Algèbres de Relations,
Étude des algèbres de Kleene avec converse

Paul Brunet,
supervisé par Damien Pous dans l'équipe Plume du Lip

22 août 2013

Le contexte général

Dans de nombreux contextes en informatique et en mathématiques les opérations d'union, de séquence ou de produit et d'itération apparaissent naturellement. Les *algèbres de Kleene*, introduites par John H Conway sous le nom d'*algèbres régulières* (voir par exemple [Con71]), proposent une théorie équationnelle permettant d'exprimer les propriétés de ces opérateurs, en étudiant l'équivalence d'expressions formées avec ces connecteurs. Dans ce cadre, on sait depuis longtemps que cette théorie est décidable, et qu'elle est complète pour les modèles de langages et de relations.

Aussi expressive que puisse être cette théorie, on peut avoir envie d'y intégrer d'autres opérations courantes. Les théories que l'on obtient de cette manière sont appelées *Algèbres de Kleene étendues*.

Le problème étudié

On s'est intéressé durant ce stage à l'étude des algèbres avec converse (CKA), qui sont des algèbres de Kleene auxquelles on a ajouté l'opération de converse. Le converse d'un mot est son image miroir (on inverse l'ordre des lettres), et le converse $R^\vee$ d'une relation R est son inverse ($xRy \overset{\Delta}{\iff} yR^\vee x$). Cette opération est très naturelle, et s'exprime simplement comme un ensemble d'équations que l'on ajoute aux axiomes des algèbres de Kleene.

La question à laquelle il faut répondre une fois cette théorie construite est celle de sa décidabilité : étant données deux expressions formelles utilisant les opérations binaires de somme et de produit, et les deux opérations unaires d'itération et de converse, peut-on automatiquement décider leur équivalence, c'est à dire l'existence d'une preuve de leur égalité utilisant les axiomes de CKA. Bloom, Ésik, Stefanescu et Bernátsky ont apporté une réponse affirmative à cette question dans deux articles ([BÉS95] et [ÉB95]) en 1995.

Néanmoins, l'algorithme proposé dans leurs travaux, s'il prouve parfaitement le résultat souhaité, est bien trop compliqué pour pouvoir être utilisé en pratique. Nous avons conçu pendant ce stage un algorithme à la fois plus simple et nettement plus efficace, qui nous a en outre permis de placer ce problème dans la classe de complexité PSpace.

La contribution proposée

Dans l'article [BÉS95], il est établi que l'équivalence de deux expressions pour tous les modèles relationnels se réduit à l'égalité d'une certaine clôture de deux langages réguliers, obtenus à partir des expressions

de départ. Reste donc à calculer cette clôture. Dans l'article original, un construction est donnée, partant d'un automate fini *déterministe*, et calculant un automate qui reconnaît la clôture du langage de départ. Cette construction s'appuie fortement sur le monoïde des transitions de l'automate d'entrée (c'est à dire l'ensemble des transformations de l'ensemble des états de l'automate induites par des mots).

Nous proposons ici une nouvelle construction plus directe de l'automate reconnaissant la clôture, sans passer par des monoïdes et qui accepte en entrée un automate quelconque, déterministe ou non. Nous établissons également sa correction.

Les arguments en faveur de sa validité

On peut analyser notre méthode et celle de [BÉS95] en terme de taille de l'automate produit. Il apparaît que, en prenant en entrée un automate non déterministe de taille n, l'automate que nous produisons est de taille au plus 2^{n^2} alors que la construction originale produit un automate dont la taille est bornée par $2^{2^{n\times 2^n}+1}$.

Nous avons implémenté en OCaml [(i)] les deux algorithmes, et nous avons testé sur un certain nombre d'exemple l'efficacité relative des deux méthodes et les résultats corroborent notre analyse : notre méthode est plus rapide et produit des automates nettement plus petits.

Le bilan et les perspectives

Ce travail s'inscrit dans le cadre d'une tentative d'intégrer à une librairie de Coq [(ii)] manipulant des expressions régulières l'opération de converse, pour avoir une tactique décidant l'égalité de deux relations exprimées par des expressions régulières avec converse. La simplicité de la preuve de correction de notre algorithme nous donne bon espoir quant à la faisabilité d'une telle entreprise.

D'autres extensions des algèbres de Kleene ont également retenu notre attention, en particulier les algèbres d'actions (voir [Pra91]), dont la décidabilité est toujours un problème ouvert.

(i). `http://caml.inria.fr/`
(ii). `http://coq.inria.fr/`

1 Algèbres de Kleene avec converse

Dans une première partie, on va définir les algèbres de Kleene puis les algèbres de Kleene avec converse, et énoncer quelques résultats sur ces théories et leurs modèles.

1.1 Les algèbres de Kleene et la théorie KA

La première notion que l'on va introduire est celle d'algèbre de Kleene, dont le but est de capturer le comportement des expressions régulières, c'est à dire les manières dont elles peuvent s'instancier.

1.1.1 Axiomatisation

Nous allons ici donner une présentation tirée d'un article de Dexter Kozen (voir [Koz94]).

Une algèbre de Kleene est une structure algébrique $\langle K, +, \cdot, ^*, 0, 1\rangle$ telle que $\langle K, +, \cdot, 0, 1\rangle$ est un semi-anneau idempotent, ce qui se traduit en les axiomes suivants :

$$a + (b + c) = (a + b) + c \tag{1}$$
$$a + b = b + a \tag{2}$$
$$a + 0 = a \tag{3}$$
$$a + a = a \tag{4}$$
$$a(bc) = (ab)c \tag{5}$$
$$1a = a \tag{6}$$
$$a1 = a \tag{7}$$
$$a(b + c) = ab + ac \tag{8}$$
$$(a + b)c = ac + bc \tag{9}$$
$$0a = 0 \tag{10}$$
$$a0 = 0. \tag{11}$$

Avec ceci, on peut définir un ordre partiel $a \leqslant b \overset{\Delta}{\iff} a + b = b$.

Par ailleurs, on souhaite que l'opération * satisfasse certaines propriétés. On arrive alors à un problème : on ne peut finiment axiomatiser les propriétés de l'étoile dans le cadre des algèbres de Kleene [(iii)] [Red64]. Différentes possibilités ont été proposées pour pallier à ce problème (notamment des schémas d'axiomes). On donne ici la solution proposée par Kozen, utilisant des implications.

$$1 + aa^* \leqslant a^* \tag{12}$$
$$1 + a^*a \leqslant a^* \tag{13}$$
$$b + ax \leqslant x \Rightarrow a^*b \leqslant x \tag{14}$$
$$b + xa \leqslant x \Rightarrow ba^* \leqslant x \tag{15}$$

La théorie équationnelle KA consiste en l'ensemble des axiomes (1) à (15). Les algèbres de Kleene sont donc des *modèles* de KA. Une *expression régulière* ou *expression rationnelle* sur l'alphabet X est une

(iii). Dans le cadre des algèbres d'actions (voir [Pra91]), en rajoutant deux opérations, adjointes à droite et à gauche au produit, on arrive à une théorie finiment axiomatisable, mais dans ce cas la décidabilité de la théorie reste un problème ouvert.

expression bien formée utilisant les opérateurs $+, \cdot, ^*$, 0 et 1, ainsi qu'un nombre arbitraire de variables libres prises dans X. On notera l'ensemble de ces expressions Reg_X. On peut remarquer que c'est trivialement une algèbre de Kleene (une fois quotienté par la plus petite relation d'équivalence contenant les règles (1) à (15)).

1.1.2 Modèles et résultats

Les modèles de cette théorie sont nombreux (processus, traces...) mais nous ne nous intéresserons ici qu'à deux familles de modèles : les langages et les relations. Ces deux familles ont la particularité d'être initiales, c'est à dire en particulier que ce qui est universellement vrai dans les modèles de langages est prouvable dans KA. Nous allons maintenant détailler cette assertion et donner une idée de sa preuve.

Une interprétation des expressions régulières sur l'alphabet X est un morphisme $\phi : \mathrm{Reg}_X \longrightarrow K$, où $\langle K, +, \cdot, ^*, 0, 1\rangle$ est une algèbre de Kleene. Pour spécifier un tel morphisme, il suffit de le définir sur les lettres. Dans la suite, pour toute application $\sigma : X \longrightarrow K$, on notera $\hat{\sigma} : \mathrm{Reg}_X \longrightarrow K$ l'unique morphisme égal à σ sur X.

Les langages : Pour tout alphabet Σ, l'ensemble des langages sur Σ, $Lang(\Sigma) := \langle \mathscr{P}(\Sigma^*), \cup, \cdot, ^*, \emptyset, \{\epsilon\}\rangle$, forme une algèbre de Kleene, où :

- ϵ est le mot vide ;
- $L \cup M := \{w \mid w \in L \text{ ou } w \in M\}$;
- $L \cdot M := \{w \mid \exists w_1 \in L, \exists w_2 \in M : w = w_1 w_2\}$;
- $L^* := \bigcup_{n\in\mathbb{N}} L^n$, où $L^n = \underbrace{L \cdot \cdots \cdot L}_{n}$.

On notera $e \equiv_{Lang} f$ pour $\forall \Sigma, \forall \sigma : \mathscr{P}(\Sigma^*)^X, \hat{\sigma}(e) = \hat{\sigma}(f)$, c'est à dire $e = f$ est vraie pour toute interprétation dans un modèle de langage.

Soit e une expression rationnelle sur X, on appellera *langage dénoté par e*, et on notera $[\![e]\!]$, l'image de e par le morphisme généré par l'application $\sigma : \begin{array}{ccc} X & \longrightarrow & \mathscr{P}(X^*) \\ a & \longmapsto & \{a\} \end{array}$.

Soient e et f deux expressions régulières sur l'alphabet X. Il est très simple de montrer que

$$\mathrm{KA} \vdash e = f \Rightarrow e \equiv_{Lang} f.$$

C'est juste la propriété "les langages forment un modèle de KA". En revanche le sens inverse est plus subtil, et une preuve très élégante (mais assez technique) en est proposée dans [Koz94].

On prouve en réalité un résultat plus fort :

$$[\![e]\!] = [\![f]\!] \Rightarrow \mathrm{KA} \vdash e = f.$$

Il est en effet immédiat que $e \equiv_{Lang} f \Rightarrow [\![e]\!] = [\![f]\!]$. On sait que l'équivalence de deux langages dénotés par des expressions régulières est décidable, en décidant l'équivalence de deux automates (c'est une conséquence du théorème de Kleene, [Kle51]). Kozen modélise ces automates par des matrices (en considérant la matrice d'incidence pondérée du graphe orienté pondéré qui représente l'automate), et en traduisant un algorithme d'équivalence d'automates en termes matriciels. Comme les matrices à coefficients dans une algèbre de Kleene forment elles-mêmes une algèbre de Kleene, on prouve la correction de l'algorithme en utilisant les axiomes de KA, et on en déduit une preuve dans KA que les deux expressions sont bien égales.

Les relations : Pour tout ensemble S, l'ensemble des relations binaires sur S, $Rel(S) \coloneqq \langle \mathscr{P}(S^2), \cup, \circ, ^*, \emptyset, \mathrm{Id}_S \rangle$, forme une algèbre de Kleene, avec $\cup$ l'union ensembliste, $\circ$ la composition des relations :

$$x(R_1 \circ R_2)y \coloneqq \exists z : xR_1z \wedge zR_2y$$

et * est la clôture réflexive transitive d'une relation. Pour montrer que les relations forment également un modèle initial, on réduit sur les langages. Spécifiquement, on exhibe un modèle relationnel particulier qui permet de se ramener à l'égalité des langages réguliers dénotés par les expressions. Soit e une expression régulière sur X, on construit l'application

$$\begin{array}{rccl} \phi : & X & \longrightarrow & \mathscr{P}(X^* \times X^*) \\ & a & \longmapsto & \{(w, wa) \mid w \in X^*\}. \end{array}$$

Le morphisme qui en découle a une propriété intéressante : $u \in [\![e]\!] \Leftrightarrow \forall w, (w, wu) \in \hat{\phi}(e)$. On en déduit que $\hat{\phi}(e) = \hat{\phi}(f) \Rightarrow [\![e]\!] = [\![f]\!]$ et donc cela implique, par le résultat sur les langages, que $\mathrm{KA} \vdash e = f$.

1.2 Les algèbres avec converse et les théories CKA

On souhaite maintenant augmenter ce cadre avec l'opération de converse. Rappelons-en la définition sur les langages et sur les relations :

- $L^\vee \coloneqq \{w \mid miroir(w) \in L\}$, avec $miroir(\epsilon) = \epsilon$ et $\forall (w, a) \in \Sigma^* \times \Sigma, miroir(w \cdot a) = a \cdot miroir(w)$;
- $R^\vee \coloneqq \{(x, y) \mid (y, x) \in R\}$.

Les questions qui se posent maintenant sont les suivantes :

1. Peut-on encoder ces deux opérations dans la même théorie équationnelle ?
2. Quels axiomes rajouter à KA pour modéliser ces opérations ?
3. Les langages et les relations sont-ils des modèles initiaux pour les théories obtenues ?
4. Comment décider ces théories ?

La première question a une réponse simple : non. En effet l'équation $a \leqslant a \cdot a^\vee \cdot a$ est vérifiée dans tous les modèles relationnels mais pas pour les langages :

- $(x, y) \in R \Rightarrow ((x, y) \in R \wedge (y, x) \in R^\vee \wedge (x, y) \in R) \Rightarrow (x, y) \in R \circ R^\vee \circ R$;
- si on considère $L = \{a\}$, alors $L \cdot L^\vee \cdot L = \{aaa\}$ et $a \notin \{aaa\}$.

Il nous faut donc construire deux théories, correspondant à ces deux familles de modèles.

1.2.1 Modèle des langages : théorie $\mathbf{CKA}_1$

Dans l'article [BÉS95] le théorème suivant est démontré :

1 Théorème:
Une axiomatisation complète de la variété $Lang^\vee$ des langages générés par les opérations de concaténation, union, étoile et converse consiste des axiomes (1) *à* (15) *auxquels on ajoute :*

$$(a + b)^\vee = a^\vee + b^\vee \tag{16}$$
$$(a \cdot b)^\vee = b^\vee \cdot a^\vee \tag{17}$$
$$(a^*)^\vee = (a^\vee)^* \tag{18}$$
$$a^{\vee\vee} = a. \tag{19}$$

■

Par la suite, on appellera CKA_1 cette théorie. Bien qu'une preuve soit donnée dans l'article, on donnera ici une preuve originale, plus intuitive à nos yeux.

On notera $e \equiv_{Lang^\vee} f$ pour $\forall \Sigma, \forall \sigma : \mathscr{P}(\Sigma^*)^X, \hat{\sigma} e = \hat{\sigma} f$.

Preuve On peut vérifier très simplement que toutes ces équations sont satisfaites par tout modèle de langage. Il faut donc montrer que toute équation vérifiée dans tout modèle de langage est prouvable dans CKA_1 . Pour cela, on se ramène à un problème de langages réguliers. On considère deux expressions e et f dans $\mathrm{Reg}_X^\vee$.

– On considère la transformation τ suivante, qui pousse les occurences de l'opérateur $^\vee$ aux feuilles (c'est à dire les variables) des expressions :

$$\begin{array}{l|l} \forall a \in X, \tau(a) \coloneqq a & \forall a \in X, \nu(a) \coloneqq a^\vee \\ \tau(\epsilon) \coloneqq \epsilon & \nu(\epsilon) \coloneqq \epsilon \\ \tau(e_1 \cdot e_2) \coloneqq \tau(e_1) \cdot \tau(e_2) & \nu(e_1 \cdot e_2) \coloneqq \nu(e_2) \cdot \nu(e_1) \\ \tau(e_1 + e_2) \coloneqq \tau(e_1) + \tau(e_2) & \nu(e_1 + e_2) \coloneqq \nu(e_1) + \nu(e_2) \\ \tau(e^*) \coloneqq \tau(e)^* & \nu(e^*) \coloneqq \nu(e)^* \\ \tau(e^\vee) \coloneqq \nu(e) & \nu(e^\vee) \coloneqq \tau(e). \end{array}$$

On peut voir que dans $\tau(e)$, $^\vee$ n'apparaît que sur les lettres. De plus, e et $\tau(e)$ sont prouvablement égales en utilisant les axiomes (16)-(19).

– On ajoute à X une copie primée (disjointe) de lui-même. On peut voir $\tau(e)$ comme une expression régulière sur l'alphabet $X \cup X'$ en remplaçant les $a^\vee$ apparaissant dans $\tau(e)$ par a'. On appellera $\mathbf{e}$ l'expression ainsi obtenue. Il est immédiat [(iv)] de voir que

$$\mathrm{KA} \vdash \mathbf{e} = \mathbf{f} \Rightarrow \mathrm{CKA}_1 \vdash e = f. \tag{20}$$

– Considérons les interprétations $\sigma_1 : \begin{array}{ccc} X & \longrightarrow & \mathscr{P}\left((X \cup \{\bullet\})^*\right) \\ a & \longmapsto & \{a \cdot \bullet\} \end{array}$ et $\sigma_2 : \begin{array}{ccc} X \cup X' & \longrightarrow & \mathscr{P}\left((X \cup \{\bullet\})^*\right) \\ a & \longmapsto & \{a \cdot \bullet\} \\ a' & \longmapsto & \{\bullet \cdot a\} \end{array}$.

On peut montrer par induction sur e que $\hat{\sigma}_1(e) = \hat{\sigma}_2(\mathbf{e})$. De plus, on peut vérifier que $\hat{\sigma}_2$ est injective (modulo égalité des langages dénotés) c'est à dire que $\hat{\sigma}_2(\mathbf{e}) = \hat{\sigma}_2(\mathbf{f}) \Rightarrow [\![\mathbf{e}]\!] = [\![\mathbf{f}]\!]$.

– On déduit de tout cela que

$$(e \equiv_{Lang^\vee} f) \Rightarrow (\hat{\sigma}_1(e) = \hat{\sigma}_1(f)) \Rightarrow (\hat{\sigma}_2(\mathbf{e}) = \hat{\sigma}_2(\mathbf{f})) \Rightarrow ([\![\mathbf{e}]\!] = [\![\mathbf{f}]\!]).$$

Or $\mathbf{e}$ et $\mathbf{f}$ étant des expressions régulières, on sait que $[\![\mathbf{e}]\!] = [\![\mathbf{f}]\!] \Leftrightarrow \mathrm{KA} \vdash \mathbf{e} = \mathbf{f}$, ce qui nous permet de conclure, en utilisant (20).

On a donc bien établi $\forall e, f : \left(e \equiv_{Lang^\vee} f\right) \Leftrightarrow (\mathrm{CKA}_1 \vdash e = f)$. □

On a au passage montré que l'égalité (en terme de preuve de CKA_1) de deux expressions est décidable, puisqu'on l'a ramené de manière calculable à un problème d'égalité de langages réguliers. Le langage canoniquement dénoté par $\mathbf{e}$ resservira par la suite. On l'appellera le langage régulier primé dénoté par l'expression e.

(iv). Il n'y a qu'à prendre la preuve de $\mathrm{KA} \vdash \mathbf{e} = \mathbf{f}$, remplacer partout a' par $a^\vee$ (syntaxiquement), ce qui donne une preuve de $\mathrm{KA} \vdash \tau(e) = \tau(f)$, puis de brancher les preuves $\mathrm{CKA}_1 \vdash e = \tau(e)$ et $\mathrm{CKA}_1 \vdash f = \tau(f)$ pour obtenir la preuve voulue.

1.2.2 Modèles relationnels : théorie CKA_2

On a dit plus haut qu'on pouvait trouver une différence simple entre les langages et les relations du point de vue de la converse : l'inéquation

$$a \leqslant aa^{\vee}a. \tag{21}$$

Et il s'avère que c'est tout ! En effet dans [ÉB95] le théorème suivant est démontré :

2 Théorème:
Une axiomatisation complète de la variété $Rel^{\vee}$ des relations générées par les opérations de concaténation, union, étoile et converse consiste des axiomes (1)-(19)*, auxquels on ajoute l'inéquation* (21)*.* ■

On appellera CKA_2 cette théorie. On ne donnera pas ici la preuve de ce résultat, qui est assez technique. En revanche on va s'intéresser à la méthode permettant de décider si deux expressions sont égales pour tous modèles relationels.

On a vu plus haut que pour décider l'équivalence dans la variété des langages, on passe par le langage régulier **e**, sur l'alphabet primé. Le théorème 2 affirme que la seule différence entre le cas des langages et celui des relations est l'inéquation $a \leqslant aa^{\vee}a$. On va donc définir une opération $\bar{\cdot}$ (correspondant à $^{\vee}$) sur les mots de l'alphabet primé, puis on va clore **e** par une relation correspondant à l'inéquation (21). On considère des expressions sur un alphabet X, et on pose $\mathbf{X} := X \cup X'$, où X' est une copie primée (disjointe) de X.

Définition ($\overline{w}$)
Pour tout mot $w \in \mathbf{X}^*$, on définit inductivement : $\overline{\epsilon} = \epsilon$ et pour tous $x \in X$ et $w \in \mathbf{X}^*$:

$$\begin{cases} \overline{xw} = \overline{w}x' \\ \overline{x'w} = \overline{w}x. \end{cases}$$ *

Définition ($u \rightsquigarrow v$)
La relation de réduction $\rightsquigarrow$ est définie par $u \rightsquigarrow v := \exists u_1, u_2, w \in \mathbf{X}^* : u = u_1 w \overline{w} w u_2$ et $v = u_1 w u_2$.

On appelle *clôture* d'un ensemble L de mots sur $\mathbf{X}$ l'ensemble $\mathfrak{cl}(L) := \{v \mid \exists u \in L : u \rightsquigarrow^* v\}$, c'est à dire le plus petit ensemble E contenant L tel que tout mot s'obtenant en une réduction à partir d'un mot de E est dans E. *

En parallèle [(v)] de la présente étude, nous nous sommes posé la question de la confluence de cette relation. Nous avons pu montré qu'elle avait effectivement cette propriété, et nous avons même prouvé en Coq la propriété suivante :

$$\forall u, v_1, v_2, u \rightsquigarrow v_1 \text{ et } u \rightsquigarrow v_2 \Rightarrow \exists v : v_1 \rightsquigarrow^{\leqslant 4} v \text{ et } v_2 \rightsquigarrow^{\leqslant 4} v.$$

La preuve de ce résultat est très technique, et nécessite une étude de cas assez fastidieuse [(vi)]. En annexe B est donnée une version détaillée de cette preuve.

On peut maintenant énoncer la propriété montrée dans [BÉS95] :

(v). Même si cette question n'avait pas d'impact direct sur ce qu'on présente ici, il nous paraissait naturel de l'étudier.

(vi). Ce qui a été une motivation pour faire la preuve en Coq : on a pu en automatiser une grande partie et le système nous assure que l'on n'a négligé aucun cas

Propriété 1: *Deux expressions sont équivalentes pour tous modèles relationnels si et seulement si la clôture des langages réguliers primés qu'elles dénotent sont égales.* ■

PREUVE – Le sens retour est assez simple. Commençons par remarquer que pour toute substitution σ,

$$\hat{\sigma}(e) = \bigcup_{w \in [\![e]\!]} \hat{\sigma}(w).$$

La preuve par induction de ce résultat est immédiate. On peut aussi associer à tout $\sigma : X \longrightarrow \mathscr{P}(S^2)$ un $\sigma' := \begin{bmatrix} x \mapsto \sigma(x) & x \in X \\ x' \mapsto \sigma(x)^\vee & x \in X \end{bmatrix}$. Il est simple de montrer, à nouveau par induction que $\hat{\sigma}(e) = \hat{\sigma'}(\mathbf{e})$. Cela nous donne donc (en se rappelant que $[\![e]\!] = [\![\mathbf{e}]\!]_{[\forall x, x' \mapsto x^\vee]}$) :

$$\hat{\sigma}(e) = \bigcup_{w \in [\![\mathbf{e}]\!]} \hat{\sigma'}(w).$$

Il ne reste qu'à se rendre compte que $u \rightsquigarrow v \Rightarrow \hat{\sigma'}(v) \subseteq \hat{\sigma'}(u)$, et on obtient que

$$\hat{\sigma}(e) = \bigcup_{w \in \mathfrak{cl}([\![\mathbf{e}]\!])} \hat{\sigma'}(w).$$

Il est dès lors évident que si $\mathfrak{cl}([\![\mathbf{e}]\!]) = \mathfrak{cl}([\![\mathbf{f}]\!])$ alors $\hat{\sigma}(e) = \hat{\sigma}(f)$.

– Le sens direct est en revanche plus délicat. La preuve que l'on donne ici est une reformulation de celle qui apparaît dans [BÉS95]. On va avoir besoin d'introduire une famille de morphismes ϕ_u définis comme suit, pour tout mot $u = u_1 \cdots u_n$, avec $\forall i, u_i \in \mathbf{X}$:

$$\forall x \in \mathbf{X}, \phi_u(x) := \{(i-1, i) \mid u_i = x\} \cup \{(i, i-1) \mid u_i = \overline{x}\}.$$

ϕ_u définit donc un morphisme $\hat{\phi_u}$ des expressions sur $\mathbf{X}$ vers les relations binaires sur les entiers $\{0 \ldots n\}$.

On fixe $u = u_1 \cdots u_n$. On doit maintenant prouver que

$$v \rightsquigarrow^* u \Leftrightarrow (0, n) \in \hat{\phi_u}(v). \tag{22}$$

La preuve donnée dans [BÉS95] ne nous satisfaisant pas entièrement, nous en donnons une nouvelle, que nous avons laissé en annexe A.1, faute de place. Elle repose sur la construction d'un automate reconnaissant les mots v tels que $(0, n) \in \hat{\phi_u}(v)$, pour lequel on prouve que le langage qu'il reconnaît est $\{v \mid v \rightsquigarrow^* u\}$.

Cette caractérisation en main, on peut conclure : si les deux expressions sont équivalentes pour tout modèle relationnel, alors en particulier $\forall u, \phi_u([\![\mathbf{e}]\!]) = \phi_u([\![\mathbf{f}]\!])$. Donc :

$$\begin{aligned}
\mathfrak{cl}([\![\mathbf{e}]\!]) &= \{u \mid \exists v \in [\![\mathbf{e}]\!] : v \rightsquigarrow^* u\} && \text{(Définition)} \\
&= \{u \mid \exists v \in [\![\mathbf{e}]\!] : (0, n) \in \hat{\phi_u}(v)\} && \text{(Propriété (22))} \\
&= \{u \mid (0, n) \in \hat{\phi_u}([\![\mathbf{e}]\!])\} && (\hat{\phi_u}(X) = \textstyle\bigcup_{x \in X} \hat{\phi_u}(x)) \\
&= \{u \mid (0, n) \in \hat{\phi_u}([\![\mathbf{f}]\!])\} && \text{(Remarque ci-dessus)} \\
&= \mathfrak{cl}([\![\mathbf{f}]\!]). && \text{(Par symétrie)}
\end{aligned}$$

□

On voit maintenant que pour décider l'équivalence de deux expressions dans CKA_2 , il suffit de savoir calculer la clôture d'un langage régulier.

2 Clôture d'un langage régulier : les deux méthodes

Le problème qu'on se pose ici est simple : étant données deux expressions régulières, leurs clôtures sont-elles égales ? Pour résoudre ce problème, l'approche que l'on suivra ici est celle proposée par Bloom Ésik et Stefanescu. Elle consiste à prendre un automate reconnaissant le langage de départ L, puis de lui appliquer un certain algorithme pour obtenir un nouvel automate, reconnaissant la clôture de L. Pour comparer deux clôtures, il suffit donc d'appliquer cette méthode aux deux expressions, puis de comparer les automates obtenus.

On va commencer par présenter rapidement l'algorithme proposé dans [BÉS95], comme point de comparaison, puis on exposera notre construction et ses avantages en plus de détails, en donnant une idée de sa preuve de correction.

Notations

Dans cette partie, on considérera toujours des mots et des langages sur l'alphabet $\mathbf{X} = X \cup X'$ (comme défini plus haut), avec l'opération $\bar{\cdot}$ et la relation $\rightsquigarrow$ définis comme ci-dessus. De même la clôture $\mathfrak{cl}(L)$ d'un langage L sera $\{v \mid \exists u \in L : u \rightsquigarrow^* v\}$, c'est à dire le plus petit ensemble contenant L et clos par $\rightsquigarrow$ (*i.e.* $\forall u \in \mathfrak{cl}(L), u \rightsquigarrow v \Rightarrow v \in \mathfrak{cl}(L)$).

Un automate déterministe sera donné comme un tuple $\langle Q, A, q_0, T, \delta\rangle$, où
– Q est l'ensemble (fini) des états ;
– A est l'alphabet ;
– $q_0 \in Q$ est l'état initial ;
– $T \subseteq Q$ est l'ensemble des états finaux ;
– $\delta : Q \times A \longrightarrow Q$ est la fonction de transition.

Un automate non-déterministe sera donné comme un tuple $\langle Q, A, I, T, \Delta\rangle$, où
– Q, A et T sont définis comme pour un automate déterministe ;
– $I \subseteq Q$ est l'ensemble des états initiaux ;
– $\Delta \subseteq Q \times A \times Q$ est l'ensemble des transitions.

On notera $L(\mathscr{A})$ le *langage reconnu par l'automate* $\mathscr{A}$.

On utilisera également la notation compacte $p \xrightarrow[\mathscr{A}]{w} q$ pour exprimer qu'il existe dans l'automate $\mathscr{A}$ un chemin étiqueté par le mot w allant de l'état p à l'état q.

Pour tout mot w, $\text{suffixes}(w) \coloneqq \{v \mid \exists u : uv = w\}$ est l'ensemble des suffixes de w.

2.1 La construction originale

La construction proposée dans l'article utilise un monoïde reconnaissant le langage (notion introduite par Rabin et Scott [RS59]) pour construire l'automate de clôture. Rappelons quelques définitions :

Définition (Reconnaissance par monoïde)
Soient un langage $L \subseteq \mathbf{X}^*$ et un monoïde $\langle M, \cdot, \mathbb{1}\rangle$, on dit que L *est reconnu par* M s'il existe un ensemble $P \subseteq M$ et un morphisme de monoïde $\eta : \mathbf{X}^* \longrightarrow M$, tels que :

$$u \in L \Leftrightarrow \eta(u) \in P. \qquad *$$

Si M est fini, on peut définir une étoile sur les parties de M.

Définition (Produit et étoile sur les parties d'un monoïde fini)
On prend $A, B \subseteq M$:
- $A \cdot B \coloneqq \{a \cdot b \mid a \in A, b \in B\}$
- $A^0 \coloneqq \{\mathbb{1}\}$ et $\forall n \in \mathbb{N}, A^{n+1} \coloneqq A \cdot A^n$
- et finalement :

$$A^* \coloneqq \bigcup_{n\in\mathbb{N}} A^n = \lim_{n\to+\infty} (\{\mathbb{1}\} \cup A)^n .$$

*

Comme M est fini, toutes ses parties sont finies et en nombre fini, donc pour tout A la suite croissante [(vii)] $((\{\mathbb{1}\} \cup A)^n)_{n\in\mathbb{N}}$ devient stationnaire à partir d'un certain rang : on a donc un moyen de calculer effectivement l'étoile d'une partie de M.

On définit alors un automate calculant la clôture de la manière suivante :

3 Théorème (Automate clôture de [BÉS95]):
Soit $L \subseteq \mathbf{X}^*$ un langage rationnel, reconnu par $\langle M, \cdot, \mathbb{1}\rangle$ fini, avec P et η. On notera $\lfloor x \rfloor \coloneqq \{\eta(x)\}$ pour plus de clarté. Alors l'automate déterministe

$$\mathscr{B} = \langle \mathscr{P}(M) \times \mathscr{P}(M), \mathbf{X}, (\{\mathbb{1}\}, \{\mathbb{1}\}), T, \delta\rangle$$

reconnaît la clôture de L, avec :
- $T \coloneqq \{(F, G) \mid F \cap P \neq \emptyset\}$
- $\delta((F, G), x) \coloneqq \left(F \cdot \lfloor x \rfloor \cdot \left((\lfloor \overline{x} \rfloor \cdot G \cdot \lfloor x \rfloor)^*\right), (\lfloor \overline{x} \rfloor \cdot G \cdot \lfloor x \rfloor)^*\right)$. ■

On peut simplifier (légèrement) cette définition en considérant dans un premier temps ce qu'on va appeler "le G-automate", qui gère le second membre des transitions ci-dessus :

Définition (G-Automate)
Soit un langage L, reconnu par $\langle M, \cdot, \mathbb{1}\rangle$ fini, avec P et η. On notera à nouveau $\lfloor x \rfloor \coloneqq \{\eta(x)\}$. Son *G-automate* est un automate déterministe défini par

$$G(L) \coloneqq \langle \mathscr{P}(M), \mathbf{X}, \{\mathbb{1}\}, \emptyset\ ^{(viii)}, \delta_G\rangle$$

avec :

$$\delta_G(G, x) \coloneqq (\lfloor \overline{x} \rfloor \cdot G \cdot \lfloor x \rfloor)^* .$$

*

Avec cet automate, et plus spécifiquement avec la fonction δ_G, on peut reformuler la définition de δ en :

$$\delta((F, G), x) = (F \cdot \lfloor x \rfloor \cdot \delta_G(G, x)\ ,\ \delta_G(G, x)).$$

Le $F \cdot \lfloor x \rfloor$ correspond à lire une lettre, et les états de l'automate G correspondent à des "sauts" que l'on peut effectuer. L'intuition de ces sauts est donnée par la fonction Γ définie plus loin pour justifier notre construction.

(vii). Car $(\{\mathbb{1}\} \cup A)^{n+1} = (\{\mathbb{1}\} \cup A) \cdot (\{\mathbb{1}\} \cup A)^n = (\{\mathbb{1}\} \cup A)^n \cup A \cdot (\{\mathbb{1}\} \cup A)^n \supseteq (\{\mathbb{1}\} \cup A)^n$.
(viii). On a pris ici arbitrairement un ensemble d'états finaux vide, car on s'intéresse uniquement aux transitions, pas au langage reconnu.

Dans tout ce qui a été dit ci-dessus, on n'a pas précisé comment on trouvait le monoïde reconnaissant le langage que l'on utilise. Or, pour avoir un vrai algorithme calculant la clôture, il faut une méthode pour obtenir un monoïde fini reconnaissant le langage à partir d'un automate (puisqu'on a choisi comme *input* un automate). Plusieurs choix sont possibles, utilisant tous le *monoïde des transitions* :

Définition (Monoïde des transitions)
Soit $\mathscr{A} = \langle Q, A, q_0, T, \delta\rangle$ un automate complet [ix] déterministe, chaque mot $u \in \mathbf{X}^*$ induit une fonction partielle $u_{\mathscr{A}} : Q \to Q$ qui a un état q associe l'état q' que l'on obtient en suivant l'unique chemin partant de q et étiqueté par u.

Le *monoïde des transitions* de $\mathscr{A}$, noté $M_{\mathscr{A}}$, est l'ensemble des fonctions $Q \to Q$ induites par des mots de $\mathbf{X}^*$. *

En prenant :
- $P \coloneqq \{f \in M_{\mathscr{A}} \mid f(q_0) \in T\}$
- et $\eta \coloneqq [u \mapsto u_{\mathscr{A}}]$,

on remarque que $\langle M_{\mathscr{A}}, \circ, \epsilon_{\mathscr{A}}\rangle$ reconnaît le même langage que $\mathscr{A}$. Q étant fini, il existe un nombre fini de fonctions $Q \to Q$, et donc $M_{\mathscr{A}}$ est fini. Ce monoïde peut être calculé assez simplement par un algorithme en O(*taille du monoïde* $\times$ *nombre d'états de l'automate*).

Dans [BÉS95], c'est le monoïde des transitions [x] qui est utilisé. Mais dans notre implémentation de cet algorithme, on a utilisé le *monoïde syntaxique*, qui est le monoïde des transitions de l'automate minimal reconnaissant le langage. Le monoïde syntaxique a également la propriété d'être le plus petit monoïde (en terme de taille) reconnaissant un langage donné. Nous avons choisi d'utiliser ce monoïde car, puisque qu'on s'intéresse aux couples de parties du monoïde, on a tout intérêt à faire en sorte d'avoir le moins possible d'éléments dans ledit monoïde. En contrepartie il faut minimiser l'automate qu'on prend en entrée.

2.2 Analyse et intuitions

Oublions un instant la construction ci-dessus, et cherchons à construire un *automate clôture*. Pour cela, notre première idée est d'ajouter des transitions sur l'automate de départ. C'est idée vient naturellement lorsque l'on réalise que si $u \rightsquigarrow^* v$, alors v s'obtient en effaçant certaines lettres de u [xi]. Sur un exemple simple :

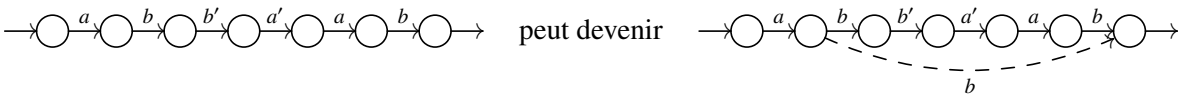


FIGURE 1 – $\mathfrak{cl}(\{abb'a'ab\})$

car on détecte le motif $ab\overline{ab}ab$, et on le saute en lisant la dernière lettre du premier tiers du motif, c'est à dire le b en seconde position.

Malheureusement, cette approche est trop simple et se heurte assez rapidement à des problèmes : en procédant ainsi on reconnaît trop de mots. Par exemple, si on modifie légèrement l'exemple ci-dessus :

(ix). On peut facilement définir le monoïde des transitions d'un automate incomplet, mais pour la clarté de l'exposé on a donné ici la définition la plus simple. Dans la pratique, on se ramène à un automate complet en ajoutant systématiquement un état puits pour "rattraper" les transitions manquantes.

(x). de l'automate reconnaissant L qu'on donne en entrée

(xi). En effet, à chaque étape de réduction, on ne fait qu'effacer des motifs $\overline{w}w$

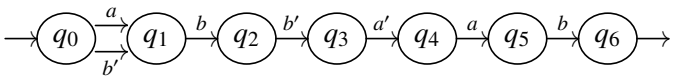


FIGURE 2 – $(a + b') \cdot bb'a'ab$

Avec notre méthode, on rajoute deux transitions, correspondant aux motifs $b'\overline{b'}b'$ et $ab\overline{ab}ab$, ce qui donne :

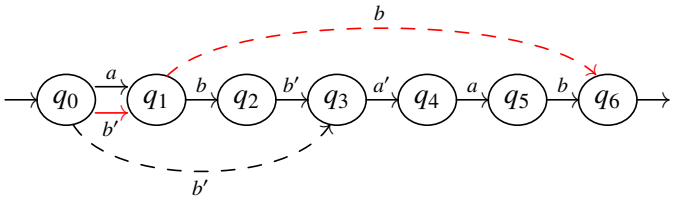


FIGURE 3 – Tentative d'automate pour $\mathfrak{cl}((a + b') \cdot bb'a'ab)$

Le problème avec cet automate est qu'il accepte le mot $b'b$, qui n'est pas dans la clôture du langage initial. En effet $\mathfrak{cl}((a + b') \cdot bb'a'ab) = abb'a'ab + ab + b'bb'a'ab + b'a'ab$. Ce qui se passe est que l'on prend l'arrête b, prévue pour le motif ab, sans avoir au préalable lu un a.

Une manière de résoudre ce problème est d'introduire une notion d'historique, permettant de différencier, sur l'automate précédent, l'état q_1 après avoir lu a et l'état q_1 après avoir lu b'. Ainsi, un état de l'automate clôture sera un couple dont le premier élément indique où l'on se trouve dans l'automate initial, et le second, *l'historique*, retient ce qu'on a déjà lu, et par là même quelles transitions supplémentaires sont autorisées. Sur l'exemple de la figure 2, cela donne l'automate donné en figure 4. On n'a représenté dans les états de

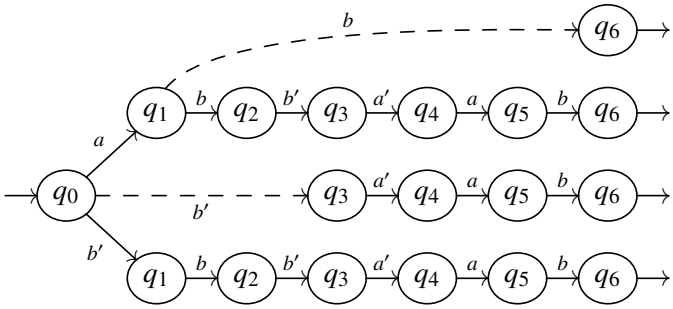


FIGURE 4 – $\mathfrak{cl}((a + b') \cdot bb'a'ab)$

cet automate que la première moitié du couple, vu qu'on n'a encore pas dit ce qu'allait être l'historique.

Dans la construction donnée dans [BÉS95], l'historique est géré par le G-automate, qui grâce à la finitude du monoïde reconnaissant le langage, permet de calculer à partir de ce qu'on a déjà lu quels motifs on peut sauter. Quant à la première moitié du couple, ils ont choisi d'utiliser aussi le monoïde, pour pouvoir simplement faire interagir les deux parties. Nous avons pour notre part choisi de prendre des couples composés d'un état de l'automate de départ et d'un mot, qui représente un ensemble d'historiques permettant les mêmes sauts.

2.3 Notre construction

La construction que nous proposons fonctionne sans passer par le monoïde syntaxique, directement à partir de l'automate initial, a priori non déterministe. Le principe est de prendre comme états des couples, le premier membre correspondant à une exécution de l'automate de départ et le second à un historique (similaire au G-automate), permettant de déterminer sur quels états on peut "sauter". On commence par s'occuper de notre version du G-automate :

Définition ($\Gamma(w)$)
On définit inductivement le langage $\Gamma(w)$ pour tout w :
- $\Gamma(\epsilon) := \{\epsilon\}$
- $\forall x \in \mathbf{X}, \forall w \in \mathbf{X}^*, \Gamma(wx) := (\overline{x}\Gamma(w)x)^*$. *

La propriété suivante illustre l'intérêt de ces langages :

Propriété 2 (Γ est une clôture supérieure): $\forall w, \Gamma(w) = \{u \mid \exists w_1, w_2 : w = w_1 w_2 \text{ et } u \rightsquigarrow^* \overline{w}_2 w_2\}$.
Autrement dit, $u \in \Gamma(w) \Leftrightarrow \exists v \in \textit{suffixes}(w) : u \rightsquigarrow^* \overline{v}v$. ■

On peut comprendre cette propriété comme : "les mots de $\Gamma(w)$ se réduisent en les deux derniers tiers d'un motif compatible avec w".

Preuve La preuve se fait en deux temps. On a d'abord besoin de montrer que :

$$\forall u \in \Gamma(w), \exists v \in \text{suffixes}(w) : u \rightsquigarrow^* \overline{v}v. \tag{23}$$

Ce résultat se montre par induction sur w sans trop de difficultés, mais la preuve étant un peu longue elle a été mise en annexe A.2. On a donc $\Gamma(w) \subseteq \{u \mid \exists v \in \text{suffixes}(w) : u \rightsquigarrow^* \overline{v}v\}$.

On peut voir assez simplement que $\{\overline{v}v \mid v \in \text{suffixes}(w)\} \subseteq \Gamma(w)$, en réalisant que si on note $\Gamma'(w) := \{\overline{v}v \mid v \in \text{suffixes}(w)\}$, alors on peut construire inductivement ce langage avec :
- $\Gamma'(\epsilon) = \epsilon$
- $\Gamma'(wx) = \epsilon + x'\Gamma'(w)x$.

Et donc, par induction sur w :
- $\Gamma'(\epsilon) \subseteq \Gamma(\epsilon)$
- $\epsilon \subseteq \Gamma(wx)$ et par induction $x'\Gamma'(w)x \subseteq x'\Gamma(w)x \subseteq (x'\Gamma(w)x)^* = \Gamma(wx)$.

Le dernier élément de la preuve est que $\Gamma(w)$ est clos supérieurement par $\rightsquigarrow$. Cela se voit en considérant un automate reconnaissant ce langage. On affirme que $\Gamma(x_n \cdots x_1)$ est reconnu par l'automate $\mathscr{G}(x_n \cdots x_1)$ suivant :

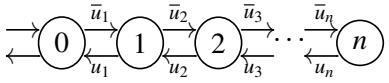


Figure 5 – Automate $\mathscr{G}(x_n \cdots x_1)$ reconnaissant $\Gamma(u_n \cdots u_1)$

On s'en convainc assez simplement, encore une fois par induction. On remarque ensuite que dans cet automate, si $q_1 \xrightarrow[\mathscr{G}(w)]{x} q_2$, alors $q_2 \xrightarrow[\mathscr{G}(w)]{x'} q_1$. Et donc pour tout mot u, si $q_1 \xrightarrow[\mathscr{G}(w)]{u} q_2$, alors $q_2 \xrightarrow[\mathscr{G}(w)]{\overline{u}} q_1$.

Par conséquent si $u_1 w u_2 \in \Gamma(v)$, alors par définition de l'automate on a $0 \xrightarrow[\mathscr{G}(v)]{u_1} q_1 \xrightarrow[\mathscr{G}(v)]{w} q_2 \xrightarrow[\mathscr{G}(v)]{u_2} 0$, et donc

$$0 \xrightarrow[\mathscr{G}(v)]{u_1} q_1 \xrightarrow[\mathscr{G}(v)]{w} q_2 \xrightarrow[\mathscr{G}(v)]{\overline{w}} q_1 \xrightarrow[\mathscr{G}(v)]{w} q_2 \xrightarrow[\mathscr{G}(v)]{u_2} 0$$

ce qui signifie que $u_1 w \overline{w} w u_2 \in \Gamma(x_n \cdots x_1)$.

Autrement dit, pour tout w, pour tout $u \in \Gamma(w)$, pour tout mot v, si $v \rightsquigarrow u$ alors $v \in \Gamma(w)$, ce qui dit exactement que $\Gamma(w)$ est clos supérieurement par $\rightsquigarrow$.

Donc par double inclusion, on a bien montré le résultat souhaité. □

On a donc une caractérisation des mots "permettant des sauts", étant donné un historique. Il nous faut maintenant projeter cela sur l'automate. On suppose maintenant que l'on travaille sur un automate non-déterministe $\mathscr{A} = \langle Q, \mathbf{X}, I, F, \Delta \rangle$ reconnaissant le langage L.

Définition (γ)
Pour tout mot $w \in \mathbf{X}^*$ on définit inductivement la relation $\gamma(w)$ entre des états de l'automate :
- $\gamma(\epsilon) \coloneqq \mathrm{Id}_Q$;
- $\gamma(wx) = (\Delta_{x'} \circ \gamma(w) \circ \Delta_x)^*$.

Où $\Delta_x \coloneqq \{(q_1, q_2) \mid (q_1, x, q_2) \in \Delta\}$. *

On peut tout de suite remarquer un lien fort entre γ et Γ :

Propriété 3: $\forall w, q_1, q_2, ((q_1, q_2) \in \gamma(w)) \Leftrightarrow \left(\exists u \in \Gamma(w) : q_1 \xrightarrow[\mathscr{A}]{u} q_2\right)$. ■

Ce résultat se prouve trivialement par induction. On peut également voir que $\gamma(w) = \hat{\sigma}(\Gamma(w))$ avec $\sigma(x) = \Delta_x$.

En combinant les propriétés 2 et 3, on obtient :

$$\forall w, q_1, q_2, ((q_1, q_2) \in \gamma(w)) \Leftrightarrow \left(\exists u : q_1 \xrightarrow[\mathscr{A}]{u} q_2 \text{ et } u \rightsquigarrow^* \overline{v} v, \text{ avec } v \in \text{suffixes}(w)\right).$$

Le nombre d'états d'un automate étant fini, γ ne peut prendre qu'un nombre fini de valeurs, et par conséquent on peut définir l'ensemble fini suivant :

Définition (G)
Considérons la relation d'équivalence $\sim_\gamma$ sur les mots : $u \sim_\gamma v \coloneqq \gamma(u) = \gamma(v)$. G est défini comme $\mathbf{X}^*$ quotienté par $\sim_\gamma$, c'est à dire l'ensemble des classes d'équivalence de $\sim_\gamma$.On notera $[w]$ les éléments de G, de telle manière que $[u] = [v] \Leftrightarrow u \sim_\gamma v \Leftrightarrow \gamma(u) = \gamma(v)$. On peut ordonner cet ensemble, avec une relation $\preccurlyeq$ définie par $[u] \preccurlyeq [v] \coloneqq \gamma(u) \subseteq \gamma(v)$. *

L'ordre $\preccurlyeq$ s'avère assez pertinent car on peut montrer que

$$u \rightsquigarrow^* v \Rightarrow [u] \preccurlyeq [v]. \tag{24}$$

Preuve Tout d'abord on remarque que $\Gamma(u) \subseteq \Gamma(v) \Rightarrow [u] \preccurlyeq [v]$, en utilisant la propriété 3. Il faut donc montrer $u \rightsquigarrow v \Rightarrow \Gamma(u) \subseteq \Gamma(v)$, c'est à dire $\Gamma(u_1 w \overline{w} w u_2) \subseteq \Gamma(u_1 w u_2)$. On peut se débarrasser de u_2 (il est immédiat de se rendre compte que $\Gamma(w_1) \subseteq \Gamma(w_2) \Rightarrow \forall x \in \mathbf{X}, \Gamma(w_1 x) \subseteq \Gamma(w_2 x)$, à partir de la définition de Γ) : on se ramène donc à montrer $\Gamma(u_1 w \overline{w} w) \subseteq \Gamma(u_1 w)$. En utilisant le fait que $\Gamma(u_1 w)$ est clos supérieurement, il n'y a qu'à montrer que $\forall u \in \Gamma(u_1 w \overline{w} w), u \rightsquigarrow^* v \in \Gamma(u_1 w)$, ce qui se fait en utilisant la caractérisation de la propriété 2, et en faisant une étude de cas sur la taille du suffixe s de $u_1 w \overline{w} w$ tel que $u \rightsquigarrow^* \overline{s} s$. □

On a maintenant tous les outils nécessaires pour construire la clôture de $\mathscr{A}$:

4 Théorème (Automate clôture):
La clôture du langage L est reconnue par l'automate $\mathscr{A}' := \langle Q \times G, \mathbf{X}, I \times \{[\epsilon]\}, F \times G, \Delta' \rangle$ *avec :*

$$\Delta' = \{((q_1, [w]), x, (q_2, [wx])) \mid (q_1, q_2) \in \Delta_x \circ \gamma(wx)\}. \qquad \blacksquare$$

On notera L' le langage reconnu par cet automate. Il faut lire l'ensemble des transitions comme "étant dans l'état q_1 avec historique w, je fais un pas x de l'automate $\mathscr{A}$ puis un saut compatible avec wx, qui devient mon nouvel historique".

On peut voir, à partir de la définition de Δ' et de la propriété 3, que pour tous q_1, q_2, u, x :

$$(q_1, [u]) \xrightarrow[\mathscr{A}']{x} (q_2, [ux]) \Leftrightarrow \left(\exists (q_3, v) \in Q \times \Gamma(ux) : q_1 \xrightarrow[\mathscr{A}]{x} q_3 \xrightarrow[\mathscr{A}]{v} q_2\right). \tag{25}$$

Rappelons ce qu'est une *simulation* :

Définition (simulation)
La relation R sur les états d'un automate $\mathscr{A} = \langle Q, \mathbf{X}, I, F, \Delta \rangle$ est une *simulation* si

$$\forall p, q \in Q, pRq \Rightarrow \left(\forall r \in Q, \forall x \in \mathbf{X}, \left(p \xrightarrow[\mathscr{A}]{x} r\right) \Rightarrow \left(\exists s \in Q : q \xrightarrow[\mathscr{A}]{x} s \wedge rRs\right)\right).$$

On peut écrire cela sous forme d'un diagramme :

$$\begin{array}{ccc} p & \text{—}R\text{—} & q \\ {\scriptstyle x}\downarrow & & \vdots\,{\scriptstyle x} \\ r & \text{- -}R\text{- -} & s \end{array} \qquad *$$

On peut maintenant donner un sens nouveau à l'ordre $\preccurlyeq$:

$$u \preccurlyeq v \Rightarrow \left(\forall q, x, q' : \left((q, [u]) \xrightarrow[\mathscr{A}']{x} (q', [ux])\right) \Rightarrow \left((q, [v]) \xrightarrow[\mathscr{A}']{x} (q', [vx])\right)\right).$$

En effet, $(q, [u]) \xrightarrow[\mathscr{A}']{x} (q', [ux])$ signifie, d'après la définition ci-dessus, que $(q, q') \in \Delta_x \circ \gamma(ux)$. Or on sait que $\gamma(u) \subseteq \gamma(v)$, et donc $\gamma(ux) = (\Delta_{x'} \circ \gamma(u) \circ \Delta_x)^* \subseteq (\Delta_{x'} \circ \gamma(v) \circ \Delta_x)^* = \gamma(vx)$, donc $(q, q') \in \Delta_x \circ \gamma(vx)$ ce qui veut dire que $(q, [v]) \xrightarrow[\mathscr{A}']{x} (q', [vx])$. Si on note $\preccurlyeq$, par abus de notation, l'ordre sur $Q \times G$ défini par l'identité sur la première composante et $\preccurlyeq$ sur la deuxième, on a le diagramme suivant :

$$\begin{array}{ccc} (q_1, [u]) & \text{———}\preccurlyeq\text{———} & (q_2, [v]) \\ {\scriptstyle x}\downarrow & & \vdots\,{\scriptstyle x} \\ (q_3, [ux]) & \text{- - -}\preccurlyeq\text{- - -} & (q_4, [vx]) \end{array}$$

puisque $[u] \preccurlyeq [v]$ implique $[ux] \preccurlyeq [vx]$. On peut donc poser la propriété suivante :

Propriété 4: *$\preccurlyeq$ est une simulation pour l'automate $\mathscr{A}'$.* ■

Prouvons maintenant la correction de notre construction.

Lemme i:
$L' \subseteq \mathfrak{cl}(L)$ ■

Preuve On montre $\forall (q_0, q, u) \in I \times Q \times \mathbf{X}^*, (q_0, [\epsilon]) \xrightarrow[\mathscr{A}']{u} (q, [u]) \Rightarrow (\exists v : v \rightsquigarrow^* u \wedge q_0 \xrightarrow[\mathscr{A}]{v} q)$, par induction sur u (le cas $u = \epsilon$ étant trivial).

Si $(q_0, [\epsilon]) \xrightarrow[\mathscr{A}']{u} (q_1, [u]) \xrightarrow[\mathscr{A}']{x} (q, [ux])$, par induction on peut affirmer qu'on peut trouver v_1 tel que $q_0 \xrightarrow[\mathscr{A}]{v_1} q_1$ et $v_1 \rightsquigarrow^* u$. On sait également (par le résultat (25) et la propriété 3) qu'il existe q_2, v_2 et $v_3 \in \text{suffixes}(ux)$ tels que $q_1 \xrightarrow[\mathscr{A}]{x} q_2$, $v_2 \rightsquigarrow^* \overline{v}_3 v_3$ et $q_2 \xrightarrow[\mathscr{A}]{v_2} q$. On obtient donc

$$q_0 \xrightarrow[\mathscr{A}]{v_1} q_1 \xrightarrow[\mathscr{A}]{x} q_2 \xrightarrow[\mathscr{A}]{v_2} q \text{ et } v_1 x v_2 \rightsquigarrow^* u x v_2 \rightsquigarrow^* u x \overline{v}_3 v_3 \rightsquigarrow^* ux.$$

Si l'on choisit $q \in F$, on obtient le résultat souhaité. □

Lemme ii:
$L \subseteq L'$ ■

Preuve Cette partie est très simple : on peut remarquer que $\forall u, \gamma(u)$ est une relation réflexive. Donc

$$\forall q_1, q_2, x, \left(q_1 \xrightarrow[\mathscr{A}]{x} q_2\right) \Rightarrow \left(\forall u, (q_1, [u]) \xrightarrow[\mathscr{A}']{x} (q_2, [ux])\right).$$

Comme de plus, en projetant sur la première composante les états de $\mathscr{A}'$, les états initiaux (respectivement finaux) de $\mathscr{A}$ sont initiaux (resp. finaux) dans $\mathscr{A}'$, on voit que l'on peut simuler toute exécution acceptante de $\mathscr{A}$ en une exécution acceptante de $\mathscr{A}'$. □

Lemme iii:
L' est clos. ■

Cette partie nécessite une lemme dont la preuve s'avère assez technique, et qu'on ne laissera donc en annexe A.3 :

Lemme iv:
$\left((q_1, [uw]) \xrightarrow[\mathscr{A}']{x} (q_2, [uwx]) \xrightarrow[\mathscr{A}']{\overline{wx}\, wx} (q_3, [uwx\, \overline{wx}\, wx])\right) \Rightarrow (q_1, [uw]) \xrightarrow[\mathscr{A}']{x} (q_3, [uwx])$. ■

Avec ce résultat, on peut donner une preuve concise du lemme iii :

Preuve L'énoncé peut se reformuler ainsi :

$$\forall u, v, \text{ tels que } u \rightsquigarrow v, \text{ si } u \in L' \text{ alors } v \in L'$$

Considérons $u = u_1 wx \cdot \overline{wx} \cdot wxu_2 = u_1 wx \cdot x'\overline{w} \cdot wxu_2$ et $v = u_1 wxu_2$ (le cas $w = \epsilon$ n'a pas d'intérêt puisque dans ce cas $u = v$). Alors en combinant le résultat iv et la propriété 4 :

$$(q_0, [\mathbb{1}]) \xrightarrow{u_1 w} (q_1, [u_1 w]) \xrightarrow{x} (q_2, [u_1 wx]) \xrightarrow{x' w^\vee wx} (q_3, [u_1 wxx' w^\vee wx]) \xrightarrow{u_2} (q_f, [u])$$
$$\searrow^{x}_{\textit{Lem. iv}} \quad (q_3, [u_1 wx]) \xrightarrow[\textit{Eq.(24)+Prop. 4}]{u_2} (q_f, [v])$$

□

Les lemmes ii et iii nous disent que L' est clos et contient L, donc par définition d'une clôture (plus petit ensemble clos contenant l'ensemble de départ) on a $\mathfrak{cl}(L) \subseteq L'$. Le lemme i nous donnant l'autre implication, on a prouvé le théorème 4.

3 Analyse, implémentation et conclusions

3.1 Complexité

Comme nous parlons ici d'algorithmes plus que de programmes, il est difficile de donner des bornes de complexité précises, en raison de la multitude de choix possibles pour les structures de données intervenant au cours du calcul. En revanche, une mesure qui nous paraît pertinente est la taille de l'automate clôture que l'on construit.

Tout d'abord, une remarque s'impose. Dans la construction originale, on suppose avoir en entrée un automate *déterministe*, afin de pouvoir construire le monoïde des transitions. En sortie, il donne également un automate déterministe. Notre algorithme ne met pas cette restriction, mais de la manière dont on l'a présenté plus haut, renvoie un automate non déterministe. Pour comparer pertinemment les deux constructions, on va étudier la taille d'un automate *déterministe* que l'on construit à partir d'un automate *non-déterministe*.

On suppose que l'on part d'un automate non-déterministe avec n états, ce que l'on appelera dans la suite la *taille* de l'automate.

1. Dans la méthode originale, il faut commencer par produire un automate déterministe, dont l'ensemble des états sera $\mathscr{P}(Q)$, et donc de taille 2^n. L'automate construit dans [BÉS95] a pour ensemble d'états des couples de parties du monoïde des transitions de l'automate d'entrée. La taille de cet ensemble peut être bornée de la manière suivante :
   – le monoïde des transitions est une partie de $\mathscr{P}(Q)^{\mathscr{P}(Q)}$, donc $|M_{\mathscr{A}}|$ est borné par $(2^n)^{2^n} = 2^{n\times 2^n}$;
   – par conséquent $|\mathscr{P}(M_{\mathscr{A}})| \leqslant 2^{2^{n\times 2^n}}$ et donc :
   – l'automate final a au plus $2^{2^{n\times 2^n}} \times 2^{2^{n\times 2^n}} = 2^{2\times 2^{n\times 2^n}} = 2^{2^{n\times 2^n+1}}$ états.
2. Dans notre construction, on peut remarquer qu'il est très simple de produire un automate déterministe : en effet, toutes les arêtes partant de $(p, [w])$ et étiquetées par x mènent à des états dont le second membre vaut $[wx]$. Donc, pour déterminiser cet automate, on n'a qu'à appliquer une *powerset construction* au premier membre. Dans la pratique, cela donne un automate ayant comme ensemble d'états $\mathscr{P}(Q) \times G$ et comme fonction de transition :

$$\delta' : ((P, [w]), x) \mapsto (\{q \mid \exists p \in P : (p, q) \in \Delta_x \circ \gamma(wx)\}, [wx]).$$

   Ainsi, on peut évaluer la taille de notre construction :
   – $\mathscr{P}(Q)$ est de taille 2^n,
   – G est un ensemble de classes d'équivalence de $\sim_\gamma$. γ prend pour valeurs des relations binaires réflexives sur Q, par conséquent $\sim_\gamma$ a moins de $2^{n\times(n-1)}$ classes d'équivalence.

   Au final on peut dire que $|\mathscr{P}(Q) \times G| \leqslant 2^n \times 2^{n\times(n-1)} = 2^{n^2}$, ce qui est nettement moins que le $2^{2^{n\times 2^n+1}}$ qu'on a avec l'autre construction.

De plus, la limite que l'on a donnée est lâche, puisque les valeurs de γ sont en réalité des relations réflexives et *transitives*. Malheureusement, il semble plus difficile de dénombrer les relations transitives, nous n'avons donc pris en compte que la réflexivité.

3.2 Implémentation

On a implémenté ces deux constructions en OCaml, de manière très fonctionnelle et modulaire. Les sources et la documentation sont disponible ici : `perso.ens-lyon.fr/paul.brunet`.

Ce programme prend en entrée un fichier contenant une liste d'équations, d'inéquations strictes ou larges, et de négations d'équations (de la forme $e \neq f$) sur des expressions régulières avec converse et teste si elles sont valides dans CKA_2 . Pour cela on leur applique les trois constructions :

- construction originale,
- version déterministe de notre construction,
- version non-déterministe de notre construction.

Puis on teste dans les trois cas l'équivalence des automates construits de part et d'autre. L'équivalence de deux automates est implémentée à l'aide de la méthode de Hopcroft et Karp [HK71].

De plus, de nombreuses options sont disponibles en ligne de commande, afin de ne selectionner qu'une partie des méthodes proposées, de chronométrer l'exécution, d'afficher la taille des automates produits, voire de dessiner ces automates.

3.3 Conclusions et perspectives

Nous sommes partis des travaux de Bloom, Ésik, Stefanescu et Bernátsky, parus en 1995 [BÉS95][ÉB95].

Nous avons proposé un algorithme nettement plus efficace pour calculer la clôture d'un langage et donc pour décider la prouvabilité dans CKA_2 . Nous avons implémenté en OCaml cette construction et l'avons testée sur un certain nombre d'expressions régulières avec converse. Nous avons pu constater que notre méthode est nettement plus efficace que celle proposée dans l'article original.

Nous avons également pu reformuler de manière plus simple les preuves des résultats sur les modèles de langages :
pour tous $e, f \in \mathrm{Reg}_X^\vee$:

$$[\![\mathbf{e}]\!] = [\![\mathbf{f}]\!] \Leftrightarrow \mathrm{CKA}_1 \vdash e = f \Leftrightarrow e \equiv_{Lang^\vee} f.$$

Nous aimerions maintenant trouver une preuve plus simple de la complétude de CKA_2 :

$$e \equiv_{Rel^\vee} f \Rightarrow \mathfrak{cl}([\![\mathbf{e}]\!]) = \mathfrak{cl}([\![\mathbf{f}]\!]) \Rightarrow \mathrm{CKA}_2 \vdash e = f.$$

On a pu simplifier la partie $e \equiv_{Rel^\vee} f \Rightarrow \mathfrak{cl}([\![\mathbf{e}]\!]) = \mathfrak{cl}([\![\mathbf{f}]\!])$. Pour la suite la preuve proposée dans [ÉB95] ne nous satisfait pas et nous aimerions en trouver une analogue à la preuve de complétude des algèbres de Kleene donnée par Kozen dans [Koz94].

L'objectif de cette démarche est d'implémenter cette méthode dans la librairie *Relation Algebra* [(xii)] de Coq pour qu'elle puisse traiter les expressions avec converse.

Nous aimerions pouvoir réutiliser cette approche par automates pour apporter des solutions à d'autres problèmes de décision d'algèbres de Kleene étendues, par exemple pour les algèbres d'actions [Pra91].

Références

[BÉS95] Stephen L Bloom, Zoltán Ésik, and Gh Stefanescu. Notes on equational theories of relations. *algebra universalis*, 33(1) :98–126, 1995. 1, 2, 5, 7, 8, 9, 10, 11, 12, 17, 18

(xii). `perso.ens-lyon.fr/damien.pous/ra/`

[Con71] J.H. Conway. *Regular algebra and finite machines.* Chapman and Hall Mathematics Series, 1971. 1

[ÉB95] Zoltán Ésik and L Bernátsky. Equational properties of kleene algebras of relations with conversion. *Theoretical Computer Science*, 137(2) :237–251, 1995. 1, 7, 18

[HK71] John E Hopcroft and Richard M Karp. A linear algorithm for testing equivalence of finite automata. Technical report, Cornell University, 1971. 18

[Kle51] S.C. Kleene. *Representation of Events in Nerve Nets and Finite Automata*. Memorandum (Rand Corporation). Rand Corporation, 1951. 4

[Koz94] Dexter Kozen. A completeness theorem for kleene algebras and the algebra of regular events. *Information and Computation*, 110 :366–390, 1994. 3, 4, 18

[Pra91] Vaughan Pratt. Action logic and pure induction. In *Logics in AI : European Workshop JELIA '90, LNCS 478*, pages 97–120. Springer-Verlag, 1991. 2, 3, 18

[Red64] V. N. Redko. On defining relations for the algebra of regular events. *Ukrainskii Matematicheskii Zhurnal*, pages 120–126, 1964. 3

[RS59] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM J. Res. Dev.*, 3(2) :114–125, April 1959. 9

A Compléments : preuves des lemmes admis

A.1 Preuve du résultat (22)

$$v \rightsquigarrow^* u \Leftrightarrow (0, n) \in \hat{\phi_u}(v).$$

On va commencer par définir un automate $\Phi(u)$, qui reconnaitra les mots v tels que $(0, n) \in \hat{\phi}_u(v)$, puis on montrera que le langage reconnu par cet automate est la clôture supérieure par $\rightsquigarrow$ de $\{u\}$. Rappelons la définition de ϕ_u, pour tout mot $u = u_1 \cdots u_n$, avec $\forall i, u_i \in \mathbf{X}$:

$$\forall x \in \mathbf{X}, \phi_u(x) \coloneqq \{(i-1, i) \mid u_i = x\} \cup \{(i, i-1) \mid u_i = \overline{x}\}.$$

Définition ($\Phi(u)$)
Pour tout mot $u = u_1 \cdots u_n \in \mathbf{X}^*$, on définit l'automate $\Phi(u) = \langle\{0 \dots n\}, \mathbf{X}, \{0\}, \{n\}, \Phi_u\rangle$, avec l'ensemble de transitions $\Phi_u \coloneqq \{(i-1, u_i, i) \mid 1 \leqslant i \leqslant n\} \cup \{(i, \overline{u}_i, i-1) \mid 1 \leqslant i \leqslant n\}$. *

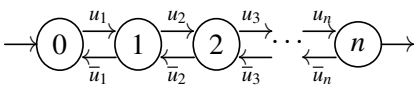


FIGURE 6 – $\Phi(u_1 \cdots u_n)$

En comparant les définitions de $\Phi(u)$ et de ϕ_u, on réalise assez simplement que

$$(i, j) \in \phi_u(x) \Leftrightarrow (i, x, j) \in \Phi_u,$$

et à partir de cela que $(i, j) \in \hat{\phi}_u(v)$ équivaut à l'existence d'un chemin de i à j étiqueté par v dans l'automate $\Phi(u)$. Par conséquent on a la relation suivante entre ϕ et Φ :

$$(0, n) \in \hat{\phi}_u(v) \Leftrightarrow v \in L(\Phi_u). \tag{26}$$

Par ailleurs, on peut remarquer que $(i, x, j) \in \Phi_u \Leftrightarrow (j, \overline{x}, i) \in \Phi_u$. On peut étendre ce résultat aux chemins de la manière suivante :

$$i \xrightarrow[\Phi(u)]{w} j \Rightarrow j \xrightarrow[\Phi(u)]{\overline{w}} i.$$

Donc, pour tous mots u_1,w,u_2,

$$\left(0 \xrightarrow[\Phi(u)]{u_1} i \xrightarrow[\Phi(u)]{w} j \xrightarrow[\Phi(u)]{u_2} n\right) \Rightarrow \left(0 \xrightarrow[\Phi(u)]{u_1} i \xrightarrow[\Phi(u)]{w} j \xrightarrow[\Phi(u)]{\overline{w}} i \xrightarrow[\Phi(u)]{w} j \xrightarrow[\Phi(u)]{u_2} n\right).$$

On a donc montré que si $v_1 \in L(\Phi(u))$ et $v_2 \rightsquigarrow v_1$, alors $v_2 \in L(\Phi(u))$, ce qui s'étend, comme $u \in L(\Phi(u))$:

$$v \rightsquigarrow^* u \Rightarrow v \in L(\Phi(u)). \tag{27}$$

On a maintenant ce qu'il faut pour prouver un sens de l'équivalence :

$$v \rightsquigarrow^* u \overset{(27)}{\Longrightarrow} v \in L(\Phi(u)) \overset{(26)}{\Longrightarrow} (0, n) \in \hat{\phi}_u(v).$$

On souhaite maintenant prouver l'autre sens. On va procéder par induction sur u pour montrer que

$$\forall u, (v \in L(\Phi(u)) \Rightarrow v \rightsquigarrow^* u). \tag{28}$$

Le cas ϵ étant trivial, il reste le cas $xu = xu_2 \cdots u_n$. On peut remarquer que l'automate $\Phi(xu_2 \cdots u_n)$ reconnait le même langage que l'automate $\Phi'(xu_2 \cdots u_n)$ donné sur la figure 7.

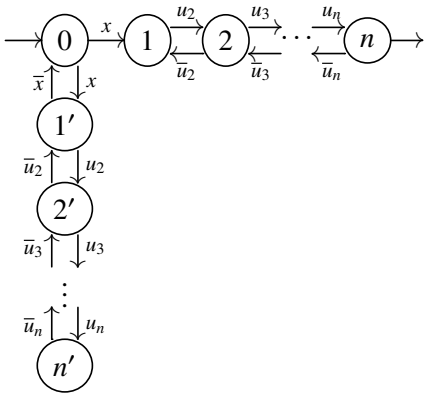


FIGURE 7 – $\Phi'(xu_2 \cdots u_n)$

On reconnait alors en la partie verticale l'automate de $\Gamma(\overline{u}_n \cdots \overline{u}_2 \overline{x}) = \Gamma(\overline{xu})$ (comme donné sur la figure 5). Cela nous indique que l'équation suivante est vérifiée :

$$L(\Phi(xu)) = \Gamma(\overline{xu}) \cdot x \cdot L(\Phi(u)). \tag{29}$$

Ainsi, on a pour $v \in L(\Phi(xu))$, $v = v_1 x v_2$ avec $v_1 \in \Gamma(\overline{xu})$ et $v_2 \in L(\Phi(u))$. D'après le résultat (23) on peut affirmer que $v_1 \rightsquigarrow^* \overline{w}_1 w_1$ avec $\overline{xu} = w_2 w_1$, c'est à dire $xu = \overline{w}_1 \overline{w}_2$. Par induction $v_2 \rightsquigarrow^* u$, soit :

$$v = v_1 x v_2 \rightsquigarrow^* v_1 xu \rightsquigarrow^* \overline{w}_1 w_1 xu = \overline{w}_1 w_1 \overline{w}_1 \overline{w}_2 \rightsquigarrow \overline{w}_1 \overline{w}_2 = xu.$$

On a donc prouvé le résultat (28) ce qui nous permet de compléter la preuve de la manière suivante :

$$v \rightsquigarrow^* u \underset{(28)}{\overset{(27)}{\rightleftharpoons}} v \in L(\Phi(u)) \overset{(26)}{\Longleftrightarrow} (0, n) \in \hat{\phi}_u(v).$$

A.2 Preuve du résultat (2)

On va montrer par induction sur w que :

$$\forall u \in \Gamma(w), \exists v \in \text{suffixes}(w) : u \rightsquigarrow^* \overline{v} v.$$

1. Le cas $w = \epsilon$ est trivial ;

2. $u \in \Gamma(wx)$ implique qu'il y a un $n \in \mathbb{N}$ tel que $u \in (x' \cdot \Gamma(w) \cdot x)^n$. On va maintenant prouver par récurrence sur n :

$$\forall n, \forall u \in (x' \cdot \Gamma(w) \cdot x)^n, \exists v \in \text{suffixes}(wx) : u \rightsquigarrow^* \overline{v}v.$$

   (a) Si $n = 0$ alors $u = \epsilon$, ce qui donne encore un cas trivial ;

   (b) $u \in (x' \cdot \Gamma(w) \cdot x)^{n+1} = (x' \cdot \Gamma(w) \cdot x)^n \cdot x' \cdot \Gamma(w) \cdot x$, c'est à dire que $u = u_1 x' u_2 x$ avec $u_1 \in (x' \cdot \Gamma(w) \cdot x)^n$ et $u_2 \in \Gamma(w)$. Par récurrence il existe $w_2 \in \text{suffixes}(wx)$ tel que $u_1 \rightsquigarrow^* \overline{w}_2 w_2$. Par induction il existe $w_4 \in \text{suffixes}(w)$ tel que $u_2 \rightsquigarrow^* \overline{w}_4 w_4$. On peut compléter w à chaque fois ce qui nous donne, pour résumer :

$$\begin{aligned} u_1 &\rightsquigarrow^* \overline{w}_2 w_2 && \text{(Récurence sur } n\text{)} \\ u_2 &\rightsquigarrow^* \overline{w}_4 w_4 && \text{(Induction sur } w\text{)} \\ wx &= w_1 w_2 && (w_2 \in \text{suffixes}(wx)) \\ w &= w_3 w_4. && (w_4 \in \text{suffixes}(w)) \end{aligned}$$

   Plusieurs cas sont possibles pour l'équation $wx = w_1 w_2$:

   i. $w_2 = \epsilon$: dans ce cas $u = u_1 x' u_2 x \rightsquigarrow^* \overline{\epsilon}\epsilon x' u_2 x \rightsquigarrow^* x' \overline{w}_4 w_4 x = \overline{w_4 x} \cdot w_4 x$ et $w_4 x \in \text{suffixes}(wx)$.

   ii. Sinon on sait que w_2 finit par x, on peut donc modifier légèrement nos notations en remplaçant w_2 par $w_2 x$ et donc $u_1 \rightsquigarrow^* \overline{w_2 x} \cdot w_2 x = x' \overline{w}_2 \cdot w_2 x$. Par conséquent :

$$u = u_1 \cdot x' u_2 x \rightsquigarrow^* x' \overline{w}_2 w_2 x \cdot x' \cdot u_2 \cdot x \rightsquigarrow^* x' \overline{w}_2 w_2 x \cdot x' \cdot \overline{w}_4 w_4 \cdot x.$$

   À nouveau on étudie deux cas de figure, dépendant des tailles relatives de w_1 et w_3 (en se rappelant que $w = w_1 w_2 = w_3 w_4$) :

   A. $|w_1| < |w_3|$:

<table>
<tr><td colspan="2">w_1</td><td colspan="2">w_2</td></tr>
<tr><td colspan="2">w_1</td><td>w_5</td><td>w_4</td></tr>
<tr><td colspan="2">w_3</td><td colspan="2">w_4</td></tr>
</table>

   On peut décomposer $w_3 = w_1 w_5$ et $w_2 = w_5 w_4$ ce qui nous permet d'écrire :

$$u \rightsquigarrow^* x' \overline{w_5 w_4} \cdot w_5 w_4 \cdot x \cdot x' \overline{w}_4 w_4 x = x' \overline{w_5 w_4} w_5 \cdot w_4 x \cdot \overline{w_4 x} \cdot w_4 x.$$

   On identifie le motif $w_4 x \cdot \overline{w_4 x} \cdot w_4 x$ que l'on réduit pour obtenir

$$u \rightsquigarrow^* x' \overline{w_5 w_4} w_5 w_4 x = x' \overline{w}_2 w_2 x = \overline{w_2 x} \cdot w_2 x,$$

   ce qui est sympathique puisque $w_2 x \in \text{suffixes}(wx)$.

   B. $|w_3| \leqslant |w_1|$:

<table>
<tr><td colspan="2">w_1</td><td>w_2</td></tr>
<tr><td>w_3</td><td>w_5</td><td>w_2</td></tr>
<tr><td>w_3</td><td colspan="2">w_4</td></tr>
</table>

   On décompose à nouveau $w_1 = w_3 w_5$ et $w_4 = w_5 w_2$:

$$u \rightsquigarrow^* x' \overline{w}_2 w_2 x \cdot x' \cdot \overline{w_5 w_2} w_5 w_2 \cdot x = x' \overline{w}_2 w_2 x \cdot x' \cdot \overline{w}_2 \overline{w}_5 w_5 w_2 \cdot x.$$

   On repère et réduit le motif $x' \overline{w}_2 w_2 x x' \overline{w}_2 = x' \overline{w}_2 \cdot \overline{x' \overline{w}_2} \cdot x' \overline{w}_2$ ce qui nous donne

$$u \rightsquigarrow^* x' \overline{w}_2 \overline{w}_5 w_5 w_2 x = \overline{w_4 x} \cdot w_4 x,$$

   et ça suffit puisque $w_4 x \in \text{suffixes}(wx)$.

A.3 Preuve du lemme iv

On aura besoin de deux lemmes auxiliaires :

Lemme v:

$\forall w = w_1 w_2, \forall v \in \Gamma(uw\overline{w}_2), \overline{w}_2 v \rightsquigarrow^* v' \in \Gamma(uw)\overline{w}_2$ ■

PREUVE Par la propriété 2, $v \rightsquigarrow^* \overline{v}_1 v_1$, pour un $v_1 \in \text{suffixes}(uw\overline{w}_2)$. On va faire une étude de cas sur v_1.

1. $w_2 = w_3 w_4 \wedge v_1 = \overline{w}_3 : \overline{w}_2 v \rightsquigarrow^* \overline{w}_4 \overline{w}_3 w_3 \overline{w}_3 \rightsquigarrow \overline{w}_4 \overline{w}_3 = \overline{w}_2$.
2. $w = w_3 w_4 \wedge v_1 = w_4 \overline{w}_2$: décomposons l'équation $w_3 w_4 = w = w_1 w_2$.
   (a) $w_3 = w_1 k \wedge w_2 = k w_4 : \overline{w}_2 v \rightsquigarrow^* \overline{w}_2 k w_4 \overline{w}_4 w_4 \overline{w}_2 \rightsquigarrow \overline{w}_2 k w_4 \overline{w}_2 = \overline{w}_2 w_2 \overline{w}_2 \rightsquigarrow \overline{w}_2$.
   (b) $w_1 = w_3 k \wedge w_4 = k w_2 : \overline{w}_2 v \rightsquigarrow^* \overline{w}_2 w_2 \overline{w}_2 \overline{k} w_4 \overline{w}_2 \rightsquigarrow \overline{w}_2 \overline{k} w_4 \overline{w}_2 = \overline{w}_4 w_4 \overline{w}_2 \in \Gamma(uw)\overline{w}_2$.
3. $u = u_1 u_2 \wedge v_1 = u_2 w \overline{w}_2 :$ $\overline{w}_2 v \rightsquigarrow^* \overline{w}_2 w_2 \overline{w}_2 \overline{w}_1 \overline{u}_2 u_2 w \overline{w}_2$
   $\rightsquigarrow \overline{w}_2 \overline{w}_1 \overline{u}_2 u_2 w \overline{w}_2 = \overline{w}\ \overline{u}_2 u_2 w \overline{w}_2 \in \Gamma(uw)\overline{w}_2$.

□

Lemme vi:

$\forall w = w_1 w_2, \forall v \in \Gamma(uw\overline{w}\ w_1), \overline{w}\ w_1 v \rightsquigarrow^* v' \in \Gamma(uw)\overline{w}\ w_1$ ■

PREUVE La propriété 2 nous dit que $v \rightsquigarrow^* \overline{v}_1 v_1$, avec $v_1 \in \text{suffixes}(uw\overline{w}\ w_1)$. Faisons une analyse de cas sur v_1 :

1. $w_1 = w_3 w_4 \wedge v_1 = w_4 : \overline{w}\ w_1 v \rightsquigarrow^* \overline{w}\ w_3 w_4 \overline{w}_4 w_4 \rightsquigarrow \overline{w}\ w_1$;
2. $w = w_3 w_4 \wedge v_1 = \overline{w}_3 w_1 : \overline{w}\ w_1 v \rightsquigarrow^* \overline{w}_2 \overline{w}_1 w_1 \overline{w}_1 w_3 \overline{w}_3 w_1 \rightsquigarrow \overline{w}_4 \overline{w}_3 w_3 \overline{w}_3 w_1 \rightsquigarrow \overline{w}\ w_1$;
3. $w = w_3 w_4 \wedge v_1 = w_4 \overline{w} w_1 :$ $\overline{w}\ w_1 v \rightsquigarrow^* \overline{w}\ w_1 \overline{w}_1 w \overline{w}_4 w_4 \overline{w}\ w_1$
   $= \overline{w}_2 \overline{w}_1 w_1 \overline{w}_1 w \overline{w}_4 w_4 \overline{w}\ w_1$
   $\rightsquigarrow \overline{w}_2 \overline{w}_1 w \overline{w}_4 w_4 \overline{w}\ w_1$
   $= \overline{w}\ w_3 w_4 \overline{w}_4 w_4 \overline{w}\ w_1$
   $\rightsquigarrow \overline{w}\ w_3 w_4 \overline{w}\ w_1$
   $= \overline{w}\ w \overline{w}\ w_1$
   $\rightsquigarrow \overline{w}\ w_1$;
4. $u = u_1 u_2 \wedge v_1 = u_2 w \overline{w}_1 :$ $\overline{w}\ w_1 v \rightsquigarrow^* \overline{w}\ w_1 \overline{w}_1 w \overline{w}\ \overline{u}_2 u_2 w \overline{w}_1$
   $= \overline{w}_2 \overline{w}_1 w_1 \overline{w}_1 w \overline{w}\ \overline{u}_2 u_2 w \overline{w}_1$
   $\rightsquigarrow \overline{w}_2 \overline{w}_1 w \overline{w}\ \overline{u}_2 u_2 w \overline{w}_1$
   $= \overline{w}\ w \overline{w}\ \overline{u}_2 u_2 w \overline{w}_1$
   $\rightsquigarrow \overline{w}\ \overline{u}_2 u_2 w \overline{w}_1 \in \Gamma(uw)\overline{w}\ w_1$.

□

Ces deux lemmes nous permettent de prouver le résultat souhaité :

$$\text{Si } (q_1, [uw]) \xrightarrow[\mathscr{A}']{x} (q_2, [uwx]) \xrightarrow[\mathscr{A}']{x' \overline{w}\ wx} (q_3, [uwxx'\overline{w}wx]) \text{ alors } (q_1, [uw]) \xrightarrow[\mathscr{A}']{x} (q_3, [uwx]).$$

PREUVE Pour alléger un peu l'écriture, on notera $\mathbf{w} := wx = x_1 \cdots x_n$.

On peut voir, avec le résultat (25) et la propriété 3, que l'éxécution $(q_2, [u\mathbf{w}]) \xrightarrow[\mathscr{A}']{\overline{\mathbf{w}}\mathbf{w}} (q_3, [u\mathbf{w}\overline{\mathbf{w}}\mathbf{w}])$ implique qu'il existe v tel que $q_2 \xrightarrow[\mathscr{A}]{v} q_3$, avec :

$$v \in x'_n \Gamma(u\mathbf{w}x'_n) x'_{n-1} \Gamma(u\mathbf{w}x'_n x'_{n-1}) \cdots x'_1 \Gamma(u\mathbf{w}\overline{\mathbf{w}}) x_1 \Gamma(u\mathbf{w}\overline{\mathbf{w}}x_1) x_2 \cdots x_{n-1} \Gamma(u\mathbf{w}\overline{\mathbf{w}}x_1 \cdots x_{n-1}) x_n \Gamma(u\mathbf{w}\overline{\mathbf{w}}\mathbf{w}).$$

On peut alors appliquer itérativement les lemmes v et vi, ce qui donne :

$$
\begin{array}{c}
x'_n\Gamma(u\mathbf{w}x'_n)x'_{n-1}\Gamma(u\mathbf{w}x'_nx'_{n-1})\cdots x'_1\Gamma(u\mathbf{w}\overline{\mathbf{w}})x_1\Gamma(u\mathbf{w}\overline{\mathbf{w}}x_1)x_2\cdots x_{n-1}\Gamma(u\mathbf{w}\overline{\mathbf{w}}x_1\cdots x_{n-1})x_n\Gamma(u\mathbf{w}\overline{\mathbf{w}}\mathbf{w}) \\
\rightsquigarrow^* \\
\Gamma(u\mathbf{w})x'_nx'_{n-1}\Gamma(u\mathbf{w}x'_nx'_{n-1})\cdots x'_1\Gamma(u\mathbf{w}\overline{\mathbf{w}})x_1\Gamma(u\mathbf{w}\overline{\mathbf{w}}x_1)x_2\cdots x_{n-1}\Gamma(u\mathbf{w}\overline{\mathbf{w}}x_1\cdots x_{n-1})x_n\Gamma(u\mathbf{w}\overline{\mathbf{w}}\mathbf{w}) \\
\rightsquigarrow^* \\
\Gamma(u\mathbf{w})\Gamma(u\mathbf{w})x'_nx'_{n-1}\cdots x'_1\Gamma(u\mathbf{w}\overline{\mathbf{w}})x_1\Gamma(u\mathbf{w}\overline{\mathbf{w}}x_1)x_2\cdots x_{n-1}\Gamma(u\mathbf{w}\overline{\mathbf{w}}x_1\cdots x_{n-1})x_n\Gamma(u\mathbf{w}\overline{\mathbf{w}}\mathbf{w}) \\
\vdots \\
\Gamma(u\mathbf{w})^n\overline{\mathbf{w}}\Gamma(u\mathbf{w}\overline{\mathbf{w}})x_1\Gamma(u\mathbf{w}\overline{\mathbf{w}}x_1)x_2\cdots x_{n-1}\Gamma(u\mathbf{w}\overline{\mathbf{w}}x_1\cdots x_{n-1})x_n\Gamma(u\mathbf{w}\overline{\mathbf{w}}\mathbf{w}) \\
\rightsquigarrow^* \\
\Gamma(u\mathbf{w})^n\overline{\mathbf{w}}x_1\Gamma(u\mathbf{w}\overline{\mathbf{w}}x_1)x_2\cdots x_{n-1}\Gamma(u\mathbf{w}\overline{\mathbf{w}}x_1\cdots x_{n-1})x_n\Gamma(u\mathbf{w}\overline{\mathbf{w}}\mathbf{w}) \\
\rightsquigarrow^* \\
\Gamma(u\mathbf{w})^{n+1}\overline{\mathbf{w}}x_1x_2\cdots x_{n-1}\Gamma(u\mathbf{w}\overline{\mathbf{w}}x_1\cdots x_{n-1})x_n\Gamma(u\mathbf{w}\overline{\mathbf{w}}\mathbf{w}) \\
\vdots \\
\Gamma(u\mathbf{w})^{2n}\overline{\mathbf{w}}\mathbf{w}\subseteq\Gamma(u\mathbf{w}).
\end{array}
$$

Donc $v \rightsquigarrow^* v' \in \Gamma(u\mathbf{w})$ ce qui signifie, comme $\Gamma(u\mathbf{w})$ est clos vers le haut pour $\rightsquigarrow$, que $v \in \Gamma(u\mathbf{w})$. On peut aussi déduire de $(q_1,[uw]) \xrightarrow[\mathscr{A}']{x} (q_2,[uwx])$ qu'il existe un état q_4 et un mot $z \in \Gamma(u\mathbf{w})$ tels que $q_1 \xrightarrow[\mathscr{A}]{x} q_4 \xrightarrow[\mathscr{A}]{z} q_2$. Ainsi $q_4 \xrightarrow[\mathscr{A}]{zv} q_3$ et $zv \in \Gamma(u\mathbf{w})\Gamma(u\mathbf{w}) = \Gamma(u\mathbf{w})$. Par conséquent $(q_4,q_3) \in \gamma(uwx)$ c'est à dire $(q_1,q_3) \in R_x \circ \gamma(uwx)$ ce qui signifie que

$$(q_1,[uw]) \xrightarrow[\mathscr{A}']{x} (q_3,[uwx]).$$

□

B Preuve de confluence de la relation de réécriture

On va donner ici le plan d'une preuve de la confluence de la relation de réécriture $uw\overline{w}wv \rightsquigarrow uwv$, qui apparaît de manière prépondérante dans l'étude de la décidabilité de l'algèbre CKA_2. Cette preuve a été réalisée en Coq, aussi nous ne nous attarderons pas trop ici sur les détails calculatoires.

Remarque

Nous avons en réalité pu prouver en Coq un résultat plus fort : soit $'$ une opération involutive [(xiii)] et $\bar{\cdot}$ définie par $\overline{\epsilon} = \epsilon$ et $\overline{wx} = x'\overline{w}$, alors la relation $\rightsquigarrow$ définie par $ab\overline{b}bc \rightsquigarrow abc$ vérifie la propriété suivante :

$$\forall u, v_1, v_2 \in X^*, u \rightsquigarrow v_1 \wedge u \rightsquigarrow v_2 \Rightarrow \exists v : v_1 \rightsquigarrow^{\leqslant 4} v \wedge v_2 \rightsquigarrow^{\leqslant 4} v.$$

Le schéma de cette preuve étant le même que celle présentée ci-dessous, on a choisi de donner ici une preuve de la confluence locale de la relation de réduction utilisée pour définir $cl(L)$, qui en étant un peu plus simple est déjà suffisamment technique.

B.1 Définitions et notations

On notera $|x|$ pour la longueur du mot x et ϵ le mot vide.

On se place ici dans un alphabet fixé X, que l'on muni d'une opération "prime" involutive et sans point fixe :

$$\forall x \in X, x' \neq x \wedge x = x''. \tag{30}$$

On étend cette opération aux mots par une opération "converse", définie de la manière suivante :

$$\begin{cases} \overline{\epsilon} = \epsilon \\ \forall x \in X, \forall w \in X^*, \overline{xw} = \overline{w}x' \end{cases} \tag{31}$$

Définition (Relation de réduction)

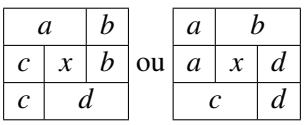
$$u \rightsquigarrow v \quad \text{signifie que} \quad \exists a, b, c \in X^* : u = ab\overline{b}bc \wedge v = abc. \qquad *$$

La preuve en Coq utilise intensivement le lemme suivant :

Lemme vii (Décomposition):

$$\forall a, b, c, d \in X^*, ab = cd \Rightarrow \exists e : (a = ce \wedge d = eb) \vee (c = ae \wedge b = ed). \qquad \blacksquare$$

Graphiquement, cela signifie que si $ab = cd$ alors :



(32)

On aura aussi besoin de quelques autres lemmes.

(xiii). La différence avec ce qui a été défini plus haut est que x et x' ne sont ici pas nécessairement différents, alors que précédèrent on exige que X et X' soient disjoints.

Lemme viii ($ab = ca$):

$$\forall a, b, c \in X^*, ab = ca \wedge c \neq \epsilon \Rightarrow \exists d, e : c = de \wedge a \in (de)^* d. \qquad \blacksquare$$

PREUVE On va procéder par induction sur a.
- Si $a = \epsilon$, $c = c\epsilon$ et $a = \epsilon \in (c\epsilon)^*\epsilon$.
- On applique le lemme de décomposition.
  - Si $c = ax \wedge b = xa$, alors $a \in (ax)^* a$.
  - Sinon $a = cx \wedge a = xb$, donc $xb = cx$. Comme $c \neq \epsilon$, $0 < |c|$ et donc $|x| < |x| + |c| = |a|$. On applique donc l'hypothèse d'induction qui nous indique que $c = de$ et $x \in (de)^* d$, c'est à dire $a = cx \in de(de)^* d \subseteq (de)^* d$. □

Lemme ix ($u = \overline{u}$):

$$\forall u \in X^*, u = \overline{u} \Leftrightarrow \exists t : u = t\bar{t}. \qquad \blacksquare$$

PREUVE **Sens direct :** par induction sur u :
- $\epsilon = \overline{\epsilon}$ et $\epsilon = \epsilon\overline{\epsilon}$.
- $xu = \overline{xu} = \overline{u}x'$, si $u = \epsilon$, alors $x = x'$, ce qui est interdit par définition de $.'$, donc $u = vy$ ($v \in X^*$ et $y \in X$). De plus $xvy = \overline{vy}x' = y'\overline{v}x'$, donc $y = x'$ et $v = \overline{v}$. Par induction ($|v| = |xu| - 2$), on obtient $v = t\bar{t}$, et donc $xu = xt\bar{t}x' = xt \cdot \overline{xt}$.

Sens retour : $\overline{t \cdot \bar{t}} = \bar{\bar{t}} \cdot \bar{t} = t \cdot \bar{t}$. □

Lemme x ($u\overline{u} \cdot \overline{v} = v \cdot u\overline{u}$):

$$\forall u, v \in X^*, u\overline{u} \cdot \overline{v} = v \cdot u\overline{u} \wedge 2|u| \leqslant |v| \Rightarrow \exists t : v = u\overline{u} \cdot t\bar{t}. \qquad \blacksquare$$

PREUVE On décompose $u\overline{u} \cdot \overline{v} = v \cdot u\overline{u}$:
- $u\overline{u} = vx \wedge u\overline{u} = xv$: $|v| + |x| = |vx| = |u\overline{u}| = 2|u| \leqslant |v|$ donc $x = \epsilon$, donc $v = vx = u\overline{u} = u\overline{u}\epsilon\overline{\epsilon}$.
- $v = u\overline{u}x \wedge \overline{v} = xu\overline{u}$. Dans ce cas $xu\overline{u} = \overline{v} = \overline{u\overline{u}x} = \overline{x} \cdot u\overline{u}$, d'où l'on déduit $x = \overline{x}$. Par le lemme précédent, on peut écrire $x = t\bar{t}$, et donc $v = u\overline{u} \cdot x = u\overline{u} \cdot t\bar{t}$. □

Lemme xi:

$$\forall a, b, c, d \in X^*, ab = cad \Rightarrow \exists t, s : b = tsd \wedge c = st \wedge ca = ats. \qquad \blacksquare$$

PREUVE On va faire une induction sur a :
- $b = c\epsilon d \Rightarrow b = \epsilon cd \wedge c = c\epsilon \wedge c\epsilon = \epsilon\epsilon c$.
- On décompose l'hypothèse :

– $a = cx \wedge ad = xb$: dans ce cas on remarque que $cxd = xb$, on va appliquer l'hypothèse d'induction (si $c = \epsilon$ le résultat est trivial et sinon $|a| = |c|+|x| > |x|$). Cela nous donne $b = tsd \wedge c = st \wedge cx = xts$, d'où l'on déduit $ca = ccx = cxts = ats$.
– $c = ax \wedge b = xad$: il suffit de prendre $t = x$ et $s = a$. □

On note $u_1 \leftrightsquigarrow u_2$ pour $\exists v : u_1 \rightsquigarrow^* v \wedge u_2 \rightsquigarrow^* v$. Le théorème que l'on souhaite montrer ici est le suivant :

Théorème (Confluence locale):

$$\forall u, v1, v2 \in X^*, u \rightsquigarrow v_1 \wedge u \rightsquigarrow v_2 \Rightarrow v_1 \leftrightsquigarrow v_2.$$

Pour cela, on va faire une étude de cas exhaustive. Cette décomposition va être articulée en deux parties, correspondant aux cas suivants :

1. $w_1\overline{w}_1w_1 = u_1w_2\overline{w}_2w_2u_2$.
2. $u_1w_1\overline{w}_1w_1 = w_2\overline{w}_2w_2u_2$.

Autrement représenté :

$w_1\overline{w}_1w_1$		
u_1	$w_2\overline{w}_2w_2$	u_2

ou

u_1	$w_1\overline{w}_1w_1$
$w_2\overline{w}_2w_2$	u_2

Par symétrie de la relation $\leftrightsquigarrow$, on voit que ces deux cas couvrent l'ensemble des paires critiques.

B.2 Premier cas

On considère tout d'abord le cas où un motif est complètement inclus dans l'autre, c'est à dire

$$w_1\overline{w}_1w_1 = u_1w_2\overline{w}_2w_2u_2.$$

On va décomposer ce cas en différents sous-cas :

1.

w_1			$\overline{w}_1w_1$
u_1	$w_2\overline{w}_2w_2$	a	$\overline{w}_1w_1$
u_1	$w_2\overline{w}_2w_2$	u_2	

ou

$w_1\overline{w}_1$	w_1		
$w_1\overline{w}_1$	a	$w_2\overline{w}_2w_2$	u_2
u_1		$w_2\overline{w}_2w_2$	u_2

2.

w_1	$\overline{w}_1$			w_1
w_1	a	$w_2\overline{w}_2w_2$	b	w_1
u_1		$w_2\overline{w}_2w_2$		u_2

3.

w_1	$\overline{w}_1$		w_1
u_1	a	b	c	w_1
u_1	$w_2\overline{w}_2w_2$		u_2

ou

w_1	$\overline{w}_1$	w_1		
w_1	a	b	c	u_2
u_1		$w_2\overline{w}_2w_2$		u_2

4.

w_1	$\overline{w}_1$	w_1		
u_1	a	$\overline{w}_1$	c	u_2
u_1	$w_2\overline{w}_2w_2$			u_2

Une analyse rapide nous assure que les cas qui apparaissent sur la même ligne ci-dessus sont équivalents. En effet, on passe de l'un à l'autre par l'opération $\bar{.}$, et il est immédiat de vérifier que $u \leftrightsquigarrow v \Leftrightarrow \overline{u} \leftrightsquigarrow \overline{v}$.

B.2.1 Premier et second sous-cas

Ces cas sont très simples. Pour le premier par exemple :
- $w_1 = u_1 w_2 \overline{w}_2 w_2 a \rightsquigarrow u_1 w_2 a$
- $u_1 w_2 u_2 = u_1 w_2 a \cdot \overline{w}_1 \cdot w_1 \rightsquigarrow^2 u_1 w_2 a \cdot \overline{u_1 w_2 a} \cdot u_1 w_2 a \rightsquigarrow u_1 w_2 a$

Et le second est tout à fait similaire :
- $w_1 = \overline{\overline{w}_1} = \overline{a w_2 \overline{w}_2 w_2 b} \rightsquigarrow \overline{a w_2 b}$
- $u_1 w_2 u_2 = w_1 \cdot a w_2 b \cdot w_1 \rightsquigarrow^2 \overline{a w_2 b} \cdot a w_2 b \cdot \overline{a w_2 b} \rightsquigarrow \overline{a w_2 b}$

B.2.2 Troisième sous-cas

w_1		$\overline{w}_1$		w_1
u_1	a	b	c	w_1
u_1	$w_2 \overline{w}_2 w_2$		u_2	

Trois configurations sont possibles :

1.

w_1				$\overline{w}_1$		w_1
u_1	w_2	$\overline{w}_2$	a	b	c	w_1
u_1	w_2	$\overline{w}_2$	w_2		u_2	

2.

w_1		$\overline{w}_1$				w_1
u_1	a	b	$\overline{w}_2$	w_2	c	w_1
u_1	w_2		$\overline{w}_2$	w_2	u_2	

3.

w_1			$\overline{w}_1$			w_1
u_1	w_2	a	b	w_2	c	w_1
u_1	w_2	$\overline{w}_2$		w_2	u_2	

Les deux premiers cas sont simples. Traitons le premier :

$$\begin{aligned}
u_1 w_2 \cdot u_2 &= u_1 \cdot w_2 \cdot c w_1 \\
&= u_1 a \cdot bc \cdot w_1 \\
&= u_1 a \cdot \overline{w}_1 \cdot w_1 \\
&= u_1 a \cdot \overline{u_1 w_2 \overline{w}_2 a} \cdot u_1 w_2 \overline{w}_2 a \\
&= u_1 a \cdot u_1 \cdot ab \cdot \overline{ab} \cdot a \cdot u_1 \cdot ab \cdot \overline{ab}\, a \\
&= u_1 \cdot a\overline{a}a \cdot b\, \overline{u_1 ab}\, u_1 ab \overline{ab}\, a \\
&\rightsquigarrow u_1 ab\, \overline{u_1 ab}\, u_1 ab \cdot \overline{ab}\, a \\
&\rightsquigarrow u_1 \cdot ab \cdot \overline{ab}\, a \\
&= u_1 w_2 \overline{w}_2 a = w_1
\end{aligned}$$

Le second :

$$\begin{aligned}
u_1 w_2 \cdot u_2 &= u_1 \cdot w_2 \cdot c \cdot w_1 \\
&= u_1 a \cdot bc \cdot w_1 \\
&= w_1 \cdot bc \cdot w_1 \\
&= \overline{b \overline{w}_2 w_2 c} \cdot bc \cdot \overline{b \overline{w}_2 w_2 c} \\
&= \overline{w_2 \cdot c} \cdot w_2 \cdot \overline{b}\, bc\, \overline{w_2 \cdot c} \cdot w_2 \cdot \overline{b} \\
&= \overline{abc}\, a \cdot b\overline{b}b \cdot c\, \overline{abc}\, ab\overline{b} \\
&\rightsquigarrow \overline{abc} \cdot abc \cdot \overline{abc} \cdot ab\overline{b} \\
&\rightsquigarrow \overline{abc} \cdot ab\overline{b} = \overline{w_2 c} w_2 \overline{b} = \overline{b \overline{w}_2 w_2 c} = w_1
\end{aligned}$$

Le troisième en revanche est plus compliqué. On commence par poser l'équation

$$u_1\overline{b}\overline{a}\ a = u_1w_2a = w_1 = \overline{\overline{w_1}} = \overline{c}\ \overline{w}_2\ \overline{b} = \overline{c}\ ab\ \overline{b} \tag{33}$$

On y applique le lemme de décomposition, et dans les deux cas on re-décompose la deuxième équation, ce qui nous donne les quatre cas :

1. $u_1 = \overline{c}x \wedge ab\overline{b} = x\overline{b}\overline{a}a$
   (a) $u_1 = \overline{c}x \wedge a = xy \wedge \overline{b}\overline{a}a = yb\overline{b}$. En replaçant a par xy dans la dernière équation on obtient :

   $$\overline{b}\overline{y}\ \overline{x}xy = yb\overline{b}$$

   Comme $|\overline{b}\overline{y}| = |b| + |y| = |yb|$, on en déduit :

   $$\overline{x}xy = \overline{b} \text{ c'est à dire } b = \overline{y}\ \overline{x}x$$

   On a donc

   $$\begin{array}{rcll} u_1w_2u_2 & = & \overline{c}x\overline{b}\overline{a}cw_1 & \text{et } w_1 = \overline{c}x\overline{x}xy\overline{y}\ \overline{x}xy \rightsquigarrow \overline{c}xy\overline{y}\ \overline{x}xy \rightsquigarrow \overline{c}xy \\ & = & \overline{c}x\overline{x}xy\overline{y}\ \overline{x}c\overline{c}x\overline{x}xy\overline{y}\ \overline{x}xy & \\ & \rightsquigarrow^2 & \overline{c}xy\overline{y}\ \overline{x}c\overline{c}xy\overline{y}\ \overline{x}xy & \\ & \rightsquigarrow & \overline{c}xy\overline{y}\ \overline{x}c\overline{c}xy & \\ & \rightsquigarrow & \overline{c}xy & \end{array}$$

   (b) $u_1 = \overline{c}x \wedge x = ay \wedge b\overline{b} = y\overline{b}\overline{a}a$. En passant la deuxième équation par l'opération $\overline{\ }$, on a :

   $$\overline{a}a\overline{y\overline{b}} = \overline{b\overline{b}} = b\overline{b} = y\overline{b}\overline{a}a.$$

   On reconnaît l'hypothèse du lemme (x), qui nous assure [(xiv)] que $\exists t : y\overline{b} = \overline{a}a \cdot t\overline{t}$. On a donc :

   $$\begin{array}{rcll} u_1w_2u_2 & = & u_1w_2cu_1w_2a & \text{et } w_1 = \overline{c}ay\overline{b}\overline{a}a = \overline{c}a\overline{a}at\overline{t}\overline{a}a \rightsquigarrow \overline{c}at\overline{t}\overline{a}a \\ & = & \overline{c}ay\overline{b}\overline{a}c\overline{c}ay\overline{b}\overline{a}a & \\ & = & \overline{c}a\overline{a}at\overline{t}\overline{a}c\overline{c}a\overline{a}at\overline{t}\overline{a}a & \\ & \rightsquigarrow^2 & \overline{c}at\overline{t}\overline{a}c\overline{c}at\overline{t}\overline{a}a & \\ & \rightsquigarrow & \overline{c}at\overline{t}\overline{a}a & \end{array}$$

2. $\overline{c} = u_1x \wedge \overline{b}\overline{a}a = xab\overline{b}$
   (a) $\overline{c} = u_1x \wedge \overline{b} = xy \wedge ab\overline{b} = y\overline{a}a$. En remplaçant b on a $a\overline{y}\ \overline{x}xy = y\overline{a}a$. Comme $|a\overline{y}| = |y\overline{a}|$ on en déduit que $a = \overline{x}xy$. On peut donc remplacer :

   $$\begin{array}{rcll} u_1w_2u_2 & = & u_1w_2cu_1w_2a & \text{et } w_1 = u_1w_2a = u_1\overline{b}\overline{a}a \\ & = & u_1xy\overline{y}\ \overline{x}x\overline{x}\ \overline{u}_1u_1xy\overline{y}\ \overline{x}x\overline{x}xy & = u_1xy\overline{y}\ \overline{x}x\overline{x}xy \\ & \rightsquigarrow^2 & u_1xy\overline{y}\ \overline{x}\overline{u}_1u_1xy\overline{y}\ \overline{x}xy & \rightsquigarrow u_1xy\overline{y}\ \overline{x}xy \rightsquigarrow u_1xy \\ & \rightsquigarrow & u_1xy\overline{y}\ \overline{x}\ \overline{u}_1u_1xy & \\ & \rightsquigarrow & u_1xy & \end{array}$$

(xiv). Comme $b\overline{b} = y\overline{b}\overline{a}a$, il est clair que $|b| = |y| + 2|a|$, et donc $|y\overline{b}| = 2|y| + 2|a| \geqslant 2|a|$.

(b) $\overline{c} = u_1 x \wedge x = \overline{b}y \wedge \overline{a}a = yab\overline{b}$. On remarque que :

$$yab\overline{b} = \overline{a}a = \overline{\overline{a}a} = \overline{yab\overline{b}} = b\overline{b}\overline{y}\overline{a}.$$

On applique le lemme (x) et on a $ya = b\overline{b}t\overline{t}$. Cela nous donne donc :

$$\begin{array}{rcl} u_1w_2u_2 & = & u_1w_2cu_1w_2a \\ & = & u_1\overline{b}\overline{a}\ \overline{y}b\overline{u}_1u_1\overline{b}\overline{a}a \\ & = & u_1\overline{b}t\overline{t}b\overline{b}b\overline{u}_1u_1\overline{b}b\overline{b}\overline{y}\overline{a} \\ & \rightsquigarrow^2 & u_1\overline{b}t\overline{t}b\overline{u}_1u_1\overline{b}t\overline{t}b\overline{b} \\ & \rightsquigarrow & u_1\overline{b}t\overline{t}b\overline{b} \end{array} \quad \text{et} \quad \begin{array}{rcl} w_1 & = & u_1\overline{b}\overline{a}a = u_1\overline{b}b\overline{b}\overline{y}\overline{a} \\ & = & u_1\overline{b}b\overline{b}t\overline{t}b\overline{b} \\ & \rightsquigarrow & u_1\overline{b}t\overline{t}b\overline{b} \end{array}$$

B.2.3 Quatrième sous-cas

w_1		$\overline{w}_1$	w_1	
u_1	a	$\overline{w}_1$	c	u_2
u_1	$w_2\overline{w}_2w_2$			u_2

On peut à nouveau énumérer différentes configurations :

1.

w_1		$\overline{w}_1$		w_1		
u_1	a	b	c	d	w_2	u_2
u_1	w_2		$\overline{w}_2$		w_2	u_2

ou

w_1			$\overline{w}_1$		w_1	
u_1	w_2	a	b	c	d	u_2
u_1	w_2	$\overline{w}_2$		w_2		u_2

2.

w_1		$\overline{w}_1$			w_1	
u_1	a	b	$\overline{w}_2$	c	d	u_2
u_1	w_2		$\overline{w}_2$	w_2		u_2

Comme précédemment, on voit que les deux cas sur la ligne 1. sont équivalents. Traitons le premier cas. Posons l'équation

$$\overline{c}\overline{b} = w_1 = dw_2u_2 = (d\overline{d}\overline{c})u_2 \tag{34}$$

On la décompose (parenthésé comme ci-dessus) :

1. $\overline{c} = d\overline{d}\overline{c}x \wedge u_2 = x\overline{b}$. On déduit de la première équation que $|c| = 2|d| + |c| + |x|$, et donc que $d = x = \epsilon$. Par conséquent $u_2 = \overline{b}$

$$u_1w_2u_2 = u_1ab\overline{b} = \overline{c}\overline{b}b\overline{b} \rightsquigarrow \overline{c}\overline{b} = w_1.$$

2. $d\overline{d}\overline{c} = \overline{c}x \wedge \overline{b} = xu_2$. En mélangeant ces deux équation et en se rappelant que $ab = \overline{d}\overline{c}$, on arrive à :

$$da\overline{u}_2\overline{x} = dab = d\overline{d}\overline{c} = \overline{c}x.$$

Comme $|x| = |\overline{x}|$, on déduit $da\overline{u}_2 = \overline{c} \wedge \overline{x} = x$. En repassant cela dans la seconde équation on remarque que $bu_2 = \overline{u}_2\overline{x}u_2 = \overline{u}_2xu_2 = \overline{u}_2\overline{b} = \overline{bu_2}$. Le lemme (ix) nous permet d'écrire $bu_2 = y\overline{y}$, et donc :

$$u_1w_2u_2 = u_1abu_2 = dabu_2bu_2 = day\overline{y}y\overline{y} \rightsquigarrow day\overline{y} = dabu_2 = w_1$$

L'autre cas se traite plus rapidement :

w_1			$\overline{w}_1$		w_1	
u_1	a	b	$\overline{w}_2$	c	d	u_2
u_1	w_2		$\overline{w}_2$	w_2		u_2

$$u_1w_2u_2 = u_1abu_2 = \overline{c}w_2\overline{b}bu_2 = \overline{c}ab\overline{b}bu_2 \rightsquigarrow \overline{c}abu_2 = \overline{c}cdu_2 = \overline{c}c\overline{c}w_2\overline{b} \rightsquigarrow \overline{c}w_2\overline{b} = w_1$$

B.3 Second Cas

On étudie maintenant le cas où les deux motifs sont décalés c'est à dire $u_1w_1\overline{w}_1w_1 = w_2\overline{w}_2w_2u_2$. À nouveau, différents cas se présentent, et on traitera les cas simples aussitôt :

1. $u_1w_1 = w_2\overline{w}_2w_2x \wedge u_2 = x\overline{w}_1w_1$. On peut tout de suite remarquer que $u_1w_1 \rightsquigarrow w_2x$.

(a)

u_1		w_1		$\overline{w}_1w_1$
$w_2\overline{w}_2$	a	b	x	$\overline{w}_1w_1$
$w_2\overline{w}_2$	w_2	u_2		

$w_2u_2 = abx\overline{x}\overline{b}bx \rightsquigarrow abx = w_2x.$

(b)

u_1		w_1			$\overline{w}_1w_1$
w_2	a	b	w_2	x	$\overline{w}_1w_1$
w_2	$\overline{w}_2$	w_2	u_2		

$w_2u_2 = \overline{b}\overline{a}x\overline{x}ab\overline{b}b\overline{b}\overline{a}x \rightsquigarrow^2 \overline{b}\overline{a}x = w_2x$

(c)

u_1	w_1			$\overline{w}_1w_1$
u_1	a	$\overline{w}_2w_2$	x	$\overline{w}_1w_1$
w_2	$\overline{w}_2w_2$	u_2		

$w_2u_2 = u_1ax\overline{x}\,\overline{a}\,\overline{u}_1u_1a\overline{a}a\overline{a}\,\overline{u}_1u_1ax \rightsquigarrow^3 u_1ax = w_2x$

2. $w_2\overline{w}_2w_2 = u_1w_1x \wedge \overline{w}_1 = xy \wedge u_2 = yw_1$:

(a)

u_1		w_1	$\overline{w}_1$		w_1
$w_2\overline{w}_2$	a	w_1	x	y	w_1
$w_2\overline{w}_2$	w_2		u_2		

$$\begin{aligned} w_2u_2 &= aw_1xyw_1 = aw_1\overline{w}_1w_1 \rightsquigarrow aw_1 \\ u_1w_1 &= w_2\overline{w}_2aw_1 = aw_1x\overline{x}\,\overline{aw_1}aw_1 \\ &= a\overline{y}\,\overline{x}x\overline{x}\,\overline{aw_1}aw_1 \\ &\rightsquigarrow a\overline{y}\,\overline{x}\,\overline{aw_1}aw_1 \\ &= aw_1\overline{aw_1}aw_1 \rightsquigarrow aw_1. \end{aligned}$$

(b)

u_1		w_1		$\overline{w}_1$		w_1
w_2	a	b	c	x	y	w_1
w_2	$\overline{w}_2$	w_2		u_2		

(c)

u_1	w_1			$\overline{w}_1$		w_1
u_1	a	$\overline{w}_2$	b	x	y	w_1
w_2	$\overline{w}_2$	w_2		u_2		

$$\begin{aligned} w_2u_2 &= w_2yw_1 = b\ xy\ a\overline{w}_2b \\ &= b\overline{b}w_2\overline{a}a\overline{a}\overline{u}_1b = b\overline{b}bx\overline{a}a\overline{a}\,\overline{u}_1b \\ &\rightsquigarrow^2 bx\overline{a}\overline{u}_1b = u_1a\overline{w}_2b = u_1w_1. \end{aligned}$$

(d)

u_1	w_1		$\overline{w}_1$			w_1
u_1	a	b	c	w_2	y	w_1
w_2		$\overline{w}_2$		w_2	u_2	

3. $w_1 = xu_2 \wedge u_1w_1\overline{w}_1x = w_2\overline{w}_2w_2$:

(a)

u_1		$w_1\overline{w}_1$		w_1
$w_2\overline{w}_2$	a	$w_1\overline{w}_1$	x	u_2
$w_2\overline{w}_2$	w_2			u_2

$$\begin{aligned} w_2u_2 &= aw_1\overline{w}_1xu_2 = aw_1\overline{w}_1w_1 \rightsquigarrow aw_1 \\ u_1w_1 &= w_2\overline{w}_2aw_1 = aw_1\overline{w}_1x\overline{x}w_1\overline{w}_1\overline{a}aw_1 \\ &= aw_1\overline{w}_1x\overline{x}xu_2\overline{w}_1\overline{a}aw_1 \rightsquigarrow aw_1\overline{w}_1xu_2\overline{w}_1\overline{a}aw_1 \\ &= aw_1\overline{w}_1w_1\overline{w}_1\overline{a}aw_1 \rightsquigarrow^2 aw_1. \end{aligned}$$

(b)

u_1		w_1		$\overline{w}_1$	w_1	
w_2	a	b	c	$\overline{w}_1$	x	u_2
w_2	$\overline{w}_2$		w_2			u_2

$$\begin{aligned} u_1w_1 &= w_2abc = w_2\overline{w}_2c = c\overline{u}_2\ \overline{x}x\overline{x}xu_2\overline{c}c \\ &\rightsquigarrow c\overline{u}_2\ \overline{x}xu_2\overline{c}c = c\overline{w}_1bc\overline{c}c \rightsquigarrow c\overline{w}_1bc = c\overline{w}_1xu_2 = w_2u_2. \end{aligned}$$

(c)

u_1		w_1	$\overline{w}_1$		w_1	
w_2	a	w_1	b	c	x	u_2
w_2	$\overline{w}_2$			w_2		u_2

(d)

u_1	w_1		$\overline{w}_1$		w_1	
u_1	a	b	c	d	x	u_2
w_2		$\overline{w}_2$		w_2		u_2

B.3.1 Cas 2.(b)

u_1		w_1		$\overline{w}_1$		w_1
w_2	a	b	c	x	y	w_1
w_2	$\overline{w}_2$		w_2		u_2	

Pour résoudre ce cas, on va d'abord prouver le lemme suivant :

Lemme xii:

$$\forall a, b, c, d \in X^*, \overline{a}ab = cd\overline{d} \Rightarrow abd \leftrightsquigarrow acd. \qquad \blacksquare$$

Preuve On va procéder par induction sur a, et faire une étude de cas :

- Si $a = \epsilon$ alors $b = cd\overline{d}$ et donc $abd = cd\overline{d}d \rightsquigarrow cd = acd$;
- sinon, décomposons :

1.

$\overline{a}a$	b	
$\overline{a}a$	x	$d\overline{d}$
c		$d\overline{d}$

$$\begin{aligned} abd &= axd\overline{d}d \rightsquigarrow axd \\ acd &= a\overline{a}axd \rightsquigarrow axd. \end{aligned}$$

2.

$\overline{a}$	a		b	
$\overline{a}$	x	y	z	$\overline{d}$
c		d		$\overline{d}$

$$\begin{aligned} abd &= xyz\overline{d}d = xd\overline{d}d \rightsquigarrow xd = xyz \\ acd &= a\overline{a}xyz = a\overline{a}az \rightsquigarrow az = xyz. \end{aligned}$$

3.

$\overline{a}$	a			b
$\overline{a}$	x	d	y	b
c		d		$\overline{d}$

$$\begin{aligned} abd &= x\overline{b}\overline{y}yb\overline{b}\overline{y} \rightsquigarrow x\overline{b}\overline{y} \\ acd &= x\overline{b}\overline{y}y\overline{y}yb\overline{x}x\overline{b}\overline{y} \rightsquigarrow x\overline{b}\overline{y}yb\overline{x}x\overline{b}\overline{y} \rightsquigarrow x\overline{b}\overline{y}. \end{aligned}$$

4.

$\overline{a}$		a		b
c	x	a	y	$\overline{d}$
c		d		$\overline{d}$

$$\begin{aligned} abd &= \overline{x}\,\overline{c}y\overline{y}cx\overline{x}x\overline{x}\,\overline{c}y \rightsquigarrow \overline{x}\,\overline{c}y\overline{y}cx\overline{x}\,\overline{c}y \rightsquigarrow \overline{x}\,\overline{c}y \\ acd &= \overline{x}\overline{c}cx\overline{x}\,\overline{c}y \rightsquigarrow \overline{x}\,\overline{c}y. \end{aligned}$$

5.

$\overline{a}$		a		b
c	x	y	z	b
c	d		$\overline{d}$	

Ce dernier cas est plus subtil : on remarque que

$$\overline{z}zb = \overline{z}\overline{d} = \overline{z}\,\overline{y}\,\overline{x} = \overline{a}\,\overline{x} = cx\overline{x}.$$

- Si $y = \epsilon$, alors $\overline{a} = cd \wedge \overline{d} = ab$; d'où l'on peut déduire $b = c = \epsilon$ et donc $abd = acd$.
- Sinon, $|y| > 0$ et donc $|z| < |z| + |y| = |a|$, et l'on peut appliquer l'hypothèse d'induction, qui nous indique que $zbx \leftrightsquigarrow zcx$. Or $abd = y \cdot zbx \cdot y$ et $acd = y \cdot zcx \cdot y$, et par conséquent $abd \leftrightsquigarrow acd$. $\square$

Avec ce lemme, la preuve devient aisée : en effet on peut remarquer que

$$\begin{cases} aw_1\overline{w}_1 = abcxy = \overline{w}_2 w_2 y \\ u_1 w_1 = w_2 a w_1 \\ w_2 u_2 = w_2 y w_1. \end{cases}$$

On applique donc le lemme qui nous donne exactement le résultat attendu.

B.3.2 Cas 2.(d)

Rappelons les hypothèses de ce cas :

$$\begin{cases} w_1 = ab = \overline{yw}_2\overline{c} \\ w_2 = u_1 a = \overline{c}\overline{b} \\ u_2 = y w_1 \end{cases}$$

On va décomposer la seconde équation.

– $\overline{c} = u_1 x \wedge a = x\overline{b}$: dans ce cas la première équation devient (en appliquant $\overline{\cdot}$) $\overline{b} \cdot b\overline{x} = \overline{x}\,\overline{u}_1 u_1 x \cdot \overline{b} \cdot y$. On peut donc appliquer le lemme (xi) et écrire

$$\begin{cases} b\overline{x} = tsy \\ \overline{x}\,\overline{u}_1 u_1 x = st \\ \overline{x}\,\overline{u}_1 u_1 x\overline{b} = \overline{b}ts. \end{cases}$$

On remarque alors que la dernière équation peut se réécrire :

$$\overline{x}\,\overline{u}_1 u_1 \cdot x\overline{b} = \overline{x}\,\overline{u}_1 u_1 \overline{tsy} = \overline{x}\,\overline{u}_1 u_1 \overline{y} \cdot \overline{st} = \overline{b} \cdot ts.$$

Cela nous permet d'établir que $ts = \overline{ts}$, et donc que pour un certain p on a $ts = p\overline{p}$. De là :

$$w_2 u_2 = u_1 \overline{y} p \overline{p} y \overline{y} p \overline{p} b \rightsquigarrow u_1 \overline{y} p \overline{p} b = u_1 x \overline{b} b = u_1 w_1.$$

– $u_1 = \overline{c}x \wedge \overline{b} = xa$: ici la première équation devient, après passage au converse, $xa \cdot \overline{a} = c\overline{c} \cdot xa \cdot y$. On applique à nouveau le même lemme pour obtenir

$$\begin{cases} \overline{a} = tsy \\ c\overline{c} = st \\ c\overline{c}xa = xats. \end{cases}$$

La dernière équation peut se réécrire $c\overline{c}x\overline{y} \cdot \overline{st} = xa \cdot ts$, d'où l'on déduit $ts = \overline{ts}$, et donc $ts = p\overline{p}$ pour un certain p. On arrive donc à la réduction suivante :

$$w_2 u_2 = \overline{c}x\overline{y}p\overline{p}y\overline{y}p\overline{p}p\overline{p}yx \rightsquigarrow \overline{c}x\overline{y}p\overline{p}p\overline{p}yx = u_1 w_1.$$

B.3.3 Cas 3.(c)

On se place dans un cas où :

$$\begin{cases} u_1 = w_2 a \\ w_2 = \overline{b}\overline{w}_1\,\overline{a} = cd \\ w_1 = \overline{c}\overline{b} = du_2 \end{cases}$$

On commence par décomposer l'équation :

$$\overline{b}bc\overline{a} = cd.$$

– $\overline{b}bc = ct \wedge d = t\overline{a}$: on décompose à nouveau la première équation :
  – $\overline{b}b = cx \wedge t = xc$: on décompose une dernière fois :
    – $\overline{b} = cy \wedge x = yb = y\overline{yc}$: dans ce cas l' équation définissant w_1 devient

$$\overline{c}cy = y\overline{yc}c\overline{a}u_2.$$

Une analyse rapide des tailles des deux côtés de l'équation nous assure que $y = a = u_2 = \epsilon$, et par la suite que $\overline{b} = cy = c$ et $x = yb = \overline{c}$. Subséquemment :

$$u_1w_1 = w_2aw_1 = cxc\overline{a}a\overline{c}c = c\overline{c}c\overline{c}c \rightsquigarrow c\overline{c}c = cxc\overline{a}u_2 = w_2u_2.$$

  – $b = yx \wedge c = \overline{b}y = \overline{xy}y : \overline{c}\overline{b} = du_2$ devient $\overline{y}yx\overline{x} \cdot \overline{y} = x\overline{x}\,\overline{y}y \cdot u_2$. On en déduit que $x\overline{x}\,\overline{y}y = z\overline{z}$, c'est à dire $xc = z\overline{z}$. Donc $u_1w_1 = w_2aw_1 = czz\overline{a}az\overline{z}au_2 \rightsquigarrow czz\overline{a}u_2 = cxc\overline{a}u_2 = w_2u_2$.
– $c = \overline{b}bx \wedge c = xt$: l'équation sur w_1 nous dit que $\overline{t}x\overline{b} = t\overline{a}u_2$ et donc en particulier que $\overline{t} = t$, c'est à dire $t = z\overline{z}$. On en déduit :

$$u_1w_1 = cdadu_2 = czz\,\overline{a}z\overline{z}\,\overline{a}u_2 \rightsquigarrow czz\overline{a}u_2 = cdu_2 = w_2u_2.$$

– $c = \overline{b}bct \wedge \overline{a} = td$: ici la première équation nous indique que $b = t = \epsilon$. On en déduit immédiatement :

$$u_1w_1 = cdadu_2 = c\overline{a}a\overline{a}u_2 \rightsquigarrow c\overline{a}u_2 = cdu_2 = w_2u_2.$$

B.3.4 Cas 3.(d)

On part des hypothèses suivantes : $\begin{cases} w_1 = ab = eu_2 = \overline{d}\overline{c} \\ w_2 = u_1a = de = \overline{c}\overline{b}. \end{cases}$

On décompose $ab = eu_2$.
– $a = ex \wedge u_2 = xb$: on peut remarquer que $u_1ex = de$ et donc après application du converse $\overline{e}\overline{d} = \overline{x}\,\overline{e}\,\overline{u}_1$, on peut donc appliquer le lemme (xi). On obtient $\begin{cases} \overline{d}{=}\overline{t}\overline{s}\,\overline{u}_1 \text{ soit } d{=}u_1st \\ \overline{x}{=}\overline{s}\overline{t} \quad \text{soit } x{=}ts \\ \overline{x}\,\overline{e}{=}\overline{e}\overline{t}\overline{s} \quad \text{soit } ex{=}ste. \end{cases}$

En remplaçant dans l'équation $eu_2 = \overline{d}\overline{c}$ on a $exb = st \cdot eb = \overline{t}\overline{s} \cdot \overline{u}_1\overline{c}$ et donc $st = \overline{s}\overline{t}$ soit $st = z\overline{z}$. Cela donne :

$$w_2u_2 = u_1z\overline{z}z\overline{z}eb \rightsquigarrow u_1z\overline{z}eb = u_1exb = u_1w_1.$$

– $e = ax \wedge b = xu_2$: on a $u_1a = de = dax$, donc en passant par le converse et en appliquant le même lemme que ci-dessus on obtient : $\begin{cases} \overline{u}_1{=}\overline{t}\overline{s}\,\overline{d} \text{ soit } u_1{=}dst \\ \overline{x}{=}\overline{s}\overline{t} \quad \text{soit } x{=}ts \\ \overline{x}\,\overline{a}{=}\overline{a}\overline{t}\overline{s} \quad \text{soit } ax{=}sta. \end{cases}$ Or $u_1a = \overline{c}\overline{b}$ donc $dsta = dax = \overline{c}\,\overline{u}_2\,\overline{x}$, d'où l'on infère $x = \overline{x}$ et donc $x = z\overline{z}$. Par conséquent :

$$u_1w_1 = dstaxu_2 = daxxu_2 = dazz\overline{z}z\overline{z}u_2 \rightsquigarrow daxu_2 = w_2u_2.$$