S —> b
S —> aS
}
}
On remarque que les règles respectent bien le format des grammaires de type 2.
On peut également constater qu'elles suivent également le format de type 3.
page 1

L'information complète associée à une action n'est pas décrite dans la table et peut être ajoutée par des annotations.
Une définition d'une machine finie avec actions est possible au moyen de tables de transition dans le modèle plus élaboré d'automate fini virtuel (en)[4]. Table de transitions EntréeÉtat Entrée X Entrée Y Entrée Z État A … … … État B … État C … État C … … … Automates UML Exemple de diagramme UML de fonctionnement d'un four. Le langage de spécification Unified Modeling Language (UML) a une notation propre pour décrire des automates. Ce sont des diagrammes comportementaux particuliers qui permettent de dépasser les limitations de machines finies traditionnelles tout en conservant leurs avantages principaux. Les machines ou automates UML introduisent les notions nouvelles de diagramme de structure composite, d'états hiérarchiques et d'agrégation, tout en étendant la notion d'action. Les machines finies UML ont à la fois les propriétés des machines de Mealy et des machines de Moore décrites plus bas. Elles supportent des actions qui dépendent à la fois de l'état du système et de l'événement déclenchant, comme dans les machines de Mealy, et aussi des actions d'entrée et de sortie associées à des états, comme dans les machines de Moore. Machines SDL Exemple d'une machine finie en SDL. Fig. 3 : Exemple d'un automate fini à deux états.

Le langage de spécification et de description appelé Specification and Description Language ou SDL est une norme élaborée par l'Union internationale des télécommunications (UIT)[5]. La norme contient un ensemble de symboles graphiques permettant de décrire les actions dans une transition : envoi d'un message ; réception d'un message ; démarrage d'un temporisateur (timer) ; arrêt d'un temporisateur ; lancement d'une machine concurrente ; prise de décision.
Un SDL contient des types de base, mais surtout des types de données abstraits (TDA) et une syntaxe de manipulation qui permet de décrire le système de manière formelle, possiblement de manière complète et non ambiguë. Autres diagrammes d'états Il existe de nombreuses variantes de diagrammes d'états-transitions, comme le diagramme d'un automate à deux états donné plus haut. Utilisation Les automates sont des systèmes réactifs (en)[6], réagissant aux impulsions reçues. Ils jouent un rôle important dans de nombreux champs différents, comprenant l'électrotechnique, la linguistique, l'informatique , la philosophie, la biologie, les mathématiques, et la logique. Les automates finis constituent une classe d'automates étudiée en théorie des automates et en informatique théorique. En informatique, les automates finis sont largement utilisés en modélisation du comportement d'applications, en conception matérielle, en électronique numérique, en génie logiciel, en compilation, en protocoles de communication, dans l'étude des modèles de calcul et des langages formels. Les dictionnaires linguistiques aussi peuvent être représentés par un automate fini. Le gain de place, pour un dictionnaire de Scrabble anglais, peut atteindre 80 %[3], et encore plus pour des dictionnaires de mots fléchés du français.
Classification Les automates finis peuvent être classés principalement en deux catégories, les accepteurs et les transducteurs. Les accepteurs analysent la structure de la donnée fournie, et l'acceptent si elle est conforme à la spécification décrite par l'automate. Les transducteurs au contraire traduisent une chaîne de symboles en une autre, là encore selon l'algorithme codé dans l'automate. Dans certains cas, on peut rencontrer des variantes appelées classificateurs et séquenceurs[7],[3]. Accepteurs ou reconnaisseurs Fig. 4 : Automate accepteur : analyse du mot « nice ». Les accepteurs, également appelés reconnaisseurs produisent une sortie binaire, indiquant si l'entrée reçue est acceptée ou non.



Chaque état d'un tel automate est soit un état d'acceptation, aussi appelé final ou terminal, ou un état de rejet. Si l'état courant, après la lecture de la totalité de l'entrée, est un état d'acceptation, l'entrée est acceptée, sinon elle est rejetée. L'entrée est généralement une suite de symboles (des lettres); il n'y a pas d'actions associées. L'exemple de la figure 4 montre un automate fini qui accepte le mot « nice ». Dans cet automate, seul l'état 7 est acceptant. Une machine peut aussi être décrite comme définissant un langage formel. Ce langage est composé des mots acceptés par la machine, et d'aucun mot rejeté par elle. Par définition, les langages acceptés par un automate fini sont appelés les langages reconnaissables. Par le théorème de Kleene, ce sont les langages réguliers ou rationnels, décrits par des expressions régulières ou rationnelles. Le problème de déterminer le langage accepté par un automate fini donné est une instance d'un problème plus général appelé le problème algébrique de cheminement (« algebraic path problem »), qui lui-même est une extension des problèmes de cheminement dans des graphes dont les arcs portent des poids pris dans un demi-anneau arbitraire[8],[9]. État initial L'état initial est en général indiqué en traçant une flèche qui pointe vers cet état « à partir de n'importe où »[10]. États d'acceptation, finaux, ou terminaux Fig. 5 : Représentation d'un automate fini qui détermine si une suite binaire contient un nombre pair de 0. L'état S 1 {\displaystyle S_{1}} est final. Un état d'acceptation (aussi appelé état acceptant, final ou terminal) est un état dans lequel la machine déclare que la chaîne d'entrée traitée jusqu'alors appartient au langage qu'elle reconnaît. Graphiquement les états d'acceptation sont fréquemment représentés par des cercles doublés. Une autre façon, symétrique à celle adoptée pour l'état initial, consiste à faire sortir une flèche « pointant vers nulle part » d'un tel état[3]. L'état initial peut aussi être un état final ; dans ce cas, l'automate accepte la chaîne vide. Si l'état initial n'est pas un état d'acceptation, et s'il n'existe pas d'arc vers un état final, l'automate n'accepte aucun mot. La figure 5 est l'exemple d'un automate fini déterministe. Cette automate accepte les suites binaires qui comportent un nombre pair de chiffres 0. L'état S1 est à la fois l'état initial et l'état terminal. L'automate termine dans l'état S1 si la chaîne lue contient un nombre pair (éventuellement nul) de 0 : chaque 0 fait basculer d'un état vers l'autre, alors qu'un 1 laisse l'état inchangé. Des exemples de chaînes acceptées sont ε , le mot vide, puis 1, 11, 11..., 00, 010, 1010, 10110, etc. Transducteurs Fig. 6 : Automate transducteur : modèle de Moore. Article détaillé : Transducteur fini. Les transducteurs finis génèrent en sortie des mots en fonction d'un mot d'entrée donné et d'actions associées aux états. Ils sont utilisés par exemple dans des applications de contrôle et dans le domaine de la linguistique informatique. On distingue deux types de transducteurs, les machines de Moore et les machines de Mealy. Elles diffèrent par les modalités qui déterminent les sorties. On peut démontrer qu'elles ont la même puissance d'expression.


Cours Limite et continuité avec Exercices avec solutions

PROF: ATMANI NAJIB 2BAC BIOF: PC et SVT

LIMITE ET CONTINUITE

Machine de Moore Dans le modèle de Moore, qui utilise seulement des actions d'entrée, la sortie dépend uniquement de l'état courant. Comparé au modèle de Mealy, le modèle de Moore a l'avantage de la simplicité et de facilité de compréhension. Considérons par exemple une porte d'ascenseur, modélisée par une machine qui reconnaît deux commandes : « ouvrir » et « fermer », commandes qui peuvent être données par un utilisateur. L'action d'entrée (E:) dans l'état d'« en cours d'ouverture » fait démarrer un moteur qui ouvre la porte, et dans l'état « en cours de fermeture » fait démarrer un moteur qui ferme la porte. Les états « ouvert » et « fermé » arrêtent le moteur quand la porte est entièrement ouverte ou entièrement fermée. Ils signalent par ailleurs cette situation vers l'extérieur par « porte ouverte » ou « porte fermée ». Fig. 7 : Transducteur : modèle de Mealy. Machine de Mealy Dans le modèle de Mealy, qui utilise également des actions d'entrée, la sortie dépend à la fois de l'entrée et de l'état. L'usage de machines de Mealy réduit souvent le nombre d'états. En contrepartie, le fonctionnement d'un automate est plus complexe et plus difficile à appréhender. La figure 7 montre un automate de Mealy qui a le même comportement que l'automate de Moore. Il y a deux actions d'entrée (I:): « démarrer le moteur pour fermer la porte quand la commande de fermeture arrive », et la commande symétrique pour l'ouverture.

Les états intermédiaires « en cours d'ouverture » et « en cours de fermeture » ne sont pas nécessaires. Variantes Les variantes suivantes sont d'un emploi assez rare. Un classificateur est similaire à un accepteur, mais possède au moins deux états terminaux. Un tel automate peut donc accepter plusieurs langages simultanément, selon la catégorie d'états à laquelle appartient l'état terminal atteint en fin de lecture. Un séquenceur ou générateur est un cas particulier d'automate, opérant sur un alphabet d'entrée à une seule lettre. Un tel automate produit une seule séquence qui peut être interprétée comme le résultat d'un transducteur ou d'un classificateur. Un automate fini avec un état unique est appelé parfois « combinatoire » ; il utilise uniquement des actions d'entrée. Cette notion peut servir lorsque plusieurs automates finis sont censés travailler de concert, en communiquant ; il peut alors être commode de considérer des automates combinatoires comme une brique dans un outil de conception[11]. Déterminisme Une distinction supplémentaire importante est celle entre automates finis déterministes et non déterministes. Dans un automate déterministe, chaque état possède au plus une transition pour chaque symbole d'entrée (et même exactement une dans le cas où l'automate est complet). Dans un automate non déterministe, un symbole d'entrée peut étiqueter une, plusieurs ou aucune transition pour un état donné. Cette distinction est importante en pratique, mais moins en théorie parce qu'il existe un algorithme (la construction par sous-ensembles) qui permet de transformer un automate fini non déterministe en un automate fini déterministe avec la même fonctionnalité.

Toutefois, et c'est là un biais qui peut entraver l'efficacité, l'algorithme de construction par sous-ensembles peut produire un automate dont la taille est exponentielle par rapport à la taille de l'automate de départ. C'est pourquoi certains algorithmes (comme le programme grep de Unix) tentent des compromis. Autres représentations Il existe d'autres moyens de représenter des machines à états.

Il existe par exemple des outils de modélisation et de conception de la logique de contrôleurs embarqués.

Ils combinent des états hiérarchiques de UML, de graphes de flots de contrôle, et des tables de vérité en un seul langage, ce qui donne un formalisme différent et un ensemble de sémantiques propre[12]. Ces diagrammes sont similaires aux représentations initialement introduites par David Harel[13] ; ils supportent des états hiérarchiquement imbriqués, des régions orthogonales au sens de UML, des actions sur les états et sur les transitions[14]. Modèle mathématique Articles détaillés : Automate fini non déterministe, automate fini déterministe et transducteur fini. Les définitions formelles sont les suivantes : Un automate fini déterministe A = (Σ , S , s 0 , δ , F) {\displaystyle {\mathcal {A}}=(\Sigma ,S,s_{0},\delta ,F)} est composé de : Σ {\displaystyle \Sigma } , un ensemble fini, non vide de lettres qui est l'alphabet d'entrée, S {\displaystyle S} , un ensemble fini, non vide d'états, s 0 {\displaystyle s_{0}} , l'état initial, élément de S {\displaystyle S} . Dans un automate non déterministe, s 0 {\displaystyle s_{0}} peut être un ensemble d'états. δ {\displaystyle \delta } , la fonction de transition d'états: δ : S × Σ → S {\displaystyle \delta :S\times \Sigma \rightarrow S} (dans un automate non déterministe, c'est une fonction δ : S × Σ → P (S) {\displaystyle \delta :S\times \Sigma \rightarrow {\mathfrak {P}}(S)} , donc à valeur dans l'ensemble des parties de S {\displaystyle S} , en d'autres termes δ {\displaystyle \delta } retourne un ensemble d'états). F {\displaystyle F} est l'ensemble des états terminaux, un sous-ensemble (éventuellement vide) de S {\displaystyle S} . Pour les automates déterministes ou non, il est fréquent de considérer le cas où δ {\displaystyle \delta } est une fonction partielle, donc où δ (q , a) {\displaystyle \delta (q,a)} n'est pas défini pour toute combinaison d'un état q ∈ S {\displaystyle q\in S} et d'un symbole a ∈ Σ {\displaystyle a\in \Sigma } . Si δ (q , a) {\displaystyle \delta (q,a)} n'est pas défini, l'automate signale une erreur et l'entrée est rejetée. Cette variante est utile dans la spécification d'une machine, mais certains algorithmes qui nécessitent que les fonctions soient totales, demandent une transformation préliminaire qui, au moyen de l'ajout d'un état supplémentaire, rendent la fonction de transition totale. Un automate fini est une machine de Turing très particulière, où la tête ne peut effectuer que des opérations de lecture (et pas d'écriture), et de plus se déplace toujours de la gauche vers la droite (et ne peut donc pas revenir en arrière)[15]. Un transducteur fini T = (Σ , Γ , S , s 0 , δ , ω) {\displaystyle {\mathcal {T}}=(\Sigma ,\Gamma ,S,s_{0},\delta ,\omega)} , est composé de : Σ {\displaystyle \Sigma } , un ensemble fini, non vide de lettres qui est l'alphabet d'entrée. Γ {\displaystyle \Gamma } , un ensemble fini, non vide de lettres qui est l'alphabet de sortie. S {\displaystyle S} , un ensemble fini, non vide d'états. s 0 {\displaystyle s_{0}} , l'état initial, élément de S {\displaystyle S} . Dans un automate non déterministe, s 0 {\displaystyle s_{0}} peut être un ensemble d'états. δ {\displaystyle \delta } est la fonction de transition d'états : δ : S × Σ → S {\displaystyle \delta :S\times \Sigma \rightarrow S} . ω {\displaystyle \omega } est la fonction de sortie. Si la fonction de sortie est fonction des états et de l'entrée (ω : S × Σ → Γ {\displaystyle \omega :S\times \Sigma \rightarrow \Gamma }) la définition correspond à celle du modèle de Mealy; si en revanche la fonction de sortie ne dépend que de l'état, (ω : S → Γ {\displaystyle \omega :S\rightarrow \Gamma }), on est en présence du modèle de Moore. Il n'est pas difficile de transformer un automate de Moore en un automate de Mealy en faisant ne dépendre la fonction de sortie que de l'état d'arrivée. La conversion réciproque, d'un automate de Mealy en un automate de Moore, est également possible, mais demande d'introduire des états supplémentaires[16]. Optimisation Article détaillé : Minimisation d'un automate fini déterministe. L'optimisation d'un automate fini signifie ici la recherche d'un automate fini déterministe avec le moins d'états réalisant la même fonction. L'algorithme le plus rapide est l'algorithme de Hopcroft datant de 1971[17],[18]. D'autres techniques sont l'algorithme de Moore et un algorithme de Brzozowski. Les automates finis acycliques, qui reconnaissent des ensembles finis de mots comme des dictionnaires peuvent être minimisés en temps linéaire[19]. Des études poussées ont été menées pour la comparaison en moyenne, dans le pire des cas, et pour la complexité générique de ces méthodes. L'optimisation d'un automate au sens de la recherche d'un automate fini déterministe ou non avec un nombre minimum d'états est plus complexe[18]. Implémentation Applications matérielles Le schéma électrique pour un compteur TTL à 4 bits. En électronique numérique, un automate fini peut être construit comme circuit logique programmable, ou un automate programmable industriel, avec des fonctions logiques réalisées par bascules ou relais. Une implémentation matérielle demande en général un registre pour stocker les variables d'état, un circuit de logique combinatoire qui détermine les transitions d'états, et un deuxième bloc de logique combinatoire qui détermine les sorties de l'automate. Une implémentation courante est le Richards controller (en). Un cas particulier des automates de Moore, où la sortie est directement connectée aux états, est connu sous le nom d'automate de Medvedev[20]. Il est par ailleurs recommandé, dans la conception de chips, de ne pas placer de circuit logique entre les entrées-sorties primaires et les registres, afin de minimiser les délais entre chips qui sont habituellement longs et ralentissent la fréquence des automates. On peut aussi optimiser les automates en vue de minimiser la consommation d'énergie par des techniques spécifiques (en). Applications logicielles Les concepts suivants sont fréquemment employés dans la construction d'applications utilisant des automates finis : Programmation fondée sur des automates (en) Automate fini dirigé par événements (en) Automate fini virtuel (en) État (patron de conception) Automates finis et compilateurs Les automates finis sont souvent utilisés au début du processus de compilation dans les compilateurs de langages de programmation. Une telle entité peut comporter plusieurs automates finis, qui implémentent l'analyse lexicale. Partant d'une séquence de caractères, un analyseur lexical construit une suite de lexèmes, tels que les mots réservés, les identificateurs, les littéraux ; à partir de cette suite, l'analyseur syntaxique construit l'arbre syntaxique. L'analyseur lexical et l'analyseur syntaxique gèrent la partie régulière et algébrique de la grammaire d'un langage de programmation[21]. Articles liés Automate à jetons Automate à pile Automate à file Automate cheminant Automate d'arbres Automates finis communicants (en) Automate fini déterministe Automate fini non déterministe Automate pondéré Automate quantique Diagramme états-transitions Discrete Event System Specification État (patron de conception) Intelligence artificielle Langage rationnel Logique séquentielle Machine à états abstraits (ASM) Machine de Turing Model checking Modèle de Markov caché Réseau de Petri SCXML Specification and Description Language Stratégie de régulation Système de transition d'états Table de décision Théorie des automates UML Notes et références ↑ a et b (en) Thomas Koshy, Discrete Mathematics With Applications, Academic Press, 2004, 762 p. (ISBN 978-0-12-421180-3, lire en ligne). ↑ (en) David R.

$$M^0 = \begin{pmatrix} \epsilon & 0 & 0 & 0 \\ 0 & \epsilon & 0 & 0 \\ 0 & 0 & \epsilon & 0 \\ 0 & 0 & 0 & \epsilon \end{pmatrix} \quad M = \begin{pmatrix} 0 & a & 0 & b \\ 0 & 0 & b & a \\ 0 & 0 & a+b & 0 \\ 0 & 0 & 0 & a \end{pmatrix} \quad M^2 = \begin{pmatrix} 0 & 0 & ab & aa+ba \\ 0 & 0 & b(a+b) & a^2 \\ 0 & 0 & (a+b)^2 & 0 \\ 0 & 0 & 0 & a^2 \end{pmatrix}$$

$$M^3 = \begin{pmatrix} 0 & 0 & ab(a+b) & a^3+ba^2 \\ 0 & 0 & b(a+b)^2 & a^3 \\ 0 & 0 & (a+b)^3 & 0 \\ 0 & 0 & 0 & a^3 \end{pmatrix} \quad M^4 = \begin{pmatrix} 0 & 0 & ab(a+b)^2 & a^4+ba^3 \\ 0 & 0 & b(a+b)^3 & a^4 \\ 0 & 0 & (a+b)^4 & 0 \\ 0 & 0 & 0 & a^4 \end{pmatrix}$$

Wright, « Finite State Machines » [PDF], CSC215 Class Notes, North Carolina State University, 2005. ↑ a b c et d Pin, « Automates finis », Encyclopédie de l'informatique et des systèmes d'information. ↑ Ne pas confondre avec machine virtuelle. ↑ UIT : « Z.100 : Langage de description et de spécification ». ↑ Un système réactif est un système qui est en interaction permanente avec son environnement, et dont le rythme est imposé par cet environnement. ↑ (en) Robert M.
Keller, « Classifiers, Acceptors, Transducers, and Sequencers », dans Computer Science : Abstraction to Implementation, Harvey Mudd College, 2001 (lire en ligne), p. 480.
↑ (en) Marc Pouly et Jürg Kohlas, Generic Inference : A Unifying Theory for Automated Reasoning, John Wiley & Sons, 2011, 484 p. (ISBN 978-1-118-01086-0, présentation en ligne), Chapter 6. Valuation Algebras for Path Problems, p. 223 en particulier.
↑ Storer 2012, Chapter 10. ↑ Sipser 2013, p. 34. ↑ (en) Michael Brutscheck, St. Berger, M.
Franke, Andreas Schwarzbacher et St. Becker, « Structural Division Procedure for Efficient IC Analysis », IET Irish Signals and Systems Conference (ISSC 2008), Galway, Irlande, 2008, p. 18-23 (lire en ligne). ↑ (en) Grégoire Hamon, « A Denotational Semantics for Stateflow », ACM, Jersey City, NJ, 2005, p. 164–172 (DOI 10.1145/1086228.1086260).
↑ (en) David Harel, « A Visual Formalism for Complex Systems », Science of Computer Programming, 1987, p. 231–274 (lire en ligne). ↑ (en) R. Alur, A. Kanade, S. Ramesh et K. C. Shashidhar, « Symbolic analysis for improving simulation coverage of Simulink/Stateflow models », International Conference on Embedded Software, Atlanta, 2008, p. 89–98 (lire en ligne). ↑ (en) Paul E. Black, « Finite State Machine », Dictionary of Algorithms and Data Structures, U.S. National Institute of Standards and Technology, 12 mai 2008. ↑ Anderson, 2006, p. 105–108. ↑ Carton 2008. ↑ a et b (en) Jean Berstel, Luc Boasson, Olivier Carton et Isabelle Fagnot, « Minimization of Automata », dans Automata: from Mathematics to Applications, European Mathematical Society, 2010 (arXiv 1010.5318). ↑ (en) Dominique Revuz, « Minimization of Acyclic automata in Linear Time », Theoretical Computer Science, Elsevier, vol. 92, 1992, p. 181–189 (DOI 10.1016/0304-3975(92)90142-3). ↑ (en) Hubert Kaeslin, Digital Integrated Circuit Design : From VLSI Architectures to CMOS Fabrication, Cambridge University Press, 2008, 845 p. (ISBN 978-0-521-88267-5, lire en ligne), « Mealy, Moore, Medvedev-type and combinatorial output bits », p. 787. ↑ Aho et al. 2007. Bibliographie Ouvrages généraux Ferdinand Wagner, Ruedi Schmuki, Thomas Wagner et Peter Wolstenholme, Modeling Software with Finite State Machines : A Practical Approach, Auerbach Publications, 2006, 392 p. (ISBN 978-0-8493-8086-0). Christos G. Cassandras et Stéphane Lafortune, Introduction to Discrete Event Systems, Springer Science & Business Media, 1999, 822 p. (ISBN 978-0-7923-8609-4, présentation en ligne). Timothy Kam, Tiziano Villa, Robert K. Brayton et Alberto Sangiovanni-Vincentelli, Synthesis of Finite State Machines : Functional Optimization, Springer, 1996, 282 p. (ISBN 978-0-7923-9842-4, présentation en ligne) Timothy Kam, Tiziano Villa, Robert K. Brayton et Alberto Sangiovanni-Vincentelli, Synthesis of Finite State Machines : Logic Optimization, Springer, 1997, 381 p. (ISBN 978-0-7923-9892-9) Yishai A. Feldman, « Finite-state Machines », dans Allen Kent et James G. Williams (éditeurs), Encyclopedia of Computer Science and Technology : Volume 25 - Supplement 10, Marcel Dekker, 1992 (ISBN 0-8247-2275-2, lire en ligne), p. 73-104 Jean-Éric Pin, « Automates finis », dans Jacky Akoka et Isabelle Comyn-Wattiau (éditeurs), Encyclopédie de l'informatique et des systèmes d'information, Paris, Vuibert, 2006, xxxv+1941 (ISBN 978-2-7117-4846-4), p. 966-976. James A. Storer, An Introduction to Data Structures and Algorithms, Springer Science & Business Media, 2012, 599 p. (ISBN 978-1-4612-0075-8 et 146120075X, présentation en ligne) Alfred Aho, Monica Lam, Ravi Sethi et Jeffrey Ullman (trad. de l'anglais par Philippe Deschamp, Bernard Lorho, Benoît Sagot et François Thomasset), Compilateurs : Principes, techniques et outils [« Compilers: Principles, Techniques, and Tools »], France, Pearson Education, novembre 2007, 2e éd. (1re éd. 1977), 901 p. (ISBN 978-2-7440-7037-2, présentation en ligne) Aspects pratiques « Z.100 : Langage de description et de spécification - Présentation générale de SDL-2010 », Union internationale des télécommunications, 1er septembre 2015.
Miro Samek, Practical Statecharts in C/C++ : Quantum Programming for Embedded Systems, CRC Press, 2002, 416 p. (ISBN 978-1-57820-110-5, lire en ligne). (en) Miro Samek, Practical UML Statecharts in C/C++ : Event-Driven Programming for Embedded Systems, Amsterdam, Newnes, 2009, 712 p. (ISBN 978-0-7506-8706-5, lire en ligne). Susan H. Rodger et Thomas W. Finley, JFLAP : An Interactive Formal Languages and Automata Package, Jones & Bartlett Learning, 2006, 192 p.
(ISBN 978-0-7637-3834-1, présentation en ligne). Gérard Berry La vérification de modèles (model checking), Leçon du Collège de France du 25 mars 2015. Automates en informatique théorique Olivier Carton, Langages formels, calculabilité et complexité, 2008 [détail de l'édition] (lire en ligne) Sylvain Perifel, Complexité algorithmique, Ellipses, 2014, 432 p. (ISBN 9782729886929, lire en ligne) — (La version électronique est datée de juillet 2015.) Jacques Sakarovitch, Éléments de théorie des automates, Paris, Vuibert, 2003, 816 p. (ISBN 978-2-7117-4807-5, zbMATH 1188.68177) Gérard Berry, L'informatique du temps et des événements : Leçon inaugurale prononcée le jeudi 28 mars 2013, Collège de France, 2013, 88 p. (ISBN 978-2-7226-0277-9) James A. Anderson et Thomas J. Head, Automata Theory with Modern Applications, Cambridge University Press, 2006, 264 p. (ISBN 978-0-521-84887-9 et 0521848873). George S. Boolos, John P. Burgess et Richard C. Jeffrey, Computability and Logic, Cambridge, Cambridge University Press, 2007, 5e éd. (1re éd. 1989), 366 p. (ISBN 978-0-521-70146-4). Martin Davis, Ron Sigal et Elaine J. Weyuker, Computability, Complexity, and Languages : Fundamentals of Theoretical Computer Science, San Diego, Academic Press, 1994, 2e éd., 609 p. (ISBN 978-0-12-206382-4, présentation en ligne). (en) John E. Hopcroft, Rajeev Motwani et Jeffrey D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, 2007, 3e éd. (ISBN 978-0-32146225-1) Harry R. Lewis et Christos H. Papadimitriou, Elements of the Theory of Computation, Prentice Hall, 1998, 2e éd., 361 p. (ISBN 978-0-13-262478-7). Peter Linz, An Introduction to Formal Languages and Automata, Sudbury, MA, Jones & Bartlett Publishers, 2011, 5e éd., 437 p.
(ISBN 978-1-4496-1552-9, présentation en ligne). Sanjeev Arora et Boaz Barak, Computational Complexity : A Modern Approach, Cambridge University Press, 2009, 579 p. (ISBN 978-0-521-42426-4, présentation en ligne). Nicholas Pippenger, Theories of Computability, Cambridge, Cambridge University Press, 2010, 264 p. (ISBN 978-0-521-15343-0)
Michael Sipser, Introduction to the theory of computation, Boston, MA, Cengage Learning, 2013, 3e éd., 480 p. (ISBN 978-1-133-18779-0, OCLC 761858892, présentation en ligne). Articles récents Sankardeep Chakraborty, Roberto Grossi, Kunihiko Sadakane et Srinivasa Rao Satti, « Succinct representation for (non)deterministic finite automata », Journal of Computer and System Sciences, vol. 131, 1er février 2023, p. 1–12 (DOI 10.1016/j.jcss.2022.07.002, lire en ligne) Jason Bell et Daniel Smertnig, « Noncommutative rational Pólya series », Selecta Mathematica, vol. 27, no 3, 19 mai 2021, article no 34 (34 pages) (DOI 10.1007/s00029-021-00629-2, arXiv 1906.07271) Sylvain Lombardy et Jacques Sakarovitch, « The net automaton of a rational expression », LATIN 2022: Theoretical Informatics, Lecture Notes in Computer Science vol. 13568, Springer, 2022, p. 376–392 (ISBN 978-3-031-20624-5, DOI 10.1007/978-3-031-20624-5_23) Machines abstraites en informatique théorique Yuri Gurevich, « Sequential Abstract State Machines Capture Sequential Algorithms », ACM Transactions on Computational Logic, vol. 1, no 1, juillet 2000, p. 77–111 (DOI 10.1145/343369.343384, lire en ligne) Sergey Goncharov, Stefan Milius et Alexandra Silva, « Toward a Uniform Theory of Effectful State Machines », ACM Transactions on Computational Logic, vol. 21, no 3, 2020, article no 23 (63 p.) (ISSN 1529-3785, DOI 10.1145/3372880). Apprentissage automatique Trevor Hastie, Robert Tibshirani et Jerome Friedman, The elements of statistical learning : data mining, inference, and prediction, New York, Springer, 2009, 2e éd., xxii+745 (ISBN 978-0-387-84857-0) Vincent Barra, Antoine Cornuéjols, Laurent Miclet, Apprentissage Artificiel : Concepts et algorithmes, Eyrolles, 2021 (ISBN 978-2-416-001-04-8) [détail des éditions] (en) David MacKay, Information Theory, Inference, and Learning Algorithms, Cambridge University Press, 2003 (ISBN 0-521-64298-1) [détail des éditions] (en) Tom M. Mitchell, Machine Learning, 1997 [détail des éditions] (en) Christopher M. Bishop, Pattern Recognition And Machine Learning, Springer, 2006 (ISBN 0-387-31073-8) [détail des éditions] Liens externes Sur les autres projets Wikimedia : Automate fini, sur Wikimedia Commons Modeling a Simple AI behavior using a Finite State Machine Exemple d'emploi dans les jeux vidéo NIST Dictionary of Algorithms and Data Structures Description of Finite State Machines Interactive FSM: Control Circuit, Démonstration de flot logique dans les automates finis. FSM simulator, Simulation de DFA, NFA et ε-NFA, y compris engendré par des expressions rationnelles. Notices d'autorité : Tchéquie Portail de l'informatique théorique Ce document provient de « .