# Abstractions de programmes pour l'étude de la terminaison et de la complexité.

GRIENENBERGER Emilie

27 septembre 2016

## Table des matières



# 1 Introduction

**Contexte scientifique.** Le problème de la *terminaison* de programmes est un problème classique dans le domaine de l'informatique :
*Étant donné un programme, termine-t-il sur toute entrée ?*

Ce problème est *indécidable*, c'est-à-dire qu'il n'existe aucun algorithme qui étant donné un programme $\mathcal{P}$ peut décider s'il termine. Une preuve classique de ce résultat se fait par un raisonnement diagonal : supposons par l'absurde qu'il existe un programme `HALT` qui prend en entrée un programme $P$, et répond `vrai` s'il termine et `faux` sinon, en temps fini. Alors construisons un programme `DIAG` qui prend en entrée un programme $P$ de la manière suivante :

— si `HALT`($P$), alors il boucle à l'infini,

— sinon, il renvoie `vrai`.

Par cette construction, si `DIAG` termine sur toutes ses entrées, alors `HALT`(`DIAG`) renvoie `vrai`. Ainsi, `DIAG`(`DIAG`) boucle à l'infini. On en déduit donc que `DIAG` ne termine pas sur toutes ses entrées, ce qui est absurde. On vient donc de prouver que le problème est indécidable.

Le problème ne peut donc être résolu totalement, et seules des solutions imparfaites peuvent être mises au point. De multiples méthodes se sont développées pour tenter d'atteindre les limites de cette indécidabilité. Il existe notamment deux approches pour traiter de la terminaison de programmes.

La première est de se restreindre à certaines familles de programmes où le problème est décidable, en limitant le langage de programmation par exemple. Un exemple quelque peu extrême est qu'un programme ne pouvant effectuer que des affectations termine forcément, la terminaison est donc décidable pour cette famille de programmes.

La deuxième approche, celle qui va être utilisée dans ce rapport, ne restreint pas les programmes étudiés, mais les informations étudiées au sein des programmes. Elle se base sur une simplification du programme : il est modélisé par un objet plus abstrait dont davantage de propriétés sont décidables, un automate par exemple. Évidemment, il y a une perte d'information lors de ce procédé d'abstraction : c'est cette perte qui fait que l'indécidabilité est contournée. On déduit des propriétés de l'objet abstrait certaines propriétés du programme. Par exemple, en définissant un concept de terminaison décidable pour ces objets abstraits, la méthode peut être *correcte*, ce qui signifie que si l'objet termine, son programme associé termine, ou *complète*, ce qui signifie que si le programme termine, alors l'objet abstrait associé termine également. La méthode ne peut être correcte et complète, sinon par l'absurde le problème de terminaison est décidable également. Le but est donc de se rapprocher le plus possible de la correction et de la complétude. On remarque que c'est à cette étape là que la perte d'information due à l'abstraction provoque des imprécisions.

La recherche sur ces méthodes est très riche. En effet, malgré l'indécidabilité du problème, pouvoir s'assurer de la terminaison de programmes est crucial dans de nombreux domaines. La *vérification de programmes* est en un. Il s'agit de pouvoir garantir qu'un programme répond à ses *spécifications*, c'est-à-dire que son comportement est celui qui est attendu de lui. À cette fin sont nécessaires l'assurance que le programme termine, et la *correction partielle*, qui établit par exemple les relations voulues entre entrées et sorties du programme étudié. Ces résultats sont primordiaux : dans le cas d'optimisation et de génération automatiques de programmes par exemple, il faut pouvoir assurer les propriétés du programme produit.

Prenons un autre exemple, celui des *systèmes réactifs*. Un système réactif est un système en interaction permanente avec son environnement. L'environnement agit, et le système doit réagir instantanément. Des exemples de systèmes réactifs sont les systèmes utilisés dans le traitement du signal ou les processus industriels. Il est nécessaire de pouvoir assurer que chaque calcul qu'un tel système est susceptible de mettre en œuvre termine.

Ainsi, l'objectif vers lequel les recherches tendent est une méthode automatisée qui puisse donner un *certificat*, c'est-à-dire un témoin concret, de la terminaison des programmes.

Plus précisément, les programmes étudiés dans ce rapport sont les programmes impératifs. Dans ce cadre, une notion fondamentale pour la preuve de terminaison est celle de *fonction de rang*. Il s'agit d'une fonction prenant en arguments les paramètres du programme et à valeurs dans un ensemble *bien fondé*. Un ensemble bien fondé est un ensemble qui n'admet aucune suite infinie strictement décroissante, comme $\mathbb{N}$. Cette fonction doit strictement diminuer à chaque étape du programme (par exemple à chaque exécution d'une boucle). Ce concept a été introduit dès 1967 par Robert Floyd dans [8]. Une fonction de rang est un témoin de la terminaison d'un programme. En effet, supposons donnés un programme et une fonction de rang pour ce programme. À chaque exécution du programme correspond une suite de valeurs de la fonction de rang, qui est une suite strictement décroissante par définition. Supposons que le programme ne termine pas, il existe une exécution infinie. Cependant, il ne peut y avoir de suite décroissante infinie de valeurs de la fonction de rang, donc c'est absurde.

Les recherches sur l'automatisation de la recherche de fonctions de rang pour les programmes impératifs sont ainsi très nombreuses, et progressent énormément. Elles se basent généralement sur l'abstrastion du programme étudié en *graphe de flot*, c'est-à-dire un automate représentant le programme : les états ou *points de contrôle* marquent les étapes de l'exécution du programme, et les transitions contiennent les informations (conditions, affectations,...) pertinentes au passage d'une étape à l'autre. Ce concept est expliqué dans [8]. Pour automatiser le processus de recherche de fonctions de rang sur ces objets, un

pré-traitement est effectué sur cet automate, par exemple une génération d'*invariants*, c'est-à-dire un ensemble de conditions sur les variables associé à chaque point de contrôle. Ce conditions sont vérifiées pour tout passage par ce point de contrôle. Ensuite, la fonction de rang elle-même est recherchée. Des résultats ont d'abord été établis pour les fonctions de rang monodimensionnelles, applicable sur les programmes entièrement déterministes ne contenant que des boucles simples (sans imbrications de boucles), avec par exemple [11]. Ces méthodes ont ensuite été étendues à des fonctions de rang multidimensionnelles. Le concept de fonctions de rang multidimensionnelles a donné lieu à de nombreuses autres recherches, par exemple la méthode étudiée lors de ce stage, [1], ou [4], qui s'inscrit dans la continuation de travail effectué dans [7]. En effet, elle permettent de prouver la terminaison pour une classe plus large de programmes. Bien sûr, d'autres approches existent, par exemple l'approche de [5], visant à trouver des ensembles de fonctions de rangs fonctionnant sur plusieurs parties différentes du programme ou plusieurs exécutions. En effet, parfois, deux exécutions passent par les mêmes boucles mais ont des comportements si différents qu'on ne peut trouver un fonction de rang générale pour toute les exécutions, mais seulement plusieurs fonctions telles que toutes les exécutions possibles sont couvertes par au moins l'une d'entre elles.

Il existe une autre méthode automtique pour prouver la terminaison des programmes impératifs, plus originale étant donné qu'elle était utilisée initialement pour la terminaison de programmes fonctionnels uniquement. Il s'agit de la *Size-Change Termination*. Elle nait dans [9], et son principe est le suivant : si les arguments d'une fonction sont à valeurs dans un ensemble bien fondé, et qu'à chaque appel de fonction ils décroissent strictement, alors le nombre d'appels de fonctions est fini. Elle fonctionne en modélisant les fonctions par un *size-change graph*, objet ne représentant que les décroissances de ces paramètres au fil des appels de fonctions. De nombreux travaux ont étendus et amélioré cette méthode. On citera notamment [10], utilisant un nouveau modèle de graphes pour modéliser les suites d'appels récursifs, [3] prouve quant à lui que les size-change graphs peuvent également prendre en compte les égalités entre paramètres et, dans une certaine mesure, les conditions sur les variables exprimées dans le programme. Enfin, [2] prouve qu'en utilisant un certain type de formules logiques, il est possible de capter les changements constants et affines de paramètres, et plus d'informations contextuelles. Cette technique se rapproche de la recherche de fonctions de rang, et montre que la Size-Change Termination est pertinente également dans le domaine des programmes impératifs.

**Sujet de recherche.** L'objectif du stage est de comparer deux méthodes de preuve de terminaison de programmes.

La première utilise des Integer-Interpreted Automata, un type de graphe de flot.

La deuxième utilise des Size-Change Abstractions.

**Contributions.** On observe que l'Integer-Interpreted Automaton est plus général quant à la preuve de terminaison, mais le Size-Change Abstraction est parfois plus précis pour le calcul de complexité, car ils arrivent à capter les complexités rationnelles. Pour étudier ces cas particuliers, on a étudié les automates max-plus : ils ont un lien pertinent avec le modèle Size-Change Abstraction et on a de multiples résultats sur ces objets. On sait notamment que par une méthode d'abstraction, décrite dans [6], telle que si le Size-Change Abstraction termine, les deux objets associés ont la même complexité. Ici, on étend ce résultat en montrant que pour les terminaisons d'un Size-Change Abstraction et d'un automate max-plus associé par cette méthode d'abstraction sont équivalentes. L'autre résultat est le fait que les automates max-plus peuvent atteindre toute complexité rationnelle, ce qui va justifier le fait qu'il existe en effet des programmes de toute complexité rationnelle. Ainsi, le Size-Change Abstraction est en effet plus fin au niveau de la complexité pour une classe de programmes.

# 2 Modèles étudiés : définitions et exemples

Dans cette section, on définit les deux abstractions de programmes étudiées : Integer-Interpreted Automaton, Size-Change Abstraction.

## 2.1 Integer Interpreted Automata

On introduit les Integer-Interpreted Automata, qui sont des graphes de flot d'un programme. Les Figures 1 et 2 représentent des abstractions de programmes en Integer-Interpreted Automaton. On

représente par une flèche entrante les états initiaux, et par un double cercle les états finaux. Chaque état, appelé point de contrôle, correspond à un instant de l'exécution du programme. Chaque transition est étiquetée par deux éléments :

— Au-dessus du trait, la garde représente les conditions devant être remplies pour passer la transition (par exemple elle peut représenter les conditions d'un `if`).

— Au-dessous du trait, l'action représente les affectations effectuées par la transition.

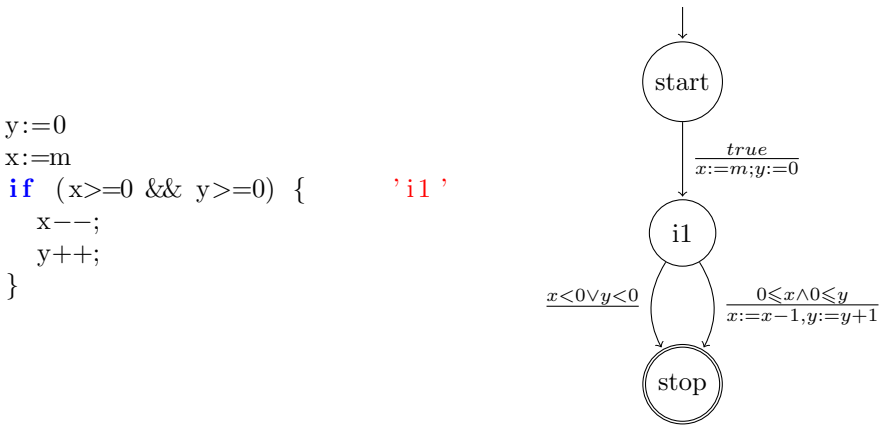



FIGURE 1 – Exemple d'abstraction en Integer-Interpreted Automaton

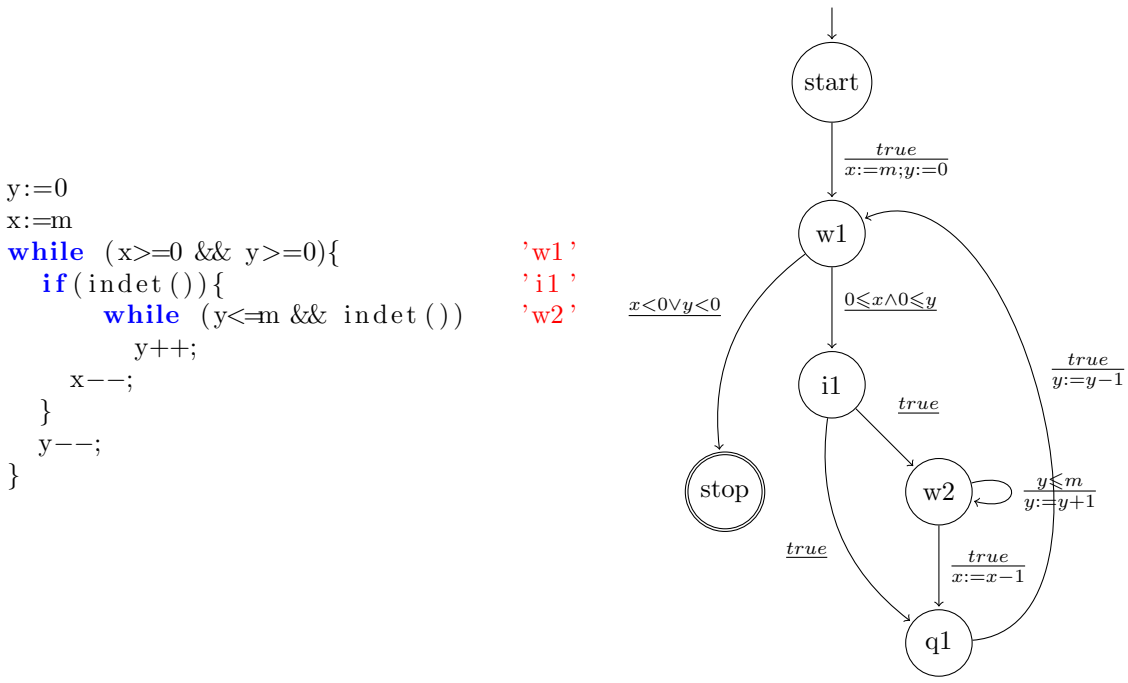



FIGURE 2 – Exemple d'abstraction avec conditions indéterminées en Integer-Interpreted Automaton

Seules les conditions affines d'un programme sont traduites dans ce modèle : si le programme contient des conditions polynomiales non affines ou un tirage aléatoire, il ne rentre plus dans ce cadre restreint. Ces conditions sont abstraites en `indet()` et sont traitées de manière non-déterministe par l'abstraction.

**Définition 1** (Integer-Interpreted Automaton)**.** *Un Integer-Interpreted Automaton est défini par un uplet* $(\mathcal{K}, \chi, k_i, k_f, \mathcal{T})$ *où :*

— $\mathcal{K}$ *est un ensemble fini d'états dits points de contrôle.*

— $\chi = \{x_1, ..., x_{|\chi|}\}$ *est un ensemble fini de variables.*
— $k_i \in \mathcal{K}$ *est l'état initial.*
— $k_f \in \mathcal{K}$ *est l'état final.*
— $\mathcal{T}$ *est un ensemble fini de transitions de la forme* $(k, g, a, k')$ *où :*
  — $k, k' \in \mathcal{K}$,
  — $g : \mathbb{Z}^{|\chi|} \to \mathbb{B} = \{true, false\}$, *appelée garde, est une formule logique exprimée par un ensemble d'inégalités affines sur* $\chi$, *donc de la forme :* $\vec{x} \mapsto G_i\vec{x} + \vec{g}_i \geq 0$ *où :*
    — *pour tout* $1 \leq i \leq p$, $G_i$ *est une matrice de* $\mathbb{Z}^{|\chi|\times|\chi|}$.
    — *pour tout* $1 \leq i \leq p$, $\vec{g}_i$ *est un vecteur de* $\mathbb{Z}^{|\chi|}$.
  — $a : \mathbb{Z}^{|\chi|} \to \mathbb{Z}^{|\chi|}$, *appelée action, est de la forme* $\vec{y} = A\vec{x} + \vec{a}$ *où :*
    — $A$ *est une matrice de* $\mathbb{Z}^{|\chi|\times|\chi|}$.
    — $\vec{a}$ *est un vecteur de* $\mathbb{Z}^{|\chi|}$.

On définit ainsi uniquement des gardes sous la forme de conjonctions d'inégalités affines. Cela englobe le cas des gardes disjonctives : pour une garde de la forme $G \vee G'$, on dédouble la transition en deux transitions, l'une ayant pour garde $G$ et l'autre ayant pour garde $G'$. On supposera que tous les états d'un Integer-Interpreted Automaton sont accessibles et co-accessibles, ce qui sera toujours le cas lors de l'abstraction d'un programme.

On définit la sémantique de ces objets avec la notion de *valuations*. Une valuation est une fonction de $\chi = \{x_1, ..., x_{|\chi|}\}$ dans $\mathbb{Z}$. Une valuation $\nu$ est de plus *bornée* par un entier $N$ si pour toute variable $x$, $\nu(x) \leq N$. On notera $\vec{\nu}$ le vecteur $[\nu(x_1), ..., \nu(x_{|\chi|})]$ lorsqu'il n'y aura pas d'ambiguité sur l'ensemble de variables.

**Définition 2** (Trace d'un Integer-Interpreted Automaton)**.** *Une trace dans un Integer-Interpreted Automaton d'ensemble de transitions* $\mathcal{T}$ *est une suite* $(k_0, \nu_0), (k_1, \nu_1), ..., (k_n, \nu_n)$ *telle que pour tout* $0 \leq i < n$, *il existe* $(k_i, g_i, a_i, k_{i+1}) \in \mathcal{T}$ *telle que* $g_i(\vec{\nu_i}) = true$ *et* $\vec{\nu}_{i+1} = a_i(\vec{\nu_i})$.

*On dit de plus qu'une trace est bornée si toutes ses valuations sont bornées par un même entier.*

**Terminaison.** On dit qu'un Integer-Interpreted Automaton *termine* s'il n'admet pas de trace infinie. Par exemple, on voit que l'automate représenté dans la Figure 2 est terminant. En effet, sinon on pourrait emprunter un cycle une infinité de fois. Or, comme $x$ est décroissant sur toutes les transitions, on ne peut prendre la transition qui fait strictement diminuer $x$ une infinité de fois, et donc on peut considérer qu'au bout d'un certain nombre de transitions, on ne l'emprunte plus. Tous les cycles restants passent alors par la transition où $y$ diminue strictement, et aucun par la transition où il peut augmenter. On ne peut prendre ces cycles une infinité de fois. Or ce sont les seuls cycles de l'automate. Toute trace est donc finie.

Voici quelques propriétés de cette abstraction qui la rendent propice à l'étude de la terminaison de programmes.

**Proposition 1** (Correction de l'Integer-Interpreted Automaton[1])**.** *Étant donné un programme* $P$ *et* $\mathcal{I}$ *son abstraction en Integer-Interpreted Automaton, si* $\mathcal{I}$ *termine, alors* $P$ *termine.*

L'outil principal utilisé pour étudier les Integer-Interpreted Automata est le concept de *fonction de rang*. Une fonction de rang $\rho$ de dimension $d$ d'un Integer-Interpreted Automaton $\mathcal{I} = (\mathcal{K}, \chi, k_0, \mathcal{T})$ est une fonction de $\mathcal{K} \times \mathbb{Z}^n \to \mathbb{N}^d$ décroissante sur toutes les transitions de $\mathcal{T}$ :

$$Pour\ tout\ (k, g, a, k') \in \mathcal{T},\ \vec{x}, \vec{x}' \in \mathbb{Z}^n,\ si\ g(\vec{x}) = true\ et\ \vec{x}' = a(\vec{x}),\ alors\ \rho(k', \vec{x}') < \rho(k, \vec{x}).$$

Il existe un algorithme, décrit dans [1], qui calcule une fonction de rang de dimension minimale d'un Integer-Interpreted Automaton, à l'aide de données issues d'un pré-traitement de cet Integer-Interpreted Automaton. Il est partiellement complet : s'il existe une fonction de rang et que le pré-traitement de l'Integer-Interpreted Automaton est assez précis, il en trouvera une.

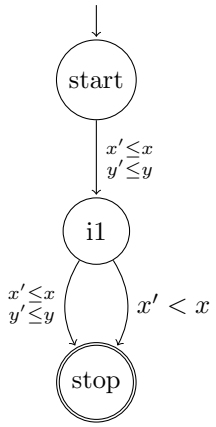
Si un Integer-Interpreted Automaton admet une fonction de rang, alors il est terminant.

**Complexité.** On peut également définir une notion de complexité pour un Integer-Interpreted Automaton terminant. Pour cela, on définit la fonction $f : \mathbb{N} \to \mathbb{N}$ qui à un entier associe la longueur de la plus longue trace dont la valuation initiale est bornée par cet entier. Dans [1] est exhibé un algorithme qui, étant donné un Integer-Interpreted Automaton, calcule un polynome $h \geq f$. La complexité de cet Integer-Interpreted Automaton est alors définie comme le degré de ce polynome. Cette complexité peut être sur-approximée grâce à une fonction de rang. De plus, la complexité temporelle d'un programme est sur-approximée par la complexité de son abstraction en Integer-Interpreted Automaton.

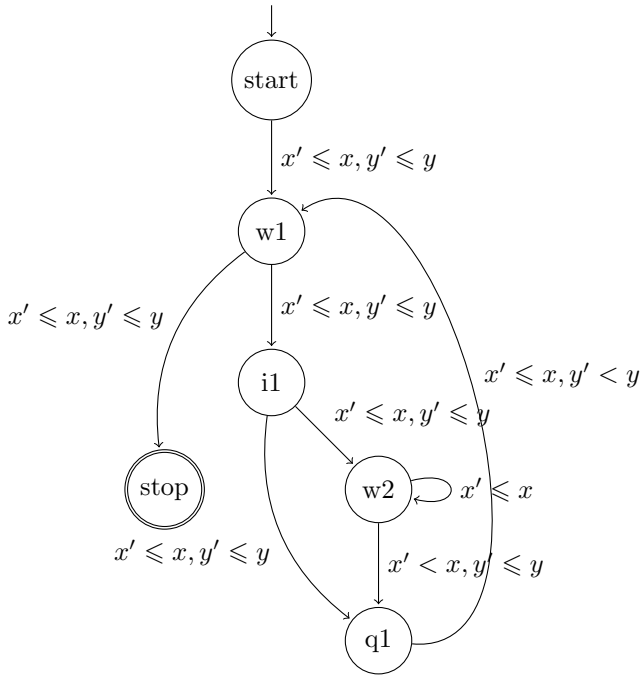
## 2.2 Size-Change Abstraction

Dans cette deuxième abstraction, on ne considère plus les gardes et on ne retient que la décroissance des variables. On peut la construire directement à partir du programme en utilisant une méthode similaire à celle qui permet d'obtenir l'Integer-Interpreted Automaton, ou alors l'abstraire de l'Integer-Interpreted Automaton.

La Figure 3 représente les abstractions en Size-Change Abstraction des Integer-Interpreted Automata des Figures 1 et 2.




(a) Size-Change Abstraction de l'Integer-Interpreted Automaton de la Figure 1




(b) Size-Change Abstraction de l'Integer-Interpreted Automaton de la Figure 2

FIGURE 3 – Exemples de Size-Change Abstractions

**Définition 3** (Size-Change Abstraction)**.** *Un automate Size-Change Abstraction est un uplet* $(\mathcal{K}, I, F, \chi, \mathcal{T})$ *où :*

- $\mathcal{K}$ *est un ensemble fini d'états,*
- $I$ *et* $F$ *sont respectivement les ensembles d'états initiaux et finaux,*
- $\chi$ *est un ensemble fini de variables,*
- $\mathcal{T}$ *est un ensemble fini de transitions tel que* $\mathcal{T} \subseteq (\mathcal{K}, \mathcal{A}, \mathcal{K})$*, où* $\mathcal{A}$ *est un ensemble d'inégalités de la forme* $x_1 > x_2'$ *ou* $x_1 \geqslant x_2'$ *où* $x_1, x_2 \in \chi$ *. Elles représentent les conditions de décroissance liant la valuation avant la transition à la valuation après. Les inégalités ne peuvent être que dans le sens décroissant : une ancienne variable plus grande qu'une nouvelle.*

On supposera que tous les états d'un Size-Change Abstraction sont accessibles et co-accessibles, ce qui sera toujours le cas pour une abstraction de programme.

On définit la sémantique de ces objets par valuations également. Une valuation est une fonction $\tau : \chi \to \mathbb{N}$. On dit que $\tau, \tau' \vDash x' < y$ (respectivement $\tau, \tau' \vDash x' \leqslant y$) lorsque $\tau'(x) < \tau(y)$ (respectivement $\tau'(x) \leqslant \tau(y)$). On étend cette définition aux transitions, on notant $\tau, \tau' \vDash (q, a, q')$ si pour tout $x' < y \in a$ (respectivement $x' \leq y \in a$), $\tau'(x) < \tau(y)$ (respectivement $\tau'(x) \leqslant \tau(y)$).

On dit de plus qu'une valuation $\tau$ est *bornée* par $N \in \mathbb{N}$ lorsque que pour tout $x \in \chi$, $\tau(x) \leq N$.

**Définition 4** (Trace)**.** *Une trace d'un Size-Change Abstraction $(\mathcal{K}, I, F, \chi, \mathcal{T})$ est une suite $T = \tau_0 \xrightarrow{t_0} \tau_1 \xrightarrow{t_1} ...$, de valuations et de transitions de $\mathcal{T}$ telles que :*

- *pour tout $i$, si $t_i = (q_i, a, q_i')$ et $t_{i+1} = (q_{i+1}, a', q_{i+1}')$, alors $q_i' = q_{i+1}$*
- *pour tout $i$, $\tau_i, \tau_{i+1} \vDash t_i$*

*Si la trace $T$ contient $k \in \mathbb{N}$ transitions, on dit de plus qu'elle est finie, et on définit sa longueur, notée $\ell(T)$, par $\ell(T) = k$. Sinon on dit qu'elle est infinie, et on pose par convention $\ell(T) = +\infty$. Elle est dite bornée par un entier $N \in \mathbb{N}$ lorsque toutes ses valuations sont bornées par ce même entier. Elle est dite acceptante si elle est finie, $q_0 \in I$, et $q_k' \in F$.*

Voici quelques résultats sur les Size-Change Abstractions.

**Terminaison.** Un automate Size-Change Abstraction est dit *terminant* lorsqu'il n'admet pas de trace infinie commençant par un état initial. Cette terminaison est décidable [9].

L'automate de la Figure 3b termine, par les mêmes arguments que ceux utilisés à la Section 2.1 pour son Integer-Interpreted Automaton associé qui est représenté en Figure 2.

**Complexité.** On définit la complexité d'un Size-Change Abstraction terminant de manière simlaire à celle d'un Integer-Interpreted Automaton terminant. On définit $f$ comme la fonction associant à un entier la longueur de la plus longue trace bornée par cet entier.

**Théorème 1** (Complexité d'un Size-Change Abstraction [6])**.** *Il existe un rationnel $\alpha$, appelé complexité du Size-Change Abstraction, tel que $f(N) =_{N \to +\infty} \Theta(N^\alpha)$. De plus, $\alpha$ est calculable par un algorithme prenant en entrée un Size-Change Abstraction.*

Cette complexité est une sur-approximation de la complexité temporelle du programme associé.

**Lien avec les Integer-Interpreted Automata.**

**Proposition 2.** *Étant donné un Integer-Interpreted Automaton $\mathcal{I}$ et son Size-Change Abstraction associé $\mathcal{S}$. Si $\mathcal{S}$ termine, alors $\mathcal{I}$ termine également.*

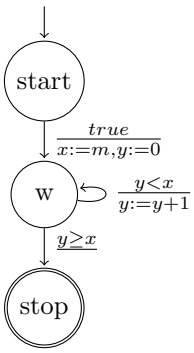
Cependant, la réciproque de cette propositions est fausse, comme nous le montre l'Exemple 1.

**Exemple 1.** *La Figure 4 présente un contre-exemple à la réciproque de la Proposition 2. En effet, l'Integer-Interpreted Automaton représenté n'admet pas de trace acceptante infinie, alors que le Size-Change Abstraction qui lui est associé en admet une. En effet, en notant $\nu$ la valuation telle que $\nu(x) = 1$, $\nu(y) = 1$, $\nu \xrightarrow{t} \nu \xrightarrow{t'} \nu \xrightarrow{t'} ...$ est une trace acceptante infinie du Size-Change Abstraction.*

# 3 Automates max-plus

On introduit ici les automates max-plus, qui vont permettre d'étudier la complexité des Size-Change Abstractions.

```
x:=m
y:=0
while (y<x) {        'w'
  y++;
};
```



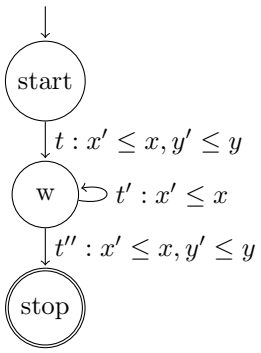




FIGURE 4 – Contre-exemple

## 3.1 Définition et exemples

Tout d'abord, on donne la définition d'une classe plus générale d'automates, les *automates pondérés*, introduits dans [12].

**Définition 5** (Automates pondérés)**.** *Un automate pondéré sur le semi-anneau* $\mathcal{S} = (S, \oplus, \otimes)$ *et l'alphabet* $\Sigma$ *est un uplet* $(\mathcal{Q}, I, F, \mathcal{T})$ *tel que :*

- $\mathcal{Q}$ *est un ensemble fini d'états,*
- $I \subseteq \mathcal{Q}$ *est l'ensemble des états initiaux,*
- $F \subseteq \mathcal{Q}$ *est l'ensemble des états finaux,*
- $\mathcal{T} \subseteq \mathcal{Q} \times \Sigma \times \mathcal{S} \times \mathcal{Q}$ *est un ensemble fini de transitions.*

**Sémantique.** Etant donné un automate pondéré $(\mathcal{Q}, I, F, \mathcal{T})$, un *chemin* est une suite

$$c = (q_1, a_1, p_1, q_1'), \ldots, (q_n, a_n, p_n, q_n')$$

de transitions de $\mathcal{T}$ telles que pour tout $1 \leq i < n$, $q_i' = q_{i+1}$. On le notera souvent :

$$q_1 \xrightarrow{a_1:p_1} q_2 \ldots q_n \xrightarrow{a_n:p_n} q_n'$$

Son *étiquette*, notée $etiq(c)$ est le mot $a_1 \ldots a_n$. Il est *acceptant* si $q_1 \in I$, $q_n' \in F$. Son *poids*, noté $poids(c)$, est le produit de toutes ses transitions :

$$poids(c) = \bigotimes_{1 \leq i \leq n} p_i$$

Finalement, le poids d'un mot $w \in \Sigma^*$ est la somme des poids des chemins acceptants dans l'automate étiquetés par ce mot :

$$poids(w) = \bigoplus_{\substack{c \text{ acceptant} \\ etiq(c)=w}} poids(c)$$

**Automates max-plus.** De manière plus spécifique, on va s'intéresser aux automates max-plus, c'est-à-dire aux automates pondérés sur le semi-anneau $(\mathbb{N} \cup \{-\infty\}, \max, +)$. Dans ce type d'automate, le poids d'un mot est donc le maximum du poids des chemins acceptants étiquetés par ce mot, et chaque chemin a pour poids la somme des poids de ses transitions. Si un mot n'est pas accepté, par convention son poids est $-\infty$, ce qui revient au fait que tous les chemins acceptants étiquetés par ce mot ont une transition de poids $-\infty$.

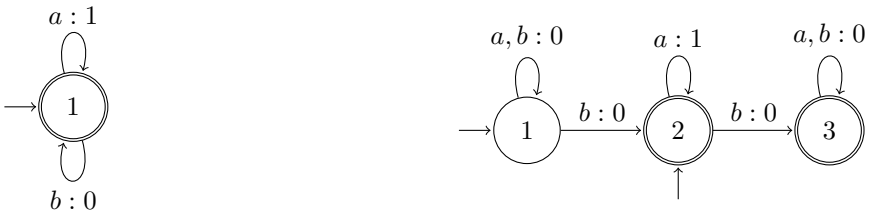
Contrairement aux automates finis non-déterministes, les automates max-plus donnent des informations quantitatives : on peut compter certaines propriétés des mots (nombre d'occurences d'une lettre, tailles de blocs...). On notera $p_{\mathcal{A}} : \Sigma^+ \to \mathbb{N} \cup \{-\infty\}$ la fonction qui à chaque mot associe son poids dans l'automate $\mathcal{A}$. Quand il n'y aura pas d'ambiguité sur $\mathcal{A}$, on notera simplement cette fonction $p$.

On remarque qu'avec cette définition, en rajoutant des transitions pesant $-\infty$ à un automate max-plus, on ne change pas la fonction calculée par cet automate max-plus.

**Exemple 2** (Quelques automates max-plus)**.** *La Figure 5 représente deux automates max-plus :*

*(a) Automate* $\mathcal{A}_1$ *:* $p_{\mathcal{A}_1}(w) = |w|_a$*.*

*(b) Automate* $\mathcal{A}_2$ *:* $p_{\mathcal{A}_2}(w) = max\{n|\ a^n\ facteur\ de\ w\}$*.*



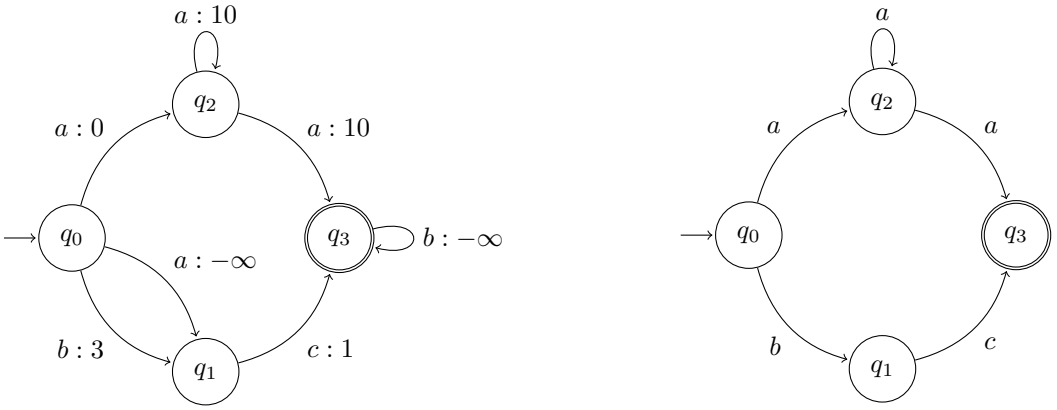

(a) $\mathcal{A}_1$ (b) $\mathcal{A}_2$

FIGURE 5 – Deux exemples d'automates max-plus

On pose comme convention pour la suite que, pour tout $n \in \mathbb{N}$, $n - \infty = -\infty + n = -\infty$.

**Définition 6** (Automate sous-jacent)**.** *Soit* $(\mathcal{Q}, I, F, \mathcal{T})$ *un automate max-plus sur l'alphabet* $\Sigma$*. Son automate sous-jacent est l'automate fini non-déterministe* $(\mathcal{Q}, I, F, \mathcal{T}')$ *où* $(q, a, q') \in \mathcal{T}'$ *si et seulement s'il existe* $s \in \mathbb{N}$ *tel que* $(q, a, s, q') \in \mathcal{T}$*. On notera* $\mathcal{L}(\mathcal{A})$ *le langage accepté par l'automate sous-jacent de* $\mathcal{A}$*.*

**Exemple 3.** *La Figure 6 représente un automate max-plus et son automate sous-jacent.*




(a) Automate max-plus $\mathcal{A}$ (b) Automate sous-jacent de $\mathcal{A}$

FIGURE 6 – Exemple d'automate sous-jacent d'un automate max-plus

Les preuves des deux lemmes suivants sont en Appendice, Section A.

**Lemme 1** (Langage rationnel associé)**.** *Soit* $\mathcal{A} = (\mathcal{Q}, I, F, \mathcal{T})$ *un automate max-plus sur l'alphabet* $\Sigma$*. Les langages* $\{w \in \Sigma^* \mid p(w) \geq 0\}$ *et* $\{w \in \Sigma^* \mid p(w) = -\infty\}$ *sont rationnels.*

**Lemme 2** (Opérations sur les automates max-plus)**.** *Soit* $\Sigma$ *un alphabet. Soient* $\mathcal{A}_1$ *et* $\mathcal{A}_2$ *deux automates max-plus sur* $\Sigma$*, et* $a \notin \Sigma$*. Les fonctions :*

— $f_1 : w \mapsto \max(p_{\mathcal{A}_1}(w), p_{\mathcal{A}_2}(w))$

— $f_2 : w \mapsto p_{\mathcal{A}_1}(w) + p_{\mathcal{A}_2}(w)$

— $f_3$ *telle que pour tout* $n \in \mathbb{N}$, $(w_i)_{1 \leq i \leq n+1} \in (\Sigma^*)^{n+1}$, $f_3(w_1 a ... a w_{n+1}) = \left(\sum_{i=1}^{n+1} p_{\mathcal{A}_1}(w_i)\right) + n$

— $f_4$ *telle que pour tout* $n \in \mathbb{N}$, $(w_i)_{1 \leq i \leq n+1} \in (\Sigma^*)^{n+1}$, $f_4(w_1 a ... a w_{n+1}) = \max_{1 \leq i \leq n+1} p_{\mathcal{A}_1}(w_i)$

*sont calculables par automate max-plus.*

**Complexité.**

**Théorème 2** (Complexité d'un automate max-plus [6])**.** *Soit $\mathcal{A}$ un automate max-plus sur l'alphabet $\Sigma$. Il existe $\alpha \in \mathbb{Q} \cup \{+\infty\}$, $\alpha \geq 1$, tel que :*

$$\sup_{\substack{w \in \Sigma^* \\ p_{\mathcal{A}}(w) \leq n}} (|w|) =_{n \to \infty} \Theta(n^\alpha)$$

*Pour $\alpha = +\infty$, cela signifie qu'il existe une suite de mots de longueurs strictement croissantes et de poids bornés.*

*De plus, il existe un algorithme qui prend en entrée un automate max-plus et calcule ce rationnel.*

La complexité d'un automate max-plus est définie par le $\alpha$ donné dans le Théorème 2. On dit de plus que l'automate max-plus termine si cette complexité est finie.

## 3.2 Lien avec les Size-Change Abstractions

On généralise ici une transformation d'un Size-Change Abstraction en automate max-plus donnée dans [6] pour faire le lien entre les complexités des deux modèles.

Considérons un Size-Change Abstraction $\mathcal{S} = (\mathcal{K}, I, F, \mathcal{X}, \mathcal{T})$.

On construit un automate max-plus $\mathcal{A}_1$, sur l'alphabet $\mathcal{T}$, $(\mathcal{Q} \times (\mathcal{X} \cup \{start; stop\}), I \times \{start\}, F \times \{stop\}, \mathcal{T}')$ où $\mathcal{T}'$ est défini de la manière suivante :

- si $t = (q_1, A, q_2) \in \mathcal{T}$ et $x_i' < x_j \in A$, alors $((q_1, x), t, 1, (q_2, y)) \in \mathcal{T}'$ pour $x \in \{start, x_j\}$ et $y \in \{stop, x_i\}$,
- si $t = (q_1, A, q_2) \in \mathcal{T}$ et $x_i' \leq x_j \in A$, alors $((q_1, x), t, 0, (q_2, y)) \in \mathcal{T}'$ pour $x \in \{start, x_j\}$ et $y \in \{stop, x_i\}$,
- si $t = (q, a, q') \in \mathcal{T}$, alors $((q_1, start), t, 0, (q_2, start)) \in \mathcal{T}'$ et $((q_1, stop), t, 0, (q_2, stop)) \in \mathcal{T}'$.

On construit un automate max-plus $\mathcal{A}_2$ de la manière suivante :

- On construit un automate fini non-déterministe reconnaissant $\{w \mid p_{\mathcal{A}_1}(w) = -\infty\}$, ce qui est possible par le Lemme 1.
- On rajoute un poids 1 à chaque transition de cet automate. L'automate $\mathcal{A}_2$ est l'automate max-plus obtenu.

Cet automate max-plus calcule donc la taille des mots qui ont un poids $-\infty$ par l'automate max-plus $\mathcal{A}_1$.

L'automate max-plus associé au Size-Change Abstraction est alors celui qui calcule la fonction de $\mathcal{T}^*$ dans $\mathbb{N}$ :

$$w \mapsto \max(p_{\mathcal{A}_1}(w), p_{\mathcal{A}_2}(w))$$

Cet automate max-plus existe par le Lemme 2. Il est l'union disjointe de ces deux automates.

Dans la suite du rapport, on le notera $\mathcal{A}_\mathcal{S} = \mathcal{A}_1 \sqcup \mathcal{A}_2$.

**Exemple 4.** *La Figure 7 représente l'abstraction en automate max-plus de la Figure 3a.*

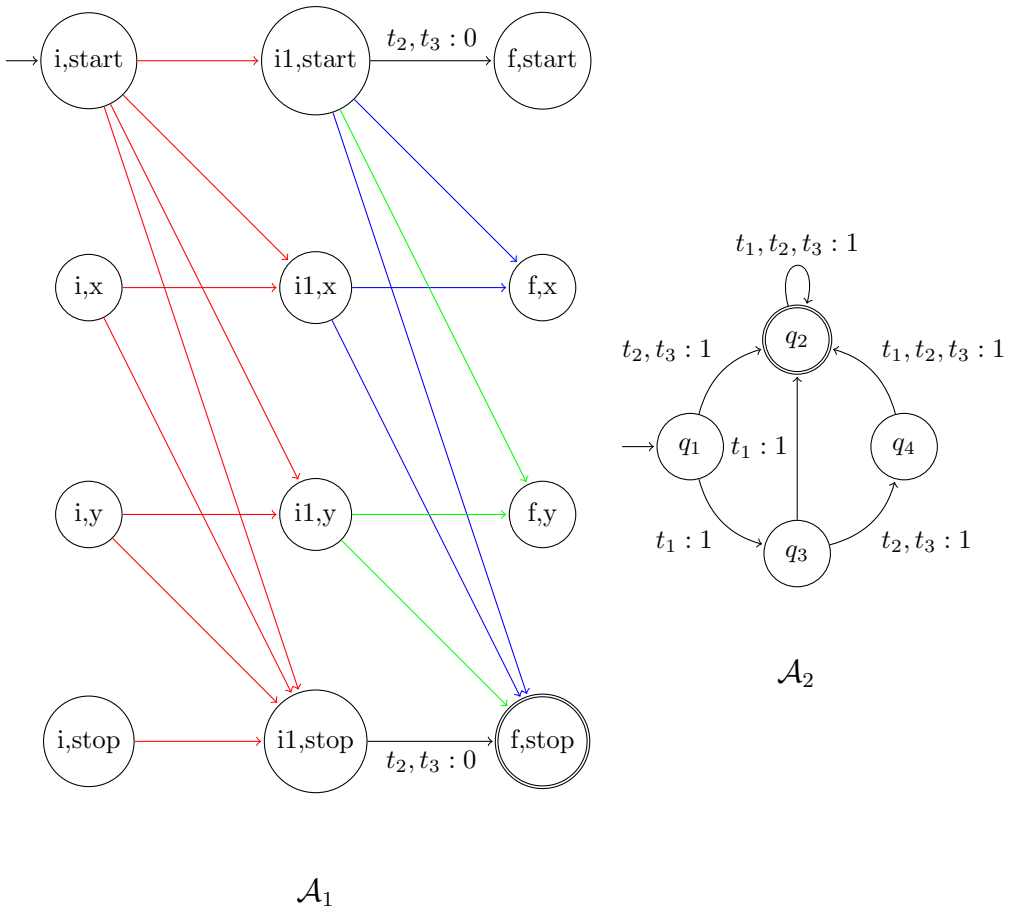
Tout d'abord, le résultat suivant constitue un lien fondamental entre ces deux modèles.

**Proposition 3** ([6])**.** *Étant donné un Size-Change Abstraction $\mathcal{S}$ terminant, alors $\mathcal{A}_\mathcal{S}$ et $\mathcal{S}$ ont la même complexité.*

Un des résultats principaux de ce mémoire, donné dans la Proposition suivante, est la preuve que l'hypothèse de terminaison est superflue.

**Proposition 4.** *Étant donné un Size-Change Abstraction $\mathcal{S}$, $\mathcal{A}_\mathcal{S}$ est terminant si et seulement si $\mathcal{S}$ est terminant.*

La preuve de ce théorème est en Appendice, dans la Section B. Elle se compose des deux implications réciproques :




représente une transition étiquetée par $t_1 : 0$

représente deux transitions étiquetées par $t_2 : 0$ et $t_3 : 1$

représente une transition étiquetée par $t_2 : 0$

FIGURE 7 – Automate max-plus obtenu grâce au Size-Change Abstraction de la Figure 3a

— Si $\mathcal{S}$ ne termine pas, alors $\mathcal{A}_\mathcal{S}$ ne termine pas. Pour cela, il faut prouver au préalable que si $\mathcal{S}$ admet une trace infinie, il admet une trace infinie bornée. Donc on peut trouver un cycle tel que la valuation de sortie est la même que la valuation d'entrée. Dans l'automate max-plus, on peut ainsi construire des mots arbitrairement longs de poids bornés en itérant ce cycle.

— Si $\mathcal{A}_\mathcal{S}$ ne termine pas, alors $\mathcal{S}$ ne termine pas. Il existe une borne $N$ et des mots arbitrairement longs de poids bornés par $N$. Donc il existe dans $\mathcal{S}$ une trace arbitrairement longue bornée par $N$. En prenant une trace suffisamment longue, on peut trouver un cycle dont la valuation de sortie est la même que la valuation d'entrée. On construit alors une trace infinie bouclant sur ce cycle.

**Corollaire 1.** *Étant donné un Integer-Interpreted Automaton $\mathcal{I}$ et son automate max-plus associé $\mathcal{A}$, si $\mathcal{A}$ termine alors $\mathcal{I}$ termine également.*

# 4 Étude d'une famille spécifique d'automates max-plus à complexité rationnelle

L'Integer-Interpreted Automaton et le Size-Change Abstraction (ou son automate max-plus associé) donnent a priori deux différentes sur-approximations de la complexité d'un programme. La complexité d'un Integer-Interpreted Automaton est entière alors que celle d'un Size-Change Abstraction est rationnelle. Il pourrait donc exister une famille de programmes tels que leurs automates max-plus et Integer-Interpreted Automata associés soient terminants, et que la complexité de leurs automates max-plus soit plus précise.

Pour explorer cette possibilité, on se concentre sur les automates max-plus de complexité rationnelle. Existe-t-il un automate max-plus de complexité $\alpha$ pour chaque $\alpha$ rationnel ? Si oui, existe-t-il un programme qui se traduit dans cet automate max-plus ? A-t-il une complexité rationnelle également ? Quel est son Integer-Interpreted Automaton associé ? Dans cette partie, on présente la démarche qui a abouti à la construction inductive d'une famille d'automates $\mathcal{A}_{\ell,k}$ de complexité $\frac{k}{\ell}$ pour tout $(k,\ell) \in \mathbb{N}^* \times \mathbb{N}$ tel que $\ell < k$. On considèrera dans toute cette section $\Sigma_k = \{a_i \mid 1 \leq i \leq k\}$, un alphabet à $k$ lettres.

## 4.1 Quelques exemples

On donne deux exemples d'automate max-plus de complexité 2 et $\frac{3}{2}$.

**Automate max-plus de complexité 2.** Un automate max-plus calculant la longueur des mots est de complexité 1. Essayons de trouver un automate max-plus $\mathcal{A}_{1,2}$ de complexité 2. On veut que pour un entier $n$ donné, le mot le plus long ayant un poids inférieur à $n$ soit de longueur équivalente à $n^2$. On peut compter jusqu'à $n^2$ avec deux compteurs initialisés à 0 en répétant les opérations :

— le premier augmente jusqu'à atteindre $n$, il redescend à 0,

— le deuxième augmente de 1,

jusqu'à ce que les deux compteurs soient à $n$. L'exécution a $n(n+1)$ étapes, et les compteurs ont au plus été égaux à $n$. On peut imaginer une machine qui fonctionnerait de manière similaire :

— Elle possède deux compteurs.

— L'exécution s'arrête lorsque l'un des deux dépasse $n$.

— À chaque itération, on a le choix entre

  — faire augmenter le premier de 1,

  — le redescendre à 0 et augmenter l'autre de 1

Intuitivement, on voit que la plus longue exécution possible est de longueur $n(n+1)$. Il y a un automate qui mime exactement ce comportement, définit sur $\{a,b\}^*$, illustré par la Figure 8.

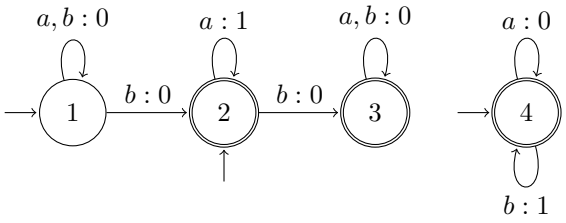



FIGURE 8 – $\mathcal{A}_{1,2}$

En effet, cet automate calcule le maximum entre le plus grand bloc de $a$ et le nombre de $b$. Ainsi, le premier compteur est représenté par le calcul du plus grand bloc de $a$, et l'autre par le calcul du nombre de $b$ :

— si on rencontre un $a$, le premier compteur augmente de 1,

— si on rencontre un $b$, le premier compteur se remet à 0, et le second est incrémenté de 1.

Cet automate a une complexité de 2. Les mots de la forme $(a^n b)^n a^n$ sont des témoins de cette complexité. On appellera informellement par la suite ce type de mots les mots bien répartis.

En rajoutant des lettres et en comptant les blocs de manière similaire, on peut obtenir une famille d'automates max-plus de complexité $k \in \mathbb{N}$.

**Automate max-plus de complexité $\frac{3}{2}$.** On a vu dans le paragraphe précédent que les mots bien-répartis sont pertinents dans le cadre de notre étude. On essaye donc de chercher un automate :

- défini sur l'alphabet $\Sigma = \{a, b, c\}$
- dans lequel les mots $((a^n b)^n c)^n (a^n b)^n a^n$, de longueur équivalente à $n^3$, aient un poids équivalent à $n^2$

On remarque que des éléments de ce mot équivalents à $n^2$ sont :

- le nombre de $b$, additionné au nombre de $c$,
- les blocs de $a$ et $b$ entre chaque $c$,
- la somme sur les couples de $c$ consécutifs de la longueur d'un bloc de $a$ entre ces deux $c$, additionné au nombre de $c$.

On peut construire un automate qui calcule exactement le maximum de ces valeurs. L'automate est représenté en Figure 9. Soit $w = w_1 c ... c w_{p+1}$ un mot sur $\Sigma$ tel que pour tout $i$, $w_i$ ne contient pas de $c$ ; l'automate calcule :

$$\max\left(\max_i(|w_i|), \sum_i (p_{\mathcal{A}_{1,2}}(w_i)) + p\right)$$

L'automate $\mathcal{A}_{3,2}$ a une complexité de $\frac{3}{2}$.

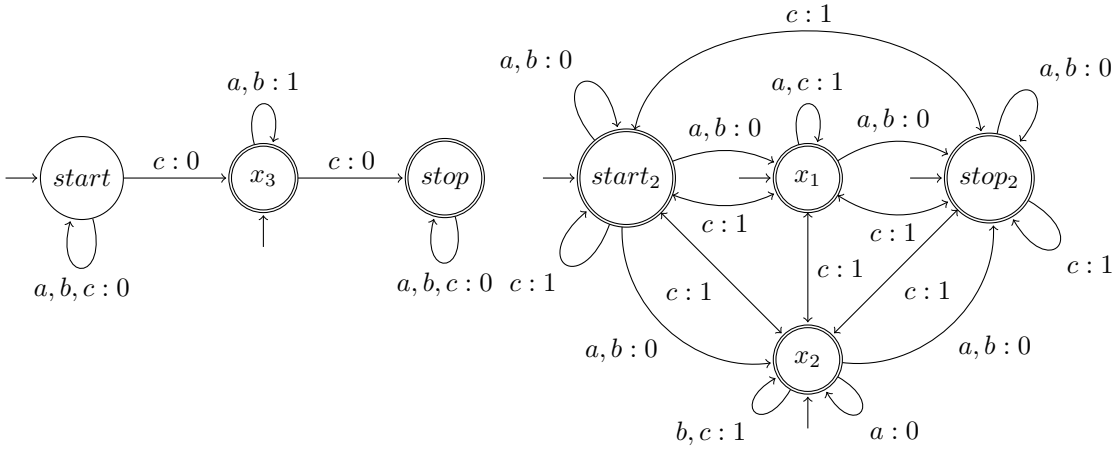



FIGURE 9 – $\mathcal{A}_{3,2}$

Dans la suite, on définit une famille d'automates qui a une complexité rationnelle en utilisant des idées similaires.

## 4.2 Familles de complexité 0 et $+\infty$

Comme illustré dans les exemples précédents, on va procéder par induction. On donne ici les familles de bases de l'induction.

**Famille $(\mathcal{A}_{0,k})_{k\in\mathbb{N}^*}$.** Soit $k \in \mathbb{N}^*$, on définit $\mathcal{A}_{0,k} = (\{q_k\}, \{q_k\}, \{q_k\}, \mathcal{T})$ sur $\Sigma_k$ où :

$$\mathcal{T} = \{(q_k, a, 0, q_k) \mid a \in \Sigma_k\}$$

La Figure 10a représente un élément de la famille $(\mathcal{A}_{0,k})_{k\in\mathbb{N}^*}$.

Cette famille a pour complexité $+\infty$ : tous les mots ont un poids nul, toute famille de mots de longueur strictement croissante est témoin de cette complexité.

**Famille $(\mathcal{A}_{k,k})_{k\geq 1}$.** Soit $k \in \mathbb{N}^*$, on définit $\mathcal{A}_{k,k} = (\{q_k\}, \{q_k\}, \{q_k\}, \mathcal{T})$ sur $\Sigma_k$ où :

$$\mathcal{T} = \{(q_k, a, 1, q_k) \mid a \in \Sigma_k\}$$

La Figure 10b représente un élément de la famille $(\mathcal{A}_{k,k})$.

Cette famille a pour complexité 1 : tous les mots ont pour poids leur longueur.


(a) $\mathcal{A}_{0,k}$ (b) $\mathcal{A}_{k,k}$

FIGURE 10 – Familles de complexité 0 et $+\infty$

## 4.3 Atteindre tous les rationnels

On définit par induction les automates $(\mathcal{A}_{\ell,k})_{\ell,k\in(\mathbb{N}\times\mathbb{N}^*),k\geq\ell}$ sur l'alphabet $\Sigma_k$.

— Si $\ell = 0$, $\mathcal{A}_{\ell,k}$ est l'automate défini dans la Section 4.2.

— Si $\ell = k$, $\mathcal{A}_{\ell,k}$ est l'automate défini dans la Section 4.2.

— Si $k > \ell > 0$, on suppose construits $\mathcal{A}_{\ell-1,k-1}$ et $\mathcal{A}_{\ell,k-1}$ de fonctions de poids respectives $p_{\ell-1,k-1}$ et $p_{\ell,k-1}$. On définit une fonction $p_{\ell,k}$ de la manière suivante : étant donné un mot $w = w_1 a_{k+1} ... a_{k+1} w_{p+1}$ où $(w_i)_{1\leq i\leq p+1} \in (\Sigma_k^*)^{p+1}$,

$$p_{\ell,k}(w) = \max\left(\max_{1\leq i\leq p+1} p_{\ell,k-1}(w_i), \sum_{1\leq i\leq p+1} p_{\ell-1,k-1}(w_i) + p\right)$$

**Théorème 3.** *Pour tout $(\ell, k) \in \mathbb{N}\times\mathbb{N}^*$, $k \geq \ell$, la fonction $p_{\ell,k}$ est calculable par un automate max-plus $\mathcal{A}_{\ell,k}$ de complexité $\frac{k}{\ell}$.*

La preuve de ce théorème est en Appendice, à la Section C. Elle suit la trame suivante :

— Tout d'abord, on prouve que $p_{\ell,k}$ est calculable par un automate max-plus en utilisant le Lemme 2.

— On montre que la complexité de l'automate $\mathcal{A}_{\ell,k}$ est plus grande que $\frac{k}{\ell}$ en exhibant une famille de mots particuliers. Cette famille $(w_{n,k})_n$ est celle des mots bien répartis sur l'alphabet $\Sigma_k$, qui ont longueur équivalente à $n^k$ et poids équivalent à $n^\ell$ dans $\mathcal{A}_{\ell,k}$.

— On montre par induction que la complexité de l'automate $\mathcal{A}_{\ell,k}$ est plus petite que $\frac{k}{\ell}$. On commence par traiter par récurrence sur $k$ le cas $\ell = 1$ qui diffère du cas général, en utilisant les propriétés de la famille de base $(\mathcal{A}_{0,k})_{k\in\mathbb{N}^*}$. On traite ensuite le cas général en prenant comme familles de base les familles $(\mathcal{A}_{1,k})_{k\in\mathbb{N}^*}$ et $(\mathcal{A}_{k,k})_{k\in\mathbb{N}^*}$. Les dépendances de cette induction sont illustrées par la Figure 11. On voudrait montrer que la famille $(w_{n,k})_n$ est optimale : pour chacun de ces mots, il n'existe pas de mot plus long de poids inférieur. Cependant, la preuve exacte nécessiterait des arguments combinatoires très complexes. On prouve l'optimalité de cette famille de manière asymptotique : il existe une constante $C$ telle que pour tout $n$, un mot de poids inférieur au poids de $w_{n,k}$ est de longueur inférieure à $C|w_{n,k}|$.

## 4.4 Retour à l'Integer-Interpreted Automaton

À partir des ces automates max-plus particuliers, on peut construire des Size-Change Abstractions et des programmes de même complexité asymptotique, donc de toute complexité rationnelle. La complexité donnée par l'Integer-Interpreted Automaton sera donc plus grande que celle donnée par l'automate max-plus, puisqu'il donne un entier sur-approximant la complexité du programme, alors que l'automate max-plus donne un rationnel.

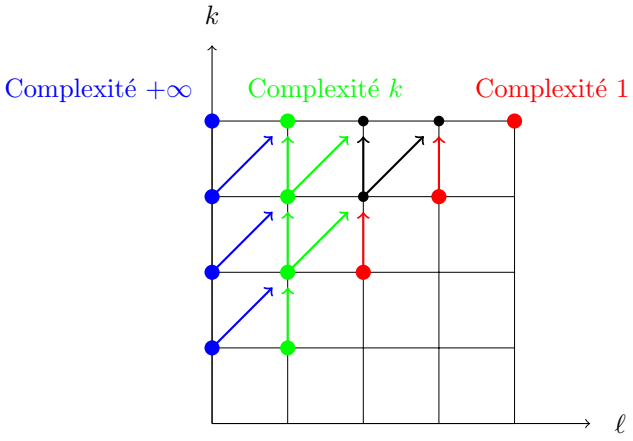



FIGURE 11 – Dépendances de l'induction

On a codé un programme prenant en entrée $k$ et qui retourne le code d'un programme de complexité $k$ dont l'automate max-plus associé est équivalent à $\mathcal{A}_{1,k}$. On a testé les programmes jusqu'à $k = 4$ et l'Integer-Interpreted Automaton associé a complexité $k$. À partir de $k = 5$, le programme généré est trop gros pour être traité par Integer-Interpreted Automaton.

Enfin, on a un programme (en Appendice) de complexité $\frac{3}{2}$ correspondant à l'automate $\mathcal{A}_{3,2}$, pour lequel l'Integer-Interpreted Automaton associé ne peut donner qu'une complexité supérieure à 2.

Les démarches, les programmes et le code développés dans ce cadre sont exposés en Appendice, dans la section D.

# 5 Composition des deux méthodes

Les deux méthodes étudiées ont ainsi des propriétés très différentes. Il parait difficile de les mêler car les procédés sont extrêmement différents. Cependant, il semble possible de les composer.

## 5.1 Intégrer un Integer-Interpreted Automaton à un automate max-plus

On analyse un programme. On souhaite traiter certaines boucles de ce programme avec la méthode des Integer-Interpreted Automata , pour ensuite réinjecter les informations obtenues dans l'automate max-plus. Ainsi, on pourra élargir le champ des programmes traités par cette abstraction, et dans certains cas grandement améliorer les résultats des deux méthodes.

L'information donnée par l'Integer-Interpreted Automaton sur les boucles considérées qui nous est utile est la complexité. Cette sous-partie présente la manière d'intégrer cette complexité à l'automate max-plus correspondant.

**Intégrer les informations aux transitions.** Lorsque l'on sait déjà qu'une boucle du programme termine pour toutes valeurs des variables en début de cette boucle, on peut remplacer cette boucle par une transition. Cependant, il faut prendre en compte la complexité de cette transition, qui ne représente plus une étape élémentaire mais un ensemble d'étapes élémentaires. Il faut de plus récupérer des informations sur l'évolution des variables pendant cette boucle.

Pour cela, on va définir un nouveau modèle d'automate max-plus , l'*automate max-plus pesant* : on rajoute un deuxième poids à chaque transition.

**Définition 7** (Automate max-plus pesant)**.** *Un automate max-plus pesant sur l'alphabet $\Sigma$ est un uplet $(\mathcal{Q}, I, F, \mathcal{T})$ tel que :*

- *$\mathcal{Q}$ est un ensemble fini d'états,*
- *$I \subseteq \mathcal{Q}$ est l'ensemble des états initiaux,*
- *$F \subseteq \mathcal{Q}$ est l'ensemble des états finaux,*

— $\mathcal{T} \subseteq \mathcal{Q} \times \Sigma \times \mathbb{N} \times \mathbb{Q} \times \mathcal{Q}$ *est un ensemble fini de transitions.*

Dans le cadre de l'abstraction de programme, ce deuxième poids représente la complexité de la transition, c'est-à-dire le nombre d'étapes élémentaires ont été condensées en une transition. Cependant, ce n'est pas un poids explicite, mais un poids correspondant à l'exposant maximal du polynôme de la complexité trouvée par Integer-Interpreted Automaton sur la boucle étudiée. Lorsque l'on utilise la méthode d'abstraction du programme en automate max-plus, on met des poids nuls partout initialement.

# 6 Conclusion

# Références


[1] Christophe Alias, Alain Darte, Paul Feautrier, and Laure Gonnord. Multi-dimensional Rankings, Program Termination, and Complexity Bounds of Flowchart Programs. In *Static Analysis Symposium*, Perpignan France, 2010.

[2] Hugh Anderson and Siau-Cheng Khoo. Affine-based size-change termination. In Atsushi Ohori, editor, *Programming Languages and Systems, First Asian Symposium, APLAS 2003, Beijing, China, November 27-29, 2003, Proceedings*, volume 2895 of *Lecture Notes in Computer Science*, pages 122–140. Springer, 2003.

[3] Amir M. Ben-Amram. Size-change termination, monotonicity constraints and ranking functions. *Logical Methods in Computer Science*, 6(3), 2010.

[4] Aaron R. Bradley, Zohar Manna, and Henny B. Sipma. The polyranking principle. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings*, volume 3580 of *Lecture Notes in Computer Science*, pages 1349–1361. Springer, 2005.

[5] Aziem Chawdhary, Byron Cook, Sumit Gulwani, Mooly Sagiv, and Hongseok Yang. Ranking abstractions. In Sophia Drossopoulou, editor, *Programming Languages and Systems, 17th European Symposium on Programming, ESOP 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4960 of *Lecture Notes in Computer Science*, pages 148–162. Springer, 2008.

[6] Thomas Colcombet, Laure Daviaud, and Florian Zuleger. Size-change abstraction and max-plus automata. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part I*, volume 8634 of *Lecture Notes in Computer Science*, pages 208–219. Springer, 2014.

[7] Patrick Cousot. Proving program invariance and termination by parametric abstraction, lagrangian relaxation and semidefinite programming. In Radhia Cousot, editor, *Verification, Model Checking, and Abstract Interpretation, 6th International Conference, VMCAI 2005, Paris, France, January 17-19, 2005, Proceedings*, volume 3385 of *Lecture Notes in Computer Science*, pages 1–24. Springer, 2005.

[8] Robert Floyd. Assigning meanings to programs. In *Mathematical Aspects of Computer Science. Proceedings of Symposium on Applied Mathematics 19. American Mathematical Society 1967. Proceedings*, pages 19—-32, 1967.

[9] Chin Soon Lee, Neil D. Jones, and Amir M. Ben-Amram. The size-change principle for program termination. In Chris Hankin and Dave Schmidt, editors, *Conference Record of POPL 2001 : The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, London, UK, January 17-19, 2001*, pages 81–92. ACM, 2001.

[10] Panagiotis Manolios and Daron Vroon. Termination analysis with calling context graphs. In Thomas Ball and Robert B. Jones, editors, *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4144 of *Lecture Notes in Computer Science*, pages 401–414. Springer, 2006.

[11] Andreas Podelski and Andrey Rybalchenko. A complete method for the synthesis of linear ranking functions. In Bernhard Steffen and Giorgio Levi, editors, *Verification, Model Checking, and Abstract Interpretation, 5th International Conference, VMCAI 2004, Venice, January 11-13, 2004, Proceedings*, volume 2937 of *Lecture Notes in Computer Science*, pages 239–251. Springer, 2004.

[12] Marcel Paul Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2-3) :245–270, 1961.

# A Propriétés des automates max-plus

*Preuve du Lemme 1.* Soit $w \in \Sigma^*$.

— Si $p(w) \geq 0$, alors il existe un chemin acceptant étiqueté par $w$ de poids positif. Notons $t_1 = (q_1, a_1, p_1, q_2), ...t_n = (q_n, a_n, p_n, q_{n+1})$ les transitions qu'il emprunte. Tous les poids des transitions de ce chemin sont positives. En effet, supposons qu'il existe $i$ tel que $p_i$ soit $-\infty$. Le poids du chemin serait $-\infty$. Si toutes les transitions ont un poids positif, alors, pour tout $i \leq n$ : $(q_i, a_i, q_{i+1})$ est dans l'ensemble des transitions de l'automate sous-jacent de $\mathcal{A}$. De plus, $q_1$ est dans l'ensemble des états initiaux de cet automate, et $q_{n+1}$ dans son ensemble des états finaux. Donc il existe un chemin acceptant étiqueté par $w$ dans l'automate sous-jacent de $\mathcal{A}$.

— Si $w$ est reconnu par l'automate sous-jacent de $\mathcal{A}$, il existe un chemin acceptant étiqueté par $w$ dans cet automate, notons le $t_1 = (q_1, a_1, q_2), ...t_n = (q_n, a_n, q_{n+1})$. Pour tout $1 \leq i \leq n$, il existe $p_i \in \mathbb{N}$ tel que $(q_i, a_i, p_i, q_{i+1}) \in \mathcal{T}$, par définition de l'automate sous-jacent. De plus, $q_1 \in I$ et $q_{n+1} \in F$. Il existe donc un chemin acceptant étiqueté par $w$ dans $\mathcal{A}$. De plus, tous les poids des transitions de ce chemin sont positives. Ainsi, leur somme est positive. Puisque par définition du poids d'un mot dans $\mathcal{A}$, $p(w)$ est plus grand que le poids de ce chemin, $p(w) \geq 0$.

Donc $\{w \in \Sigma^* \mid p(w) \geq 0\}$ est rationnel. Or, $\{w \in \Sigma^* \mid p(w) = -\infty\}$ est le complémentaire de ce langage. Par clôture par complémentaire des langages rationnels, il est également rationnel. □

*Preuve du Lemme 2.* — On pose $\mathcal{A}_3$ l'union disjointe de $\mathcal{A}_1$ et $\mathcal{A}_2$, c'est-à-dire que, en notant $\mathcal{A}_1 = (Q_1, I_1, F_1, T_1)$ et $\mathcal{A}_2 = (Q_2, I_2, F_2, T_2)$, alors

$$\mathcal{A}_3 = (Q_1 \sqcup Q_2, I_1 \sqcup I_2, F_1 \sqcup F_2, T_1 \sqcup T_2)$$

En effet, considérons un mot $w$ sur l'alphabet $\Sigma$.

— Soit $c$ le chemin acceptant de $w$ dans $\mathcal{A}_1$ de poids $p_{\mathcal{A}_1}(w)$, alors c'est également un chemin de $w$ dans $\mathcal{A}_3$, car tous les états et transitions de $\mathcal{A}_1$ sont dans $\mathcal{A}_3$. Donc $p_{\mathcal{A}_1}(w) = poids_{\mathcal{A}_1}(c) = poids_{\mathcal{A}_3}(c) \leq p_{\mathcal{A}_3}(w)$

— Symétriquement, $p_{\mathcal{A}_2}(w) \leq p_{\mathcal{A}_3}(w)$

— Inversement, puisque l'union est disjointe, le chemin acceptant de $w$ dans $\mathcal{A}_3$ de poids maximal est également un chemin de $w$ dans $\mathcal{A}_1$ ou dans $\mathcal{A}_2$. Donc $p_{\mathcal{A}_3}(w) \leq \max(p_{\mathcal{A}_1}(w), p_{\mathcal{A}_2}(w))$

On en déduit que pour tout mot $w$ :

$$p_{\mathcal{A}_3}(w) = \max(p_{\mathcal{A}_1}(w), p_{\mathcal{A}_2}(w))$$

— On pose $\mathcal{A}_3 = \mathcal{A}_1 \times \mathcal{A}_2$, c'est-à-dire le produit des automates, comme on le ferait pour des automates classiques. On pose comme poids pour chaque transition la somme des poids des transitions dont elle est le produit. Formellement, si $\mathcal{A}_1 = (Q_1, I_1, F_1, T_1)$ et $\mathcal{A}_2 = (Q_2, I_2, F_2, T_2)$, alors on pose :

$$\mathcal{A}_3 = (Q_1 \times Q_2, I_1 \times I_2, F_1 \times F_1, T_3)$$

où $T_3 = \{\ ((q_1, q_2), a, p_1 + p_2, (q_1', q_2')),\ a \in \Sigma_1 \cup \Sigma_2,\ (q_1, a, p_1, q_1') \in T_1,\ (q_2, a, p_2, q_2') \in T_2\}$.
On notera de plus ces nouvelles transitions comme les produit d'anciennes, soit en reprenant les notations précédentes : $((q_1, q_2), a, p_1 + p_2, (q_1', q_2')) = (q_1, a, p_1, q_1') \times (q_2, a, p_2, q_2')$.
Ainsi, si $w$ est un mot sur $\Sigma_1 \cup \Sigma_2$,

— Soient $c_1$ et $c_2$ deux chemins acceptants maximaux de $w$ dans $\mathcal{A}_1$ et $\mathcal{A}_2$,

$$\begin{cases} c_1 = q_1 \xrightarrow{t_1} ... \xrightarrow{t_p} q_{p+1} \\ c_2 = q_1' \xrightarrow{t_1'} ... \xrightarrow{t_p'} q_{p+1}' \end{cases}$$

Alors $c = (q_1, q_1') \xrightarrow{t_1 \times t_1'} ... \xrightarrow{t_p \times t_p'} (q_{p+1}, q_{p+1}')$ est un chemin acceptant de $w$ dans $\mathcal{A}_3$, et en notant pour tout $1 \leq i \leq p$ $p_i$ le poids de la transition $t_i$ et $p_i'$ celui de $t_i'$ :

$$poids_{\mathcal{A}_3}(c) = \sum_{1 \leq i \leq p} (p_i + p_i') = \sum_{1 \leq i \leq p} p_i + \sum_{1 \leq i \leq p} p_i' = poids_{\mathcal{A}_1}(c_1) + poids_{\mathcal{A}_2}(c_2) = p_{\mathcal{A}_1}(w) + p_{\mathcal{A}_2}(w)$$

Et donc :

$$p_{\mathcal{A}_3}(w) \geq p_{\mathcal{A}_1}(w) + p_{\mathcal{A}_2}(w)$$

— Réciproquement, si $c = (q_1, q'_1) \xrightarrow{t_1 \times t'_1} ... \xrightarrow{t_p \times t'_p} (q_{p+1}, q'_{p+1})$ est un chemin acceptant maximal de $w$ dans $\mathcal{A}_3$, alors $c_1 = q_1 \xrightarrow{t_1} ... \xrightarrow{t_p} q_{p+1}$ et $c_2 = q'_1 \xrightarrow{t'_1} ... \xrightarrow{t'_p} q'_{p+1}$ sont des chemins acceptants de $w$ dans $\mathcal{A}_1$ et $\mathcal{A}_2$, et de même

$$p_{\mathcal{A}_3}(w) = poids_{\mathcal{A}_3}(c) = \sum_{1 \leq i \leq p} (p_i + p'_i) = \sum_{1 \leq i \leq p} p_i + \sum_{1 \leq i \leq p} p'_i = poids_{\mathcal{A}_1}(c_1) + poids_{\mathcal{A}_2}(c_2)$$

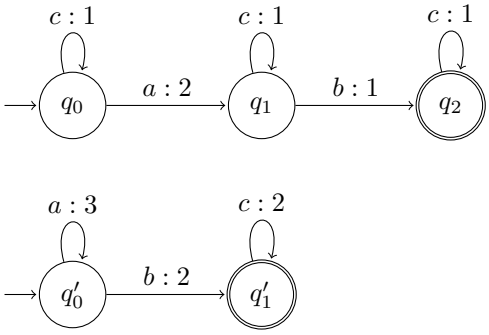
Et donc :

$$p_{\mathcal{A}_3}(w) \leq p_{\mathcal{A}_1}(w) + p_{\mathcal{A}_2}(w)$$

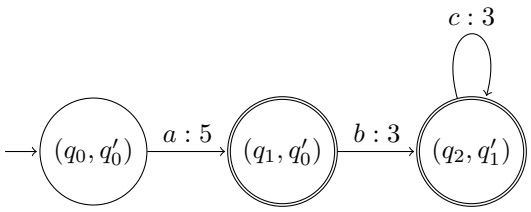
Donc, pour tout mot $w$ de $\Sigma_1 \cup \Sigma_2$,

$$p_{\mathcal{A}_3}(w) = p_{\mathcal{A}_1}(w) + p_{\mathcal{A}_2}(w)$$

**Exemple 5.** *La Figure 12 représente deux automates et leur somme.*




(a) Automates max-plus $\mathcal{A}_1$ et $\mathcal{A}_2$




(b) Somme de $\mathcal{A}_1$ et $\mathcal{A}_2$

FIGURE 12 – Exemple de somme de deux automates max-plus

— Soit $\mathcal{A} = (Q, I, F, T)$ un automate max-plus sur l'alphabet $\Sigma$. On construit $\mathcal{A}' = (Q, I, F, T')$ sur l'alphabet $\Sigma \cup \{a\}$ où :

$$T' = T \cup \{ \ (f, a, 1, i), \ i \in I, \ f \in F\}$$

Soit $w$ un mot sur $\Sigma \cup \{a\}$. On peut l'écrire de manière unique : $w = w_1 a ... a w_{p+1}$ où $(w_i)_{1 \leq i \leq p+1}$ famille de mots sur $\Sigma$. Soit $c$ un chemin acceptant maximal de $w$ dans $\mathcal{A}'$. On peut l'écrire :

$$c = q_{1,1} \xrightarrow{t_{1,1}} q_{1,2} ... \xrightarrow{t_{1,|w_1|}} q_{1,|w_1|+1} \xrightarrow{t_1} q_{2,1} ... q_{p,|w_p|+1} \xrightarrow{t_p} q_{p+1,1} \xrightarrow{t_{p+1,1}} ... \xrightarrow{t_{p+1,|w_{p+1}|}} q_{p+1,|w_{p+1}|+1}$$

où pour tout $1 \leq i \leq p+1$, $1 \leq j \leq |w_i|$,

$$\begin{cases} etiq(q_{i,1} \xrightarrow{t_{i,1}} ... \xrightarrow{t_{i,|w_i|}} q_{i,|w_i|+1}) = w_i \\ t_{i,j} \in T \\ Si \ 1 \leq i \leq p, \ t_i = (q_{i,|w_i|+1}, a, 1, q_{i+1,1}) \in T' \backslash T \\ q_{1,1} \in I, q_{p+1,|w_{p+1}|+1} \in F \end{cases}$$

On en déduit immédiatement que pour tout $1 \leq i \leq p+1$,

$$\begin{cases} etiq(q_{i,1} \xrightarrow{t_{i,1}} ... \xrightarrow{t_{i,|w_i|}} q_{i,|w_i|+1}) = w_i \\ q_{i,1} \in I \\ q_{i,|w_i|+1} \in F \end{cases}$$

Tous les $q_{i,1} \xrightarrow{t_{i,1}} ... \xrightarrow{t_{i,|w_i|}} q_{i,|w_i|+1}$ sont donc des chemins acceptants étiquetés par les $w_i$ dans $\mathcal{A}$. Ainsi :

$$p_{\mathcal{A}'}(w) = poids_{\mathcal{A}'}(c) = \sum_{1 \leq i \leq p+1} poids_{\mathcal{A}}(q_{i,1} \xrightarrow{t_{i,1}} ... \xrightarrow{t_{i,|w_i|}} q_{i,|w_i|+1}) + p$$

$$\leq \sum_{1\leq i\leq p+1} p_{\mathcal{A}}(q_{i,1} \xrightarrow{t_{i,1}} ... \xrightarrow{t_{i,|w_i|}} q_{i,|w_i|+1}) + p$$

Inversement, si on a $(c_i)_{1\leq i\leq p+1}$ chemins acceptants maximaux des $(w_i)_{1\leq i\leq p+1}$ dans $\mathcal{A}$, où pour tout $1 \leq i \leq p+1$,

$$c_i = q_{i,1} \xrightarrow{t_{i,1}} ... \xrightarrow{t_{i,|w_i|}} q_{i,|w_i|+1}$$

on peut définir $c$ un chemin acceptant de $w$ dans $\mathcal{A}'$. En notant, pour tout $1 \leq i \leq p$, $t_i = (q_{i,|w_i|+1}, a, 1, q_{i+1,1})$ :

$$c = c_1 \xrightarrow{t_1} ... \xrightarrow{t_p} c_{p+1}$$

En effet, pour tout $1 \leq i \leq p$, $(q_{i,|w_i|+1}, a, 1, q_{i+1,1}) \in T'$, car $q_{i,|w_i|+1} \in F$ et $q_{i+1,1} \in I$. Ainsi,

$$p_{\mathcal{A}'}(w) \geq poids_{\mathcal{A}'}(c) = poids_{\mathcal{A}'}(c_1 \xrightarrow{t_1} ... \xrightarrow{t_p} c_{p+1}) = \sum_{1\leq i\leq p+1} poids_{\mathcal{A}}(c_i) + p = \sum_{1\leq i\leq p+1} p_{\mathcal{A}}(c_i) + p$$

On en déduit que :

$$p_{\mathcal{A}'}(w) = \sum_{1\leq i\leq p+1} p_{\mathcal{A}}(c_i) + p$$

**Exemple 6.** *La Figure 13 représente un automate $\mathcal{A} = (Q, I, F, T)$, sur un alphabet $\Sigma$, et l'automate calculant $f_3$.*


(a) Automate max-plus $\mathcal{A}$

(b) Automate obtenu à partir de $\mathcal{A}$, calculant $f_3$

FIGURE 13 – Exemple d'application du Lemme 2

— Soit $\mathcal{A} = (Q, I, F, T)$ un automate max-plus sur l'alphabet $\Sigma$. Notons $\mathcal{A}' = (Q \sqcup \{start, stop\}, I \sqcup \{start\}, F \sqcup \{stop\}, T')$ sur $\Sigma \cup \{a\}$ où

$T' = T \sqcup \{(start, a, 0, i),\ i \in I\} \sqcup \{(f, a, 0, stop),\ f \in F\} \sqcup \{(start, l, 0, start),\ l \in \Sigma \cup \{a\}\} \sqcup \{(stop, l, 0, stop),\ l \in \Sigma \cup \{a$

Soit $w$ un mot sur $\Sigma \cup \{a\}$, $w = w_1 a ... a w_{p+1}$ où les $(w_i)_{1\leq i\leq p+1}$ sont définis sur $\Sigma$.
Soit $c$ un chemin acceptant de $w$ dans $\mathcal{A}'$. Il peut être de la forme :

$$\begin{cases} start \xrightarrow{t_1} ...start \xrightarrow{t_{k_1}} q_1 \xrightarrow{t_{k_1+1}} ... \xrightarrow{t_{k_1+k_2-1}} q_{k_2} \xrightarrow{t_{k_1+k_2}} stop \xrightarrow{t_{k_1+k_2+1}} ... \xrightarrow{t_{|w|}} stop \\ start \xrightarrow{t_1} ...start \xrightarrow{t_{k_1}} q_1 \xrightarrow{t_{k_1+1}} ... \xrightarrow{t_{|w|}} q_{|w|-k_1+1} \\ q_1 \xrightarrow{t_1} ... \xrightarrow{t_{k_2-1}} q_{k_2} \xrightarrow{t_{k_2}} stop \xrightarrow{t_{k_2+1}} ... \xrightarrow{t_{|w|}} stop \\ q_1 \xrightarrow{t_1} ... \xrightarrow{t_{|w|}} q_{|w|+1} \end{cases}$$

— Si le chemin est la première forme, on a $q_1 \in I$, $q_{k_2} \in F$. De plus, $t_{k_1}$ et $t_{k_1+k_2}$ sont étiquetées par la lettre $a$, et aucune transition $t_i$, $k_1 < i < k_1 + k_2$ n'est étiquetée par $a$, car ce sont des transitions de $T$. Ainsi, le chemin $q_1 \xrightarrow{t_{k_1+1}} ... \xrightarrow{t_{k_1+k_2-1}} q_{k_2}$, étiqueté par un mot $w'$, est acceptant dans $\mathcal{A}$, et $aw'a$ est un facteur de $w$. Donc il existe $1 < i < p+1$ tel que $w' = w_i$. Enfin,

$$\begin{aligned} poids_{\mathcal{A}'}(c) &= 0 + poids_{\mathcal{A}'}(q_1 \xrightarrow{t_{k_1+1}} ... \xrightarrow{t_{k_1+k_2-1}} q_{k_2}) + 0 \\ &= poids_{\mathcal{A}}(q_1 \xrightarrow{t_{k_1+1}} ... \xrightarrow{t_{k_1+k_2-1}} q_{k_2}) \\ &\leq p_{\mathcal{A}}(w') = p_{\mathcal{A}}(w_i) \end{aligned}$$

— Si le chemin est de la deuxième forme, avec le même raisonnement que précedemment, on obtient que $t_{k_1}$ est étiquetée par $a$, $q_1 \xrightarrow{t_{k_1+1}} ... \xrightarrow{t_{|w|}} q_{|w|-k_1+1}$ est un chemin acceptant dans $\mathcal{A}$, étiqueté par un mot $w'$, et $aw'$ est un suffixe de $w$, donc :

$$poids_{\mathcal{A}'}(c) = 0 + poids_{\mathcal{A}'}(q_1 \xrightarrow{t_{k_1+1}} ... \xrightarrow{t_{|w|}} q_{|w|-k_1+1})$$

$$= poids_{\mathcal{A}}(q_1 \xrightarrow{t_{k_1+1}} ... \xrightarrow{t_{|w|}} q_{|w|-k_1+1})$$

$$\leq p_{\mathcal{A}}(w_{p+1})$$

— Si le chemin est de la troisième forme, alors symétriquement :

$$poids_{\mathcal{A}'}(c) = poids_{\mathcal{A}'}(q_1 \xrightarrow{t_1} ... \xrightarrow{t_{k_2-1}} q_{k_2}) + 0$$

$$\leq p_{\mathcal{A}}(w_1)$$

— Si le chemin est de la quatrième forme, alors $w$ ne contient aucun $a$, et ainsi :

$$poids_{\mathcal{A}'}(c) = poids_{\mathcal{A}}(c) \leq p_{\mathcal{A}}(w)$$

Réciproquement, considérons pour $i$ fixé $c = q_1 \to ... \to q_{|w_i|+1}$ le chemin acceptant de $w_i$ dans $\mathcal{A}$ tel que $p_{\mathcal{A}}(w_i) = poids_{\mathcal{A}}(c)$. On construit le chemin dans $\mathcal{A}'$ :

— $c' = start \to ... \to start \to c \to stop \to ... \to stop$ si $1 < i < p+1$,

— $c' = start \to ... \to start \to c$ si $i = p+1$,

— $c' = c \to stop \to ... \to stop$ si $i = 1$,

— $c' = c$ si $1 = i = p+1$.

acceptant et étiqueté par $w$. C'est possible car :

— On peut étiqueter les transitions partant de *start* et entrant dans *start* par n'importe quelle lettre,

— On peut étiqueter les transitions partant de *stop* et entrant dans *stop* par n'importe quelle lettre,

— Le premier état de $c$ est initial,

— Le dernier état de $c$ est acceptant.

Dans tous ces cas,

$$p_{\mathcal{A}'}(w) \geq poids_{\mathcal{A}'}(c') = poids_{\mathcal{A}}(c) = p_{\mathcal{A}}(w_i)$$

On en déduit que pour tout $w$ dans $\mathcal{A}'$ :

$$\begin{cases} \exists\ i,\ p_{\mathcal{A}'}(w) \leq p_{\mathcal{A}}(w_i) \\ \forall\ i,\ p_{\mathcal{A}}(w_i) \leq p_{\mathcal{A}'}(w) \end{cases}$$

Ainsi,

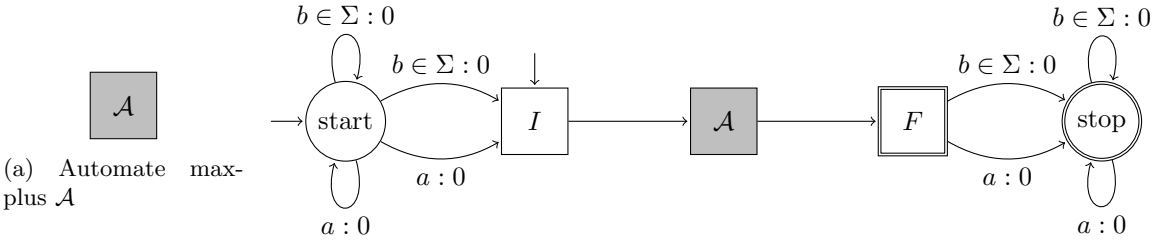
$$p_{\mathcal{A}'}(w) = \max_{1 \leq i \leq p+1} (p_{\mathcal{A}}(w_i))$$

**Exemple 7.** *La Figure 14 représente un automate $\mathcal{A} = (Q, I, F, T)$, sur un alphabet $\Sigma$, et l'automate calculant $f_4$ où $a \notin \Sigma$.*

□

# B Preuve de la Proposition 4

Pour prouver ce théorème, plusieurs résultats préalables doivent être prouvés.




(a) Automate max-plus $\mathcal{A}$

(b) Automate obtenu à partir de $\mathcal{A}$, calculant $f_4$

FIGURE 14 – Exemple d'application du Lemme 2

**Notations.** Soit $T = \tau_1 \xrightarrow{t_1} ...$ une trace d'un Size-Change Abstraction $(\mathcal{K}, I, F, \chi, \mathcal{T})$. En notant pour tout $i \leq \ell(T)$, $t_i = (q_i, a_i, q_{i+1})$ :

— Si $M \geq 0$, et $\nu$ une valuation, on note $\nu + M$ la valuation telle que pour tout $x \in \chi$ :

$$(\nu + M)(x) = \nu(x) + M$$

— On dit que le mot $t_1 t_2 ..$ est le mot *associé à la trace*. Il est dit acceptant si la trace est acceptante.

— Si $\nu, \nu'$ des valuations, on note $\nu \leq \nu'$ si pour tout $x \in \chi$, $\nu(x) \leq \nu'(x)$.

— De manière similaire, si $\nu, \nu'$ des valuations, on note $\nu < \nu'$ si pour tout $x \in \chi$, $\nu(x) < \nu'(x)$.

— On dit que $x_j, ..., x_k$ est une *chaine d'inégalités* de $T$ si pour tout $j \leq i < k$, $x'_{i+1} \leq x_i$ ou $x'_{i+1} < x_i$ est dans $a_i$. On note de plus $\mathcal{C}(T)$ l'ensemble des chaines de la trace $T$.

— Si $c = x_j, ..., x_k \in \mathcal{C}(T)$, alors on note $strict(c) \in \mathbb{N} \cup \{+\infty\}$ le nombre d'inégalités strictes de cette chaine, soit :

$$strict(c) = |\{j \leq i < k \mid x'_{i+1} < x_i \in a_i\}|$$

**Résultats.** Tout d'abord, nous établirons la co-accessiblité des cycles avec le Lemme 3.

Puis, nous prouverons les Lemmes 4, 5, 6, 7 et Corollaires 2, 3, utiles pour la preuve de la Proposition 5 qui établit l'existence d'une trace infinie bornée dans le cas où une trace infinie existe pour un Size-Change Abstraction donné.

Ensuite, nous prouverons

— le Lemme 8 concernant les Size-Change Abstractions,

— les Lemmes 9, 10, et 11, ainsi que le Corollaire 4 concernant les liens entre traces et mots d'un Size-Change Abstraction et de son automate max-plus associé,

qui seront des outils précieux pour la preuve finale.

Enfin, on prouvera le thèorème 4.

**Lemme 3** (Co-accessibilité des cycles)**.** *Étant donné un programme $P$, $\mathcal{S} = (Q, I, F, \chi, \mathcal{T})$ son Size-Change Abstraction associé, et $\mathcal{A}$ son automate max-plus associé, alors tous les cycles de $\mathcal{A}$ sont co-accessibles.*

*Démonstration.* Soit $C = (q_1, x_1) \xrightarrow{t_1:p_1} ... \xrightarrow{t_n:p_n} (q_{n+1}, x_{n+1})$ un cycle de $\mathcal{A}$, c'est-à-dire où $q_1 = q_{n+1}$. Pour tout $1 \leq i \leq n+1$, $x_i \in \chi \cup \{start, stop\}$ et $q_i \in Q$.Prouvons que ce cycle est co-accessible. Il contient au moins une transition, $((q_1, x_1), t_1, p_1, (q_2, x_2))$. Par construction, puisque $((q_1, x_1), t_1, p_1, (q_2, x_2))$ est une transition de $\mathcal{A}$, alors $((q_1, x_1), t_1, p_1, (q_2, stop))$ est également une transition de $\mathcal{A}$. Or, tous les états de $\mathcal{S}$ sont co-accessibles, donc il existe un chemin $q_2 \xrightarrow{t'2} ... \xrightarrow{t'm} q_{m+1}$ dans $\mathcal{S}$ où $q_{m+1} \in F$. Par construction de l'ensemble des transitions de $\mathcal{A}$ : pour tout $2 \leq i \leq m$, $((q_i, stop), t'_i, 0, (q_{i+1}, stop))$ est une transition de $\mathcal{A}$. Ainsi, on peut construire le chemin suivant dans $\mathcal{A}$ :

$$(q_1, x_1) \xrightarrow{((q_1,x_1),t_1,p_1,(q_2,x_2))} (q_2, stop) \xrightarrow{((q_2,stop),t'_2,0,(q_3,stop))} ... \xrightarrow{((q_m,stop),t'_m,0,(q_{m+1},stop))} (q_{m+1}, stop)$$

et de plus $(q_{m+1}, stop)$ est final. $C$ est donc co-accessible dans $\mathcal{A}$, d'où le résultat. □

**Lemme 4** (Construction de trace). *Étant donné $\mathcal{S} = (Q, I, F, \chi, \mathcal{T})$ un Size-Change Abstraction, si $(q_1, a_1, q_2), ..., (q_n, a_n, q_{n+1})$ suite d'éléments de $\mathcal{T}$ et $\tau_{n+1}$ est une valuation sur $\chi$, alors on peut construire une trace $T = \tau_1 \xrightarrow{(q_1,a_1,q_2)} ... \xrightarrow{(q_n,a_n,q_{n+1})} \tau_{n+1}$ bornée par $\max_{x\in\chi}(\tau_{n+1}(x)) + n + 1$.*

*Si de plus $q_1 \in I$ et $q_{n+1} \in F$, alors cette trace est acceptante.*

*Démonstration.* Soit $\mathcal{S} = (Q, I, F, \chi, \mathcal{T})$ un Size-Change Abstraction, et $(q_1, a_1, q_2), ..., (q_n, a_n, q_{n+1})$ une suite d'éléments de $\mathcal{T}$. On pose $M = \max_{x\in\chi}(\tau_{n+1}(x)) + 1$ et, pour tout $1 \leq i \leq n+1$, pour tout $x \in \chi$,

$$\tau_i(x) = M + n - i$$

Prouvons que $T = \tau_1 \xrightarrow{(q_1,a_1,q_2)} ... \xrightarrow{(q_n,a_n,q_{n+1})} \tau_{n+1}$ est une trace :

- $(q_1, a_1, q_2), ..., (q_n, a_n, q_{n+1})$ représente un chemin dans $\mathcal{S}$
- pour tout $1 \leq i < n$, pour tout $x, y \in \chi$,
  - Si $x' \leq y \in a_i$, alors $M + n - i = \tau_i(y) \geq \tau_{i+1}(x) = M + n - i - 1$.
  - Si $x' < y \in a_i$, alors $n + 1 - i = \tau_i(y) > \tau_{i+1}(x) = n - i$.
- Maintenant, si $i = n$ :
  - Si $x' \leq y \in a_n$, alors $M = \tau_i(y) \geq \tau_{i+1}(x)$.
  - Si $x' < y \in a_n$, alors $M = \tau_i(y) > \tau_{i+1}(x)$.

Les valuations recpectent donc les ensembles d'inégalités des transitions. $T$ est donc une trace, et de plus bornée par $M + n$. Si de plus $q_1 \in I$ et $q_{n+1} \in F$, alors cette trace est acceptante. □

**Corollaire 2** (Construction de trace). *Étant donné $\mathcal{S} = (Q, I, F, \chi, \mathcal{T})$ un Size-Change Abstraction, si $(q_1, a_1, q_2), ..., (q_n, a_n, q_{n+1})$ suite d'éléments de $\mathcal{T}$, alors on peut construire une trace $T = \tau_1 \xrightarrow{(q_1,a_1,q_2)} ... \xrightarrow{(q_n,a_n,q_{n+1})} \tau_{n+1}$ bornée par $n$.*

*Si de plus $q_1 \in I$ et $q_{n+1} \in F$, alors cette trace est acceptante.*

*Démonstration.* On pose $\tau_{n+1} = 0$, et on a le résultat immédiatement par le Lemme 4. □

**Corollaire 3** (Caractérisation des mots associés à une trace). *Étant donné $\mathcal{S} = (Q, I, F, \chi, \mathcal{T})$ un Size-Change Abstraction, si $(q_1, a_1, q_2), ..., (q_n, a_n, q_{n+1})$ suite d'éléments de $\mathcal{T}$, alors $(q_1, a_1, q_2)...(q_n, a_n, q_{n+1})$ est un mot associé à une trace.*

*Si de plus $q_1 \in I$ et $q_{n+1} \in F$, alors ce mot est acceptant.*

*Démonstration.* Par le Corollaire 2, il existe une trace $T = \tau_1 \xrightarrow{(q_1,a_1,q_2)} ... \xrightarrow{(q_n,a_n,q_{n+1})} \tau_{n+1}$, acceptante si et seulement si $q_1 \in I$ et $q_{n+1} \in F$, d'où le résultat. □

**Lemme 5** (Assemblage de traces). *Étant donné $\mathcal{S} = (Q, I, F, \chi, \mathcal{T})$ un Size-Change Abstraction, si :*

- *$T_1 = \tau_1 \xrightarrow{(q_1,a_1,q_2)} ... \xrightarrow{(q_n,a_n,q_{n+1})} \tau_{n+1}$ est trace telle que $q_1$ est initial*
- *$T_2 = \tau_{n+1} \xrightarrow{(q_{n+1},a_{n+1},q_{n+2})} ...$ est une trace infinie*

*Alors $T = \tau_1 \xrightarrow{(q_1,a_1,q_2)} ... \xrightarrow{(q_n,a_n,q_{n+1})} \tau_{n+1} \xrightarrow{(q_{n+1},a_{n+1},q_{n+2})} ...$ est une trace infinie.*

*Démonstration.* Soit $\mathcal{S} = (Q, I, F, \chi, \mathcal{T})$ un Size-Change Abstraction, et $T_1$, $T_2$, et $T$ des traces telles que ci-dessus, alors :

- $q_1 \in I$
- Si $1 \leq i \leq n$, alors les valuations $\tau_i$ et $\tau_{i+1}$ respectent $a_i$ car $T_1$ est une trace.
- Si $n \leq i$, alors les valuations $\tau_i$ et $\tau_{i+1}$ respectent $a_i$ car $T_2$ est une trace.

Donc $T$ est une trace telle que $q_1 \in I$. □

**Lemme 6** (Répétition de cycles dans une trace). *Étant donné $\mathcal{S} = (Q, I, F, \chi, \mathcal{T})$ un Size-Change Abstraction, si $C = \tau_1 \xrightarrow{(q_1,a_1,q_2)} ... \xrightarrow{(q_n,a_n,q_1)} \tau_1$ est une trace. Alors $T = \tau_1 \xrightarrow{(q_1,a_1,q_2)} ... \xrightarrow{(q_n,a_n,q_1)} \tau_1 \xrightarrow{(q_1,a_1,q_2)} ... \xrightarrow{(q_n,a_n,q_1)} \tau_1...$ est une trace infinie bornée.*

*Démonstration.* Soit $\mathcal{S} = (Q, I, F, \chi, \mathcal{T})$ un Size-Change Abstraction, et $C = \tau_1 \xrightarrow{(q_1,a_1,q_2)} ... \xrightarrow{(q_n,a_n,q_1)} \tau_1$ une trace. Posons $T = \tau_1 \xrightarrow{(q_1,a_1,q_2)} ... \xrightarrow{(q_n,a_n,q_1)} \tau_1 \xrightarrow{(q_1,a_1,q_2)} ... \xrightarrow{(q_n,a_n,q_1)} \tau_1 ...$ Si $i > 0$, notons $i = qn+j$ la division euclidienne de $i$ par $n$. Le $i^{eme}$ état de $T$ est $q_j$, par périodicité de $T$.

— Si $1 \leq j < n$, alors puisque $C$ est une trace, $\tau_i$ et $\tau_{i+1}$ respectent $a_i$.

— Si $j = n$, alors puisque $C$ est une trace, $\tau_n$ et $\tau_1$ respectent $a_n$.

$T$ est donc une trace infinie. □

**Lemme 7** (Élévation d'une trace)**.** *Étant donné $\mathcal{S} = (Q, I, F, \chi, \mathcal{T})$ un Size-Change Abstraction, si $T = \tau_1 \xrightarrow{(q_1,a_1,q_2)} ... \xrightarrow{(q_n,a_n,q_{n+1})} \tau_{n+1}$ est une trace, et $\tau'_1 \geq \tau_1$, alors $T' = \tau'_1 \xrightarrow{(q_1,a_1,q_2)} \tau_2 ... \xrightarrow{(q_n,a_n,q_{n+1})} \tau_{n+1}$ est une trace.*

*Démonstration.* Soit $\mathcal{S} = (Q, I, F, \chi, \mathcal{T})$ un Size-Change Abstraction, et $T = \tau_1 \xrightarrow{(q_1,a_1,q_2)} ... \xrightarrow{(q_n,a_n,q_{n+1})} \tau_{n+1}$ une trace, et $\tau'_1 \geq \tau_1$ une valuation. Notons $T' = \tau'_1 \xrightarrow{(q_1,a_1,q_2)} \tau_2 ... \xrightarrow{(q_n,a_n,q_{n+1})} \tau_{n+1}$.

— Si $1 < i \leq n$, alors $\tau_i$ et $\tau_{i+1}$ respectent $a_i$.

— Si $i = 1$, alors pour tout $x, y \in \chi$

  — Si $x' \leq y \in a_1$, alors $\tau'_1(y) \geq \tau_1(y) \geq \tau_2(x)$

  — Si $x' < y \in a_1$, alors $\tau'_1(y) \geq \tau_1(y) > \tau_2(x)$

Ainsi, $T'$ est une trace. □

**Proposition 5** (Traces infinies bornées)**.** *Si un Size-Change Abstraction admet une trace acceptante infinie, alors il admet une trace acceptante infinie bornée.*

*Démonstration.* Soient $\mathcal{S}$ un Size-Change Abstraction $(Q, I, F, \chi, \mathcal{T})$, et $T = \tau_1 \xrightarrow{(q_1,a_1,q_2)} ...$ une trace infinie de $\mathcal{S}$ telle que $q_1 \in I$ si elle existe.

Tout d'abord, prouvons qu'il existe une trace cyclique $C = \tau_j \xrightarrow{(q_j,a_j,q_{j+1})} ... \xrightarrow{(q_{k-1},a_{k-1},q_j)} \tau_k$ dans $T$ tel que $\tau_j \leq \tau_k$. Étant donné que $Q$ est fini et que $T$ est infinie, il existe $q \in Q$ qui apparait une infinité de fois dans $T$. Considérons la suite des occurences de $q$ dans $T$, $(i_n)_{n\in\mathbb{N}}$. On va numéroter les variables de $\chi$ : $\chi = \{y_1, ..., y_N\}$. Trouvons un cycle tel que décrit ci-dessus partant de $q$.

— Il existe une sous-suite de $(i_n)_{n\in\mathbb{N}}$ , que l'on va noter $(i_{f(n)})_{n\in\mathbb{N}}$ telle que pour tout $j > 0$,

$$\tau_{i_{f(j)}}(y_1) \leq \tau_{i_{f(j+1)}}(y_1)$$

  En effet, soit la suite est bornée, et on peut en extraire une sous-suite stationnaire, soit une sous-suite tend vers l'infini, et alors on peut en extraire une sous-suite croissante.

— On réitère ce procédé avec la sous-suite obtenue et la variable suivante, et ceci jusqu'à avoir traité toutes les variables.

On obtient donc une sous-suite de $(i_n)_{n\in\mathbb{N}}$ notée $(i_{g(n)})_{n\in\mathbb{N}}$ telle que pour tout $x \in \chi$, pour tout $j > 0$

$$\tau_{i_{g(j)}}(x) \leq \tau_{i_{g(j+1)}}(x)$$

Et ainsi,

$$\tau_{i_{g(1)}} \leq \tau_{i_{g(2)}}$$

On prend $C = \tau_{i_{g(1)}} \xrightarrow{(q_{i_{g(1)}},a_{i_{g(1)}},q_{i_{g(1)}+1})} ... \xrightarrow{(q_{i_{g(2)}-1},a_{i_{g(2)}-1},q_{i_{g(1)}})} \tau_{i_{g(2)}}$. Ce cycle est en effet tel que $\tau_{i_{g(1)}} \leq \tau_{i_{g(2)}}$.

Maintenant, déduisons-en l'existence d'une trace acceptante infinie bornée pour $\mathcal{S}$.

Par le Lemme 7, $C' = \tau_{i_{g(2)}} \xrightarrow{(q_{i_{g(1)}},a_{i_{g(1)}},q_{i_{g(1)}+1})} ... \xrightarrow{(q_{i_{g(2)}-1},a_{i_{g(2)}-1},q_{i_{g(1)}})} \tau_{i_{g(2)}}$ est une trace. Par le Lemme 6, en mettant ce cycle $C'$ bout à bout à l'infini, on obtient une trace infinie et bornée. Or, il y a un chemin d'états $q_1, ..., q_{i_{g(1)}}$ tel que $q_1 \in I$, que l'on peut extraire de la trace acceptante $T$. Par le Lemme 4, on peut construire une trace $T''$ passant par ces états et telle que la dernière valuation soit $\tau_{i_{g(2)}}$. De plus, cette trace est bornée. Enfin, par le Lemme 5, on peut assembler les traces $T'$ et $T''$ pour former une trace acceptante infinie bornée. □

**Lemme 8** (Lien chaines-valuations)**.** *Étant donnés $\mathcal{S}$ un Size-Change Abstraction $(Q, I, F, \chi, \mathcal{T})$, et $T = \tau_1 \xrightarrow{t_1} ...$ une trace de $\mathcal{S}$ telle que $\mathcal{C}(T) \neq \emptyset$, alors il existe une trace $T' = \tau'_1 \xrightarrow{t_1} ...$ bornée par $\max_{c\in\mathcal{C}(T)}(strict(c))$. Il n'existe pas de trace $T'' = \tau''_1 \xrightarrow{t_1} ...$ bornée par un entier strictement inférieur à $\max_{c\in\mathcal{C}(T)}(strict(c))$.*

*Démonstration.* Soient $\mathcal{S}$ un Size-Change Abstraction $(Q, I, F, \chi, \mathcal{T})$, et $T = \tau_1 \xrightarrow{t_1} ...$ une trace de $\mathcal{S}$ telle que $\mathcal{C}(T) \neq \emptyset$. On note pour tout $1 \leq i \leq \ell(T)$, $t_i = (q_i, a_i, q'_i)$.

— Supposons par l'absurde qu'il existe une trace $T' = \tau'_1 \xrightarrow{t_1} ...$ bornée par $N < \max_{c\in\mathcal{C}(T)}(strict(c))$. Puisque $T$ et $T'$ suivent les mêmes transitions, elles ont le même ensemble de chaines. Considérons la chaine de $T'$ $c = x_j, ..., x_k$ telle que $strict(c)$ soit maximale. Puisque la trace est bornée par $N$, $\tau'_j(x_j) \leq N$. Puis pour tout $j < i < k$,
  — Si $x'_{i+1} < x_i \in a_i$, alors $\tau'_{i+1}(x_{i+1}) \leq \tau'_i(x_i) - 1$
  — Si $x'_{i+1} \leq x_i \in a_i$, alors $\tau'_{i+1}(x_{i+1}) \leq \tau'_i(x_i)$

  Il y a $\max_{c\in\mathcal{C}(T)}(strict(c))$ inégalités strictes dans cette chaine, donc :

$$\tau'_k(x_k) \leq \tau'_j(x_j) - \max_{c\in\mathcal{C}(T)}(strict(c)) \leq N - \max_{c\in\mathcal{C}(T)}(strict(c)) \leq 0$$

  C'est absurde.

— Prouvons à présent qu'il existe une trace $T' = \tau'_1 \xrightarrow{t_1} ...$ bornée par $K = \max_{c\in\mathcal{C}(T)}(strict(c))$. Construisons-la ainsi :
  — Pour tout $x \in \chi$, $\tau'_1(x) = K$
  — Pour tout $1 \leq i < k$, pour tout $x \in \chi$ tel que $\{y \mid x' \leq y \text{ ou } x' < y \in a_i\} \neq \emptyset$ alors :

$$\tau'_{i+1}(x) = \min\left(\min_{\substack{y\in\chi \\ x'<y\in a_i}}(\tau'_i(y) - 1), \min_{\substack{y\in\chi \\ x'\leq y\in a_i}}(\tau'_i(y))\right)$$

  — Sinon, $\tau'_{i+1}(x) = K$.

  $T'$ définie ainsi est une trace. Les transitions se succèdent correctement, car ce sont les mêmes que pour le trace $T$. Les valuations respectent les inégalités des transitions, immédiatement par la définition de $T'$. Prouvons donc que toutes les valuations sont à valeurs positives. Supposons par l'absurde qu'il existe $1 \leq i \leq k$ et $x_i \in \chi$ tels que $\tau'_i(x_i) < 0$. On va construire une chaine $c$ telle que $strict(c) > K$ récursivement.

  — Tout d'abord, il existe $x_{i-1} \in \chi$ tel que $x'_i \leq x_{i-1}$ ou $x'_i < x_{i-1}$ appartient à $a_{i-1}$, sinon on aurait $\tau'_i(x_i) = K$. On a donc une chaine de taille au moins 1 : $c_1 = x_{i-1}, x_i$, et $\tau'_{i-1}(x_{i-1}) = \tau'_i(x_i) + strict(c_1)$.
    En effet, si $x'_i \leq x_{i-1} \in a_{i-1}$, alors $\tau'_{i-1}(x_{i-1}) = \tau'_i(x_i) = \tau'_i(x_i) + strict(c_1)$, et sinon $\tau'_{i-1}(x_{i-1}) = \tau'_i(x_i) + 1 = \tau'_i(x_i) + strict(c_1)$.
  — Supposons qu'à $k \leq 1$ donné, il existe une chaine de longueur $k + 1$ $c_k = x_{i-k}, ..., x_i$, où $\tau'_{i-k}(x_{i-k}) = \tau'_i(x_i) + strict(c_k)$. Si $\tau'_{i-k}(x_{i-k}) = K$, alors on s'arrête. Sinon, $i - k - 1 \neq 1$, et il existe $x_{i-k-1} \in \chi$ tel que $x'_{i-k} \leq x_{i-k-1}$ ou $x'_{i-k} < x_{i-k-1}$ appartienne à $a_{i-k-1}$, sinon par définition $\tau'_{i-k}(x_{i-k}) = K$.
    On pose $c_{k+1}$ la chaine de longueur $k + 2$ : $x_{i-k-1}, ..., x_i$. De plus, si $x'_{i-k} \leq x_{i-k-1} \in a_{i-k-1}$, alors $\tau'_{i-k-1}(x_{i-k-1}) = \tau'_{i-k}(x_{i-k}) = \tau'_i(x_i) + strict(c_k) = \tau'_i(x_i) + strict(c_{k+1})$ Sinon, $\tau'_{i-k-1}(x_{i-k-1}) = \tau'_{i-k}(x_{i-k}) + 1 = \tau'_i(x_i) + strict(c_k) + 1 = \tau'_i(x_i) + strict(c_{k+1})$.

  On obtient donc une chaine $c_L = x_{i-L}, .., x_i$, $L \in \mathbb{N}$, de longueur $L + 1$, telle que

$$\begin{cases} \tau'_i(x_i) < 0 \\ \tau'_{i-L}(x_{i-L}) = K \\ \tau'_{i-L}(x_{i-L}) = \tau'_i(x_i) + strict(c_L) \end{cases}$$

  Soit : $strict(c_L) > K$, ce qui est absurde. On a donc le résultat : $T'$ est une trace bornée par $K = \max_{c\in\mathcal{C}(T)}(strict(c))$ de $\mathcal{S}$.

□

**Lemme 9** (Mots non associés à une trace)**.** *Soit $\mathcal{S}$ un Size-Change Abstraction, et $\mathcal{A} = \mathcal{A}_1 \sqcup \mathcal{A}_2$ son automate max-plus associé. Étant donné un mot $w = t_1...t_n$ non acceptant, alors :*

$$w \notin \mathcal{L}(\mathcal{A}_1)$$

*Démonstration.* Soit $\mathcal{S} = (Q, I, F, \chi, \mathcal{T})$ un Size-Change Abstraction, et $\mathcal{A} = \mathcal{A}_1 \sqcup \mathcal{A}_2$ son automate max-plus associé, et $w = t_1...t_n$ un mot non acceptant. Supposons par l'absurde que $w \in \mathcal{L}(\mathcal{A}_1)$. Alors, en notant pour tout $1 \leq i \leq n$, $t_i = (q_i, a_i, q'_i)$ :

- $q_1 \in I$. En effet, si $w$ est accepté, il existe un chemin démarrant d'un état initial de $\mathcal{A}_1$ dont la première transition est étiquetée par $t_1$. Ainsi, $q_1 \in I$. Or, dans $\mathcal{A}_1$, une transition étiquetée par $t_1$ ne peut démarrer que d'un état de la forme $(q_1, x)$ où $x \in \chi \cup \{start, stop\}$. De plus, les états initiaux de $\mathcal{A}_1$ sont un couple formé d'un état de $I$ et de $start$. D'où le résultat précédent.
- Par le même raisonnement, $q'_n \in F$.
- Pour tout $1 \leq i < n$, $q'_i = q_{i+1}$. En effet, par construction, pour tout $1 \leq i \leq n$, $q'_i = q_{i+1}$, une transition étiquetée par $t_i$ dans $\mathcal{A}_1$ part d'un état de la forme $(q_i, x)$ où $x \in \chi \cup \{start, stop\}$ et arrive dans un état $(q'_i, x')$ où $x' \in \chi \cup \{start, stop\}$. S'il existe un chemin acceptant de $w$ dans $\mathcal{A}_1$, alors pour tout $1 \leq i < n$, $q'_i = q_{i+1}$, ce chemin contient une transition étiquetée par $t_i$ qui arrive sur un état duquel part une transition étiquetée par $t_{i+1}$. Ainsi, par identification, $q'_1 = q_{i+1}$.

La suite $t_1, .., t_n$ a toutes les propriétés nécessaires à l'application du Lemme 4. Il existe donc une trace $\tau_1 \xrightarrow{t_1} ... \xrightarrow{t_n} \tau_{n+1}$. C'est absurde. On a donc le résultat recherché. □

**Corollaire 4** (Mots non associés à une trace)**.** *Soit $\mathcal{S}$ un Size-Change Abstraction, et $\mathcal{A} = \mathcal{A}_1 \sqcup \mathcal{A}_2$ son automate max-plus associé. Étant donné un mot $w = t_1...t_n$ qui n'est pas acceptant, alors :*

$$p_{\mathcal{A}}(w) = p_{\mathcal{A}_2}(w) = |w|$$

*Réciproquement, si un mot est acceptant :*

$$p_{\mathcal{A}}(w) = p_{\mathcal{A}_1}(w)$$

*Démonstration.* Par le Lemme 9, $w \notin \mathcal{L}(\mathcal{A}_1)$. donc $p_{\mathcal{A}_1}(w) = -\infty$. Or, par construction de $\mathcal{A}_2$, $w \in \mathcal{L}(\mathcal{A}_2)$, et donc $w$ est accepté par $\mathcal{A}_2$ et $p_{\mathcal{A}_2}(w) = |w|$. Ainsi :

$$p_{\mathcal{A}}(w) = \max\left(p_{\mathcal{A}_1}(w), p_{\mathcal{A}_2}(w)\right) = p_{\mathcal{A}_2}(w) = |w|$$

Réciproquement, si un mot est acceptant, alors il appartient au complémentaire de $\mathcal{L}(\mathcal{A}_2)$, qui est en fait par définition $\mathcal{L}(\mathcal{A}_1)$. Ainsi :

$$p_{\mathcal{A}}(w) = p_{\mathcal{A}_1}(w)$$

□

**Lemme 10** (Lien chaines-poids)**.** *Étant donnés un Size-Change Abstraction $\mathcal{S}$ d'ensemble de transitions $\mathcal{T}$, et son automate max-plus associé $\mathcal{A}$, si $T = \tau_1 \xrightarrow{t_1} ... \xrightarrow{t_n} \tau_{n+1}$ une trace acceptante de $\mathcal{S}$ :*

$$p_{\mathcal{A}}(t_1...t_n) = \begin{cases} \max_{c \in \mathcal{C}(T)}(strict(c)) & si\ \mathcal{C}(T) \neq \emptyset \\ |w| & sinon \end{cases}$$

*Démonstration.* Soient $\mathcal{S}$ un Size-Change Abstraction d'ensemble de transitions $\mathcal{T}$, et $\mathcal{A} = \mathcal{A}_1 \sqcup \mathcal{A}_2$ son automate max-plus associé d'ensemble de transitions $T'$. $T = \tau_1 \xrightarrow{t_1} ... \xrightarrow{t_n} \tau_{n+1}$ une trace acceptante de $\mathcal{S}$. Si $\mathcal{C}(T) \neq \emptyset$, en notant pour tout $i \leq \ell(T)$, $t_i = (q_i, a_i, q_{i+1})$ :

- Considérons $c = x_1, .., x_{n+1} \in \mathcal{C}(T)$ telle que $strict(c)$ maximale. Notons $j$ et $k$ les indices de début et de fin, respectivement, de la chaine de $c$ qui atteint ce maximum.
  - $q_1 \in I$, donc $(q_0, start)$ est initial dans $\mathcal{A}$.
  - $q_{n+1} \in F$, donc $(q_{n+1}, stop)$ est final dans $\mathcal{A}$.
  - Pour tout $1 \leq i < j$, $t_i \in \mathcal{T}$, donc $((q_i, start), t_i, 0, (q_{i+1}, start))$ est une transitions de $\mathcal{A}$.

— $(q_j, a_j, q_{j+1}) \in \mathcal{T}$, et $x'_{j+1} \leq x_j$ ou $x'_{j+1} < x_j$ est dans $a_j$, donc $((q_j, start), t_j, 0, (q_{j+1}, x_{j+1}))$ ou $((q_j, start), t_j, 1, (q_{j+1}, x_{j+1}))$ est une transition de $\mathcal{A}$.

— Pour tout $j < i < k-1$ : $x'_{i+1} \leq x_i$ ou $x'_{i+1} < x_i$ est dans $a_i$, donc $((q_i, y), t_i, 0, (q_{i+1}, x))$ ou $((q_i, y), t_i, 1, (q_{i+1}, x))$ est une transition de $\mathcal{A}$.

— $(q_{k-1}, a_{k-1}, q_k) \in \mathcal{T}$, et $x'_k \leq x_{k-1}$ ou $x'_k < x_{k-1}$ est dans $a_{k-1}$, donc $((q_{k-1}, x_{k-1}), t_{k-1}, 0, (q_k, stop))$ ou $((q_{k-1}, xk-1), t_{k-1}, 1, (q_k, stop))$ est une transition de $\mathcal{A}$.

— Pourt tout $k \leq i < n$, $t_i \in \mathcal{T}$, donc $((q_i, stop), t_i, 0, (q_{i+1}, stop))$ est une transitions de $\mathcal{A}$.

Il existe donc un chemin acceptant $c'$ étiqueté par $w = t_1...t_n$ dans $\mathcal{A}$, et plus précisément dans $\mathcal{A}_1$, de poids poisitif. Le poids d'une transition étiquetée par $t_i$, où $1 \leq i \leq n$, dans ce chemin est égal à 1 si et seulement si $j \leq i < k$, et si l'inégalité correspondante dans la chaine $x_j, ..., x_k$ est stricte. Le poids de ce chemin est donc exactement $strict(c)$. On en déduit immédiatement :

$$p_{\mathcal{A}}(w) \geq poids(c') = strict(c) = \max_{c \in \mathcal{C}(T)}(strict(c))$$

— $w$ est associé à une trace, $T$. Si $\mathcal{C}(T) \neq \emptyset$, alors on peut construire, en reprenant la construction précédente, un chemin acceptant de poids positif étiqueté par $w$ dans $\mathcal{A}$, et plus précisément dans $\mathcal{A}_1$. Son poids dans $\mathcal{A}_2$ est donc de $-\infty$. Ainsi, un chemin acceptant de poids maximal dans $\mathcal{A}$ étiqueté par $w$ est un chemin acceptant de poids maximal dans $\mathcal{A}_1$ étiqueté par $w$, et est de la forme :

$$\begin{aligned} c = &(q_1, start) \xrightarrow{t_1:p_1} ...(q_{k_1-1}, start) \xrightarrow{t_{k_1-1}:p_{k_1-1}} \\ &(q_{k_1}, x_{k_1}) \xrightarrow{t_{k_1}:p_{k_1}} ...(q_{k_2-1}, x_{k_2-1}) \xrightarrow{t_{k_2-1}:p_{k_2-1}} \\ &(q_{k_2}, stop) \xrightarrow{t_{k_2}:p_{k_2}} ... \xrightarrow{t_n:p_n} (q_{n+1}, stop) \end{aligned}$$

où tous les $p_i$ sont positifs.

Ce chemin correspond à une chaine d'inégalités de $T$, car pour tout $k_1 - 1 \leq i \leq k_2$, $p_i \in \{0, 1\}$, donc pour tout $1 \leq i \leq n$, $x'_{i+1} \leq x_i$ ou $x'_{i+1} < x_i$ est dans $a_i$ par construction de $\mathcal{A}_1$ : $ch = x_j, ..x_k \in \mathcal{C}(T)$. De même que précedemment, le poids de $w$, qui est le poids de $c$ est alors égal à $strict(ch)$ :

$$p_{\mathcal{A}}(w) = poids(c) = strict(ch) \leq \max_{c \in \mathcal{C}(T)}(strict(c))$$

On a donc le résultat si une chaine d'inégalités existe. Sinon, cela signifie que toutes les transitions $t_1, ..., t_n$ sont vides. En effet, par l'absurde, supposons que $t_i = (q_i, a_i, q_{i+1})$ telle que $x' < y \in a_i$ ou $x' \leq y \in a_i$. Alors $y, x$ est une chaine. Ainsi, $\max_{c \in \mathcal{C}(T)} = -\infty$, et toutes les transitions sont vides. alors il n'existe aucune transition dans $\mathcal{A}_1$ de la forme :

— $((q_j, start), t_i, 0, (q_k, x_k))$

— $((q_j, x_j), t_i, 0, (q_k, stop))$

— $((q_j, start), t_i, 0, (q_k, stop))$

où $q_j, q_k$ états de $\mathcal{S}$, $x_j, x_k$ variables de $\mathcal{S}$, $1 \leq i \leq n$.

Donc les étatx initiaux ne sont pas co-accessibles et les états finaux ne sont accessibles en se restreignant aux transitions étiquetées par des éléments de $\{t_1, ...t_n\}$. Ainsi, le mot $w$ n'est pas accepté par $\mathcal{A}_1$. Il est donc accepté par $\mathcal{A}_2$, et :

$$p_{\mathcal{A}}(w) = p_{\mathcal{A}_2}(w) = |w|$$

□

**Lemme 11** (Lien mot-trace)**.** *Soient $\mathcal{S}$ un Size-Change Abstraction $(Q, I, F, \chi, \mathcal{T})$ et $\mathcal{A} = \mathcal{A}_1 \sqcup \mathcal{A}_2$ son automate max-plus associé. Si $w = t_1...t_n$ un mot sur $\mathcal{T}$, alors*

— *Soit il existe une trace acceptante $T = \tau_1 \xrightarrow{t_1} ... \xrightarrow{t_n} \tau_{n+1}$ de $\mathcal{S}$ bornée par $p_{\mathcal{A}}(w)$.*

— *Soit $p_{\mathcal{A}}(w) = |w|$.*

*Démonstration.* Soient $\mathcal{S}$ un Size-Change Abstraction $(Q, I, F, \chi, \mathcal{T})$ et $\mathcal{A} = \mathcal{A}_1 \sqcup \mathcal{A}_2$ son automate max-plus associé, et $w = t_1...t_n$ un mot sur $\mathcal{T}$. Opérons une disjonction de cas :

— Supposons que $w \in \mathcal{L}(\mathcal{A}_1)$, un chemin maximal de $w$ dans $\mathcal{A}$ est un chemin maximal de $w$ dans $\mathcal{A}_1$, et donc est de la forme :

$$\begin{aligned} c =& (q_1, start) \xrightarrow{t_1:p_1} ...(q_{k_1-1}, start) \xrightarrow{t_{k_1-1}:p_{k_1-1}} \\ & (q_{k_1}, x_{k_1}) \xrightarrow{t_{k_1}:p_{k_1}} ...(q_{k_2-1}, x_{k_2-1}) \xrightarrow{t_{k_2-1}:p_{k_2-1}} \\ & (q_{k_2}, stop) \xrightarrow{t_{k_2}:p_{k_2}} ... \xrightarrow{t_n:p_n} (q_{n+1}, stop) \end{aligned}$$

où, en notant $x_i = start$ si $i < k_1$, et $x_i = stop$ si $i \geq k_2$,

— $(q_1, x_1)$ initial dans $\mathcal{A}$, ce qui revient à dire que $q_1 \in I$.

— $(q_{n+1}, x_{n+1})$ final dans $\mathcal{A}$, ce qui revient à dire que $q_{n+1} \in F$.

— Pour tout $1 \leq i \leq n$, $((q_i, x_i), t_i, p_i, (q_{i+1}, x_{i+1}))$ est une transition de $\mathcal{A}_1$, donc $t_i$ est de la forme $(q_i, a_i, q_{i+1})$.

Ainsi, on construit un chemin $q_1 \xrightarrow{t_1} ... \xrightarrow{t_n} q_{n+1}$ dans $\mathcal{S}$, où $q_1 \in I$, et $q_{n+1} \in F$. Par le Lemme 4, il existe une trace acceptante de la forme :

$$T = \tau_1 \xrightarrow{t_1} ... \xrightarrow{t_n} \tau_{n+1}$$

De plus, $\mathcal{C}(T) \neq \emptyset$ car $w$ est accepté par $\mathcal{A}_1$. En effet, si $w$ est accepté par $\mathcal{A}_1$, au moins un état initial est co-accessible en ne considérant que les transitions étiquetés par des lettres de $w$. Donc il existe une transition du type $((q, start), t, p, (q', x))$ où $q, q' \in Q$, $t \in \{t_1, ..., t_n\}$, $x \in \chi \cup \{stop\}$, $p \geq 0$. Donc, il existe un $i$ tel que $1 \leq i < n$ et $x', x'' \in \chi$ tel que $x'' \leq x'$ ou $x'' < x'$ soit dans l'ensemble d'inégalités de la transition $t_i$, par construction des transitions de $\mathcal{A}_1$. Par le Lemme 8, il existe donc une trace acceptante $T' = \tau'_1 \xrightarrow{t_1} ... \xrightarrow{t_n} \tau'_{n+1}$ bornée par $\max_{c \in \mathcal{C}(T)}(strict(c))$. Cependant, d'après le Lemme 10, $\max_{c \in \mathcal{C}(T)}(strict(c)) = p_{\mathcal{A}}(w)$. On a donc le résultat.

— Supposons que $w \notin \mathcal{L}(\mathcal{A}_1)$, un chemin maximal de $w$ dans $\mathcal{A}$ est un chemin maximal de $w$ dans $\mathcal{A}_2$, et donc $p_{\mathcal{A}}(w) = p_{\mathcal{A}_2}(w) = |w|$

□

*Preuve de théorème 4.* Soient $\mathcal{S} = (Q, I, F, \chi, \mathcal{T})$ un Size-Change Abstraction et $\mathcal{A} = \mathcal{A}_1 \sqcup \mathcal{A}_2$ son automate max-plus associé. $\boxed{\Leftarrow}$ Supposons qu'il existe $N$ tel que pour tout $n$, il existe un mot $w$ tel que $|w| \geq n$ et $p_{\mathcal{A}}(w) \leq N$, ce qui revient à dire que $\mathcal{A}$ ne termine pas.

— Soit $w = t_1...t_m$ tel que $p_{\mathcal{A}} \leq N < m$. Alors il existe une trace acceptante du Size-Change Abstraction associé de longueur $m$ telle que les variables soient bornées par $p_{\mathcal{A}} \leq N$, par le Lemme 11.

— Il existe donc pour tout $n$ une trace acceptante de longueur supérieure à $n$ bornée par $N$. Prenons $n = N^{|\chi|}|Q|^2$. Il existe donc une trace acceptante de longueur $L$ strictement supérieure à $N^{|\chi|}|Q|^2$. Notons la $\tau_1 \xrightarrow{t_1} ... \xrightarrow{t_{L-1}} \tau_L$. Dans cette trace acceptante, il y a au moins une valuation $\tau_i$ qui apparaît plus de $|\mathcal{K}|^2$ fois. Il y a donc au moins un couple $(j, k)$ qui vérifie :

$$\begin{cases} j < k \\ \tau_i = \tau_k = \tau_j \\ si\ t_j = (q_j, a_j, q'_j)\ et\ t_{k-1} = (q_{k-1}, a_{k-1}, q'_{k-1})\ alors\ q_j = q'_{k-1} \end{cases}$$

— La trace acceptante infinie $\tau_1 \xrightarrow{t_1} ... \xrightarrow{t_{j-1}} \tau_j \xrightarrow{t_j} ... \xrightarrow{t_{k-1}} \tau_k \xrightarrow{t_j} ...$ qui boucle sur ce cycle est donc une trace acceptante de $\mathcal{S}$.

$\boxed{\Rightarrow}$ Supposons que $\mathcal{S}$ admette une trace acceptante infinie. Alors il existe une trace acceptante infinie $\tau_1 \xrightarrow{t_1} ...$ bornée, par la Proposition 5. Donc il existe une valuation $\tau_i$ qui apparaît une infinité de fois dans cette trace, et donc plus de $|Q|^2$ fois. On en déduit comme précédemment qu'il existe une valuation qui vérifie :

$$\begin{cases} j < k \\ \tau_i = \tau_k = \tau_j \\ t_j = (q_j, a_j, q'_j) \wedge t_{k-1} = (q_{k-1}, a_{k-1}, q'_{k-1}) \Rightarrow q_j = q'_{k-1} \end{cases}$$

Notons $t_k = (q_k, a_k, q'_k)$. Il existe un chemin partant de $q'_k$ et arrivant dans un état final de taille inférieure à $|\mathcal{T}|$ par co-accessibilité des cycles de $\mathcal{A}$ (Lemme 3) Notons les transitions empruntées par ce chemin : $t'_1, ..., t'_m$, ou $m \leq |\mathcal{T}|$ Posons pour tout $N \in \mathbb{N}$, $w_N = t_1...t_{j-1}(t_j...t_k)^N t'_1...t'_m$, noté $w_N = t''_1...t''_L$, où $L = (k-j+1)N+(j-1)+m$. Prouvons que les $w_N$ ont un poids borné par $\max(|\chi||Q|, k-j+1)+j-1+|\mathcal{T}|$. Considérons un chemin acceptant de poids maximum de $w_N$. Puisque $w_N$ est acceptant, d'après le Corollaire 3, et est reconnu par $\mathcal{A}_1$, d'après 4, ce chemin a la forme suivante :

$$\begin{aligned} c = &(q_1, start) \xrightarrow{t''_1:0} ...(q_{k_1-1}, start) \xrightarrow{t''_{k_1-1}:p_{k_1-1}} \\ &(q_{k_1}, x_{k_1}) \xrightarrow{t''_{k_1}:p_{k_1}} ...(q_{k_2-1}, x_{k_2-1}) \xrightarrow{t''_{k_2-1}:p_{k_2-1}} \\ &(q_{k_2}, stop) \xrightarrow{t''_{k_2}:0} ... \xrightarrow{t''_L:0} (q_{L+1}, stop) \end{aligned}$$

où tous les $p_i$ sont positifs. Le poids de ce chemin est égal à $\sum_{k_1-1 \leq i < k_2} p_i$. Sur cette plage d'indices, certains peuvent être associés aux transitions $t_1, ..., t_{j-1}$ et $t'_1, ...t'_m$. On majore la somme de tous ces poids par $j-1+|\mathcal{T}|$. On ne s'occupe à présent que des poids des transitions étiquetées par les transitions du cycle, les $t_j, ..., t_k$. Procédons à une disjonction de cas sur la valeur de ces poids :

- Soit tous ces poids sont nuls.
- Soit il existe au moins une transition de poids $> 1$. Prouvons qu'il ne peut pas y avoir plus de $\max(|\chi||Q|, k-j+1)$ transitions dans ce cas. Considérons le plus petit $i$ tel que $p_{i+k_1} = 1$.
  - Si après $k-j+1$ transitions, on est dans un état de la forme $(q, stop)$, ou si le chemin est assez court pour qu'il se finisse ou qu'il ait commencé à emprunter les transitions $t'_1, ...t'_m$ avant d'avoir pu parcourir $k-j+1$ transitions, alors, les transitions ne pouvant que peser 0 ou 1, le poids du mot $w_N$ est plus petit que $k-j+1$.
  - Sinon, après $k-j+1$ transitions, on arrive sur un état $(q_{i+k_1+k-j+1}, x_{i+k_1+k-j+1})$, où $q_{i+k_1+k-j+1} = q_{i+k_1}$, car un cycle est passé. Prouvons par l'absurde que $x_{i+k_1+k-j+1} \neq x_{i+k_1}$. La transition $t_{i+k_1}$ contient dans son ensemble d'inégalités $x'_{i+k_1+1} < x_{i+k_1}$. Puisque toutes les autres transitions ont un poids au moins 0, on a un chemin :

    $$(q_{k_1+i}, x_{k_1+i}) \xrightarrow{t'_{k_1+i}:1} (q_{i+k_1+1}, x_{i+k_1+1}) \xrightarrow{t'_{k_1+i+1}:\geq 0} ... \xrightarrow{t'_{k_1+i+k-j+1}:\geq 0} (q_{k_1+i}, x_{k_1+i})$$

    Donc $x_{k_1+i}, ..., x_{k_1+i+k-j+1}, x_{k_1+i}$ est une chaine de la trace infinie $T$ telle que $strict(c) > 0$. Puisque le cycle $t_j, ..., t_k$ se répète à l'infini dans la trace $T$, on peut même construire une chaine infinie de $T$ telle que $strict(c) = +\infty$ en répétant cette chaine à l'infini. C'est absurde car $T$ est bornée. Donc si une transition $t_{k_1+i}$ a un poids 1, le chemin ne peut réemprunter l'état $(q_{k_1+i}, x_{k_1+i})$. Or, il y a en tout $|\chi||Q|$ états tels que celui-là. Donc le chemin peut passer par au plus $|\chi||Q|$ transitions de poids 1 de ce type.

  Ainsi, il ne peut y avoir en tout plus de $\max(k-j+1, |\chi||Q|) + 2|\mathcal{T}|$ transitions de poids 1 dans ce chemin. Immédiatement, on en déduit que, pour tout $N \geq 0$ :

  $$p_{\mathcal{A}}(w_N) \leq \max(k-j+1, |\chi||Q|) + 2|\mathcal{T}|$$

  Ce qui revient à dire que l'on peut trouver des mots arbitrairement grands de poids inférieurs $\max(k-j+1, |\chi||Q|) + 2|\mathcal{T}|$. Ainsi, $\mathcal{A}$ ne termine pas.

□

# C Preuve du théorème 3

Tout d'abord, prouvons l'existence d'une famille calculant ces poids.

**Proposition 6** (Existence de $\mathcal{A}_{\ell,k}$)**.** *Pour tout $(k, \ell) \in (\mathbb{N}^* \times \mathbb{N})$, $k \geq \ell$, $p_{\ell,k}$ est calculable par un automate max-plus $\mathcal{A}_{\ell,k}$.*

*Démonstration.* Opérons une induction. L'hypothèse d'induction, pour tout $(k, \ell) \in (\mathbb{N}^* \times \mathbb{N})$, $k \geq \ell$, est :

$$\mathcal{H}_{\ell,k} : "p_{\ell,k} \text{ est calculable par un automate } max-plus"$$

— On a $\mathcal{H}_{0,k}$ pour tout $k > 0$, comme vu en Section 4.2

— On a $\mathcal{H}_{k,k}$ pour tout $k > 0$, comme vu en Section 4.2

— Si $\ell > 0$ et $k > \ell$, alors par induction, $\mathcal{H}_{\ell,k-1}$ et $\mathcal{H}_{\ell-1,k-1}$ sont vérifiées. Ainsi, $p_{\ell-1,k-1}$ et $p_{\ell,k-1}$ sont calculables par un automate max-plus. Par le Lemme 2, $\max_{1\leq i\leq p+1} p_{\ell,k-1}(w_i)$ et $\sum_{1\leq i\leq p+1} p_{\ell-1,k-1}(w_i)+p$ sont calculables par automate max-plus. Par le même Lemme, $p_{\ell,k}(w) = \max\left(\max_{1\leq i\leq p+1} p_{\ell,k-1}(w_i), \sum_{1\leq i\leq p+1} p_{\ell-1,k-1}(w_i) + p\right)$ est calculable par automate max-plus.

□

Ensuite, majorons la complexité de la famille d'automates.

**Proposition 7** (Majoration de la complexité des automates $\mathcal{A}_{1,k}$)**.** *Les automates $\mathcal{A}_{1,k}$ ont une complexité en $k$ au plus.*

*Démonstration.* On va raisonner par récurrence sur $k \geq 1$. L'hypothèse de récurrence est :

$$\mathcal{H}_k : \text{"}il\ existe\ \alpha_k\ tel\ que\ pour\ tout\ mot\ w \in \Sigma_k,\ \alpha_k p_{1,k}(w)^k \geq |w|\text{"}$$

— Si $k = 1$ : pour tout mot $w \in \Sigma_1$, $p_{1,k}(w)^k = p_{1,1}(w) = |w|$, et donc $p_{1,k}(w)^k \geq |w|$. $\mathcal{H}_1$ est vérifiée.

— Supposons que un $k > 1$ donné, $\mathcal{H}_{k-1}$ soit vérifiée. Soit $w = w_1 a_k ... a_k w_{p+1} \in \Sigma_k$ où $(w_i)_{1\leq i\leq p+1} \in (\Sigma_{k-1}^*)^{p+1}$. Posons $N = p_{1,k}(w)$. Avec ces notations, par définition de $p_{1,k}$ et de $p_{0,k-1}$ :

$$N = \max\left(\max_{1\leq i\leq p+1} p_{1,k-1}(w_i), \sum_{1\leq i\leq p+1} (p_{0,k-1}(w_i)) + p\right) = \max\left(\max_{1\leq i\leq p+1} p_{1,k-1}(w_i), p\right)$$

Immédiatement :

$$\begin{cases} p \leq N & (1) \\ \forall\ 1 \leq i \leq p+1,\ p_{1,k-1}(w_i) \leq N & (2) \end{cases}$$

De plus, $\mathcal{H}_{k-1}$ affirme que :

$$Il\ existe\ \alpha_{k-1}\ tel\ que\ pour\ tout\ mot\ w \in \Sigma_{k-1},\ \alpha_{k-1} p_{1,k-1}(w)^{k-1} \geq |w| \quad (3)$$

Ainsi :

$$\begin{aligned} |w| &\leq \sum_{i=1}^{p+1} |w_i| + p \\ &\leq \alpha_{1,k-1} \sum_{i=1}^{p+1} p_{1,k-1}(w_i)^{k-1} + p \\ &\leq (p+1)\alpha_{k-1} p_{1,k-1}(w_i)'^{k-1} + N \\ &\leq (N+1)\alpha_{k-1} N^{k-1} + N \\ &\leq (N+1)\alpha_{k-1} N^{k-1} + N \\ &\leq (2\alpha_{1,k-1} + 1) N^k \end{aligned}$$

En posant $\alpha_k = (2\alpha_{1,k-1} + 1)$, on a que : $\alpha_k p_{1,k}(w)^k \geq |w|$, soit exactement $\mathcal{H}_k$.

On a ainsi prouvé que l'ordre de la complexité de $\mathcal{A}_{1,k}$ est au plus en $k$ pour tout $k \geq 1$. □

**Proposition 8** (Majoration de la complexité des automates $\mathcal{A}_{\ell,k}$, $\ell > 0$)**.** *Les automates $\mathcal{A}_{\ell,k}$ ont une complexité en $\frac{k}{\ell}$ au plus.*

*Démonstration.* On va raisonner par induction pour prouver ce résultat. Plus précisément, on va prouver que si $\mathcal{A}_{\ell,k-1}$ et $\mathcal{A}_{\ell-1,k-1}$ se comportent comme prévu, alors $\mathcal{A}_{\ell,k}$ a la complexité recherchée. L'hypothèse d'induction est, pour $(\ell, k) \in (\mathbb{N}^*)^2$ :

$$\mathcal{H}_{(\ell,k)} : \text{"}Il\ existe\ \alpha_{\ell,k}\ tel\ que\ pour\ tout\ w \in \Sigma_k,\ \alpha_{\ell,k} p_{\ell,k}(w)^{\frac{k}{\ell}} \geq |w|\text{)"}$$

— Tout d'abord, considérons les cas de base nécessaires à ce raisonnement. La Figure 11 illustre la situation.
  — Pour la famille $(\mathcal{A}_{1,k})_{k\geq 1}$, l'hypothèse d'induction est la propriété prouvée par la Proposition 7.
  — Pour la famille $(\mathcal{A}_{k,k})_{k\geq 1}$, pour tout $k \geq 1$, $w \in \Sigma_k$, $p_{k,k}(w)^k = |w|^k$, et donc $p_{k,k}(w)^k \geq |w|$. $\mathcal{H}_{k,k}$ est vérifiée.
— Procédons maintenant à l'induction. Soit $w = w_1 a_{k+1} ... a_{k+1} w_{p+1}$ où pour tout $1 \leq i \leq p+1$, $w_i \in \Sigma_{k-1}^*$ et $p_{\ell,k}(w) = N$. On va poser les notations suivantes :

$$\textit{Pour tout } 1 \leq i \leq p+1 \begin{cases} p_i = p_{\ell-1,k-1}(w_i) \\ p'_i = p_{\ell,k-1}(w_i) \end{cases}$$

Avec ces notations, on peut écrire :

$$N = \max\left( \max_{1\leq i\leq p+1} p'_i, \sum_{1\leq i\leq p+1} p_i + p \right)$$

Immédiatement :

$$\begin{cases} \sum_{1\leq i\leq p+1} p_i + p \leq N & (1) \\ \textit{Pour tout } 1 \leq i \leq p+1,\ p'_i \leq N & (2) \end{cases}$$

De plus, les hypothèses d'induction affirment que :

$$\textit{Il existe } \alpha_{\ell,k-1} \textit{ et } \alpha_{\ell-1,k-1} \textit{ tels que pour tout } 1 \leq i \leq p+1, \begin{cases} \alpha_{\ell-1,k-1} p_i^{\frac{k-1}{\ell-1}} \geq |w_i| & (3) \\ \alpha_{\ell,k-1} {p'_i}^{\frac{k-1}{\ell}} \geq |w_i| & (4) \end{cases}$$

On déduit de (2), (3) et (4), et du fait que $1 \leq \ell < k$ et que toutes les longueurs sont supérieures à 1 :

$$\textit{Pour tout } 1 \leq i \leq p+1,\ |w_i| \leq \alpha_{\ell,k-1} N^{\frac{k-1}{\ell}} \ \wedge\ |w_i| \leq \alpha_{\ell-1,k-1} p_i^{\frac{k-1}{\ell-1}}$$

Ainsi,

$$\begin{aligned} |w| &\leq \sum_{i=1}^{p+1} |w_i| + p \\ &\leq \max(\alpha_{\ell,k-1}, \alpha_{\ell-1,k-1}) \sum_{i=1}^{p+1} \min\left( N^{\frac{k-1}{\ell}}, p_i^{\frac{k-1}{\ell-1}} \right) + p \end{aligned}$$

Faisons une disjonction de cas sur la valeur de ce minimum.

1. Notons $I_{\geq} = \left\{ 1 \leq i \leq p+1 \mid p_i^{\frac{k-1}{\ell-1}} \geq N^{\frac{k-1}{\ell}} \right\}$. Quel est son cardinal ?

$$\forall i \in I_{\geq},\ p_i \geq N^{\frac{\ell-1}{\ell}}$$

On en déduit que :

$$\begin{aligned} |I_{\geq}| N^{\frac{\ell-1}{\ell}} &\leq \sum_{i\in I_{\geq}} p_i \\ &\leq \sum_{i=1}^{p+1} p_i + p \\ &\leq N \\ \Rightarrow |I_{\geq}| &\leq N^{\frac{1}{\ell}} \end{aligned}$$

En sommant sur $I_{\geq}$, on obtient :

$$\begin{aligned}\sum_{i\in I_{\geq}} N^{\frac{k-1}{\ell}} &\leq |I_{\geq}| N^{\frac{k-1}{\ell}} \\ &\leq N^{\frac{1}{\ell}} N^{\frac{k-1}{\ell}} \\ &= N^{\frac{k}{\ell}}\end{aligned}$$

2. Maintenant, considérons $I_{<}$, le complémentaire de $I_{\geq}$ dans l'ensemble des indices. On considère une partition $(I_1, ..., I_q)$ de cet ensemble telle que :

$$\begin{cases} \forall j < q, \frac{1}{2} N^{\frac{l-1}{l}} \leq \sum_{i\in I_j} p_i \leq N^{\frac{\ell-1}{\ell}} \\ \sum_{i\in I_q} p_i < \frac{1}{2} N^{\frac{\ell-1}{\ell}} \end{cases}$$

**Remarque 1.** *$I_q$ peut être vide.*

Occupons-nous d'abord de $I_q$.

$$\begin{aligned}\sum_{i\in I_q} p_i &< \frac{1}{2} N^{\frac{\ell-1}{\ell}} \\ \Rightarrow \sum_{i\in I_q} p_i^{\frac{k-1}{\ell-1}} &< \left(\frac{1}{2} N^{\frac{\ell-1}{\ell}}\right)^{\frac{k-1}{\ell-1}} \\ &< N^{\frac{k}{\ell}}\end{aligned}$$

Qu'en est-il des autres paquets ? Comme précédemment, on va majorer le nombre de paquets.

$$\begin{aligned}\frac{1}{2} N^{\frac{\ell-1}{\ell}} (q-1) &\leq (q-1) \sum_{i\in I_j} p_i \\ &\leq \sum_{j<q} \sum_{i\in I_j} p_i \\ &\leq \sum_{i\leq p+1} p_i + p \leq N \\ \Rightarrow q - 1 &\leq 2N N^{-\frac{\ell-1}{\ell}} \\ &= 2N^{\frac{1}{\ell}}\end{aligned}$$

De plus, si $j < q$ :

$$\begin{aligned}\sum_{i\in I_j} p_i &\leq N^{\frac{\ell-1}{\ell}} \\ \Rightarrow \sum_{i\in I_j} p_i^{\frac{k-1}{\ell-1}} &\leq \left(N^{\frac{\ell-1}{\ell}}\right)^{\frac{k-1}{\ell-1}} \\ &= N^{\frac{k-1}{\ell}}\end{aligned}$$

Soit :

$$\begin{aligned}\sum_{j<q} \sum_{i\in I_j} p_i^{\frac{k-1}{\ell-1}} &\leq (q-1) N^{\frac{k-1}{\ell}} \\ &\leq 2N^{\frac{1}{\ell}} N^{\frac{k-1}{\ell}} \\ &\leq 2N^{\frac{k}{\ell}}\end{aligned}$$

En réunissant tous ces résultats on obtient enfin :

$$\begin{aligned}
|w| &\leq \sum_{i=1}^{p+1} \min\left(N^{\frac{k-1}{\ell}}, p_i^{\frac{k-1}{\ell-1}}\right) + p \\
&\leq \max(\alpha_{\ell,k-1}, \alpha_{\ell-1,k-1}) \left(\sum_{i\in I_{\geq}} N^{\frac{k-1}{\ell}} + \sum_{i\in I_{<}} p_i^{\frac{k-1}{\ell-1}}\right) + p \\
&= \max(\alpha_{\ell,k-1}, \alpha_{\ell-1,k-1}) \left(\sum_{i\in I_{\geq}} N^{\frac{k-1}{\ell}} + \sum_{i\in I_q} p_i^{\frac{k-1}{\ell-1}} + \sum_{j<q}\sum_{i\in I_j} p_i^{\frac{k-1}{\ell-1}}\right) + p \\
&< 4\max(\alpha_{\ell,k-1}, \alpha_{\ell-1,k-1}) N^{\frac{k}{\ell}} + p \\
&\leq (4\max(\alpha_{\ell,k-1}, \alpha_{\ell-1,k-1}) + 1) N^{\frac{k}{\ell}}
\end{aligned}$$

On a donc bien prouvé :

$$\exists\ \alpha_{\ell,k},\ \forall\ w \in \Sigma_{k+1},\ \alpha_{\ell,k} p_{\ell,k}(w)^{\frac{k}{\ell}} \geq |w|$$

On a ainsi prouvé que la complexité est au plus en $\frac{k}{\ell}$.

□

On souhaite maintenant exhiber une famille témoin qui permettrait de minorer la complexité de cette famille d'automates. On définit donc la famille $(w_{k,n})_{(k,n)\in\mathbb{N}^2}$ où :

$$\textit{Pour tout } n \in \mathbb{N},\ \begin{cases} w_{0,n} = \varepsilon \\ \textit{pour tout } i > 0,\ w_{i+1,n} = (w_{i,n} a_{i+1})^n w_{i,n} \in \Sigma_{i+1}^* \end{cases}$$

**Lemme 12** (Taille des mots $w_{k,n}$)**.** $\forall\ k > 0,\ |w_{k,n}| \sim_{n\to\infty} n^k$

**Proposition 9** (Poids des $w_{k,n}$ dans les automates $\mathcal{A}_{1,k}$)**.** *Si* $k > 0$, $p_{1,k}(w_{k,n}) \sim_{n\to\infty} n$

*Démonstration.* Procédons par récurrence sur $k \geq 1$. L'hypothèse de récurrence $\mathcal{H}_k$, $k \geq$ est :

$$\mathcal{H}_k :\ \text{"}p_{1,k}(w_{k,n}) \sim_{n\to\infty} n\text{"}$$

— Si $k = 1$, pour tout $w \in \Sigma_1$, $p_{1,1}(w) = |w|$. Donc, pour tout $n \geq 0$, $p_{1,1}(w_{1,n}) = |w_{1,n}|$. D'après le Lemme 12, $p_{1,1}(w_{1,n}) \sim_{n\to\infty} n$.

— Supposons que pour un $k$, $\mathcal{H}_{k-1}$ soit vérifiée. Alors $p_{1,k-1}(w_{k-1,n}) \sim_{n\to\infty} n$ Par définition de $p_{1,k}$ :

$$\begin{aligned}
p_{1,k}(w_{k,n}) &= \max\left(p_{1,k-1}(w_{k-1,n}), n\right) \\
&\sim_{n\to\infty} \max(n, n) \\
&\sim_{n\to\infty} n
\end{aligned}$$

Par récurrence, on a le résultat attendu. □

**Proposition 10** (Poids des $w_{k,n}$ dans les automates $\mathcal{A}_{\ell,k}$, $\ell > 0$)**.** *Si* $k, \ell > 0$, $p_{\ell,k}(w_{k,n}) \sim_{n\to\infty} n^\ell$

*Démonstration.* Procédons par induction sur le couple $(\ell, k)$. On prend comme hypothèse d'induction

$$\mathcal{H}_{\ell,k} :\ \text{"}p_{\ell,k}(w_{k,n}) \sim_{n\to\infty} n^\ell\text{"}$$

— Pour la famille $(\mathcal{A}_{1,k})$, c'est ce qui est prouvé dans la Proposition 9.

— Pour la famille $(\mathcal{A}_{k,k})$ :

$$\textit{Pour tout } k,\ \textit{pour tout } n,\ p_{k,k}(w_{k,n}) = |w_{k,n}|$$

Donc, pour tout $k$, $p_{k,k}(w_{k,n}) \sim_{n\to\infty} n^k$ par le Lemme 12.

— Supposons l'hypothèse d'induction vraie pour les couples $(\ell, k-1)$ et $(\ell-1, k-1)$. On a donc :

$$\begin{cases} p_{\ell,k-1}(w_{k-1,n}) \sim_{n\to\infty} n^{\ell} \\ p_{\ell-1,k-1}(w_{k-1,n}) \sim_{n\to\infty} n^{\ell-1} \end{cases}$$

Or,

$$\begin{aligned} p_{\ell,k}(w_{k,n}) &= \max\left(p_{\ell,k-1}(w_{k-1,n}), (n+1)p_{\ell-1,k-1}(w_{k-1,n}) + n\right) \\ &\sim_{n\to\infty} \max(n^{\ell}, (n+1)n^{\ell-1} + n) \\ &\sim_{n\to\infty} \max(n^{\ell}, n^{\ell}) \\ &\sim_{n\to\infty} n^{\ell} \end{aligned}$$

On vient de prouver $\mathcal{H}_{\ell,k}$. □

On a donc une famille témoin telle le ratio longueur/poids soit équivalent à $\frac{k}{\ell}$ lorsque la longueur tend vers $+\infty$. La complexité de l'automate $\mathcal{A}_{\ell,k}$ est donc exactement en $\frac{k}{\ell}$.

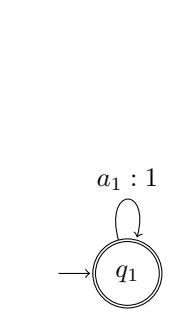
# D Recherche de programmes associés et observations

## D.1 Famille de complexité $k$

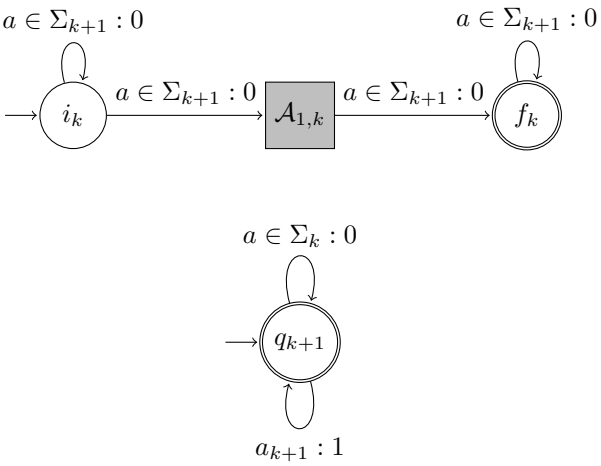
**Construction inductive des automates.** On utilise la construction du théorème 3 pour exhiber une construction récursive possible de la famille d'automates $(\mathcal{A}_{1,k})_{k\in\mathbb{N}^*}$ de complexité $k$ :

— $\mathcal{A}_{1,1}$ représenté en Figure 15a.

— $\mathcal{A}_{1,k+1}$ est défini sur $\Sigma_{k+1}$ à partir de $\mathcal{A}_{1,k}$, tel que représenté en Figure 15b. On a utilisé les constructions présentées dans la preuve de Lemme 2.




(a) Automate $\mathcal{A}_{1,1}$




(b) Automate $\mathcal{A}_{1,k+1}$

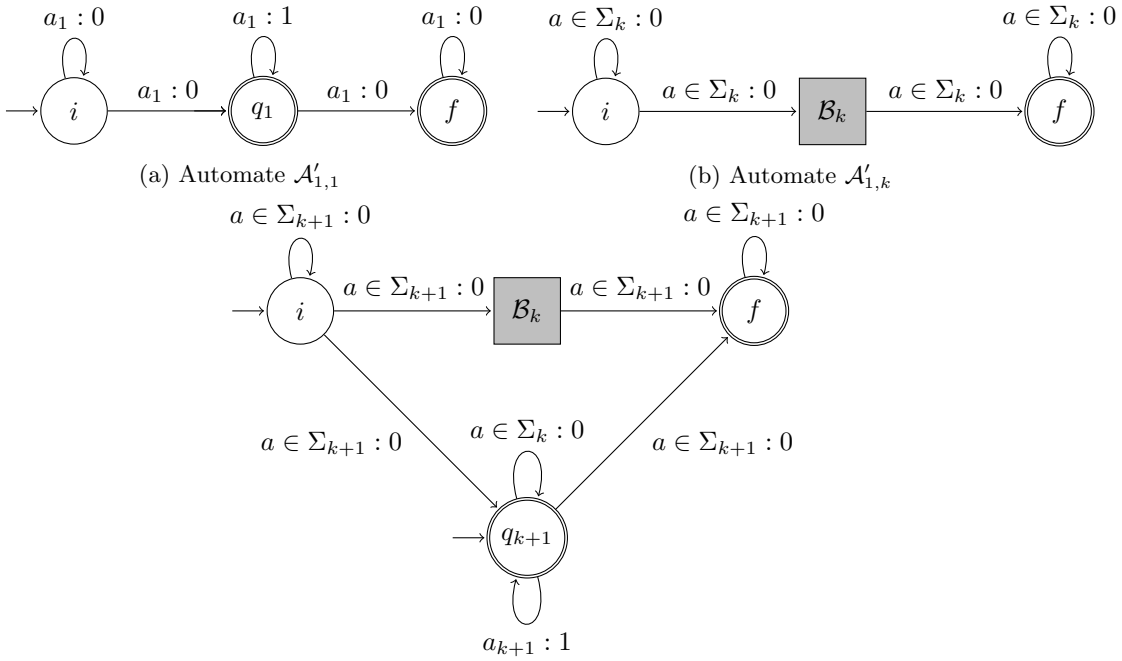
FIGURE 15 – Famille $(\mathcal{A}_{1,k})_{k\in\mathbb{N}^*}$

On va modifier cette construction pour obtenir une famille équivalente en complexité, $(\mathcal{A}'_{1,k})_{k\in\mathbb{N}^*}$ , qui peut être associée à une famille d'Size-Change Abstractions . La Figure 16 représente la construction récursive de cette famille :

— La Figure 16a représente le premier élément de la famille

— La Figure 16b représente la forme du $k^{eme}$ élément de la famille.

— La Figure 16c représente la construction du $k+1^{eme}$ élément de la famille en fonction du $k^{eme}$.




(a) Automate $\mathcal{A}'_{1,1}$

(b) Automate $\mathcal{A}'_{1,k}$

(c) Automate $\mathcal{A}_{1,k+1}$

FIGURE 16 – Famille $(\mathcal{A}_{1,k})_{k\in\mathbb{N}^*}$

**Size-Change Abstraction associés.** Quels sont les Size-Change Abstractions associés à ces automates ?

— Pour $\mathcal{A}'_{1,1}$, le Size-Change Abstraction à un état associé est :

$$a_1 = \{x'_1 < x_1\}$$

— Pour $\mathcal{A}'_{1,2}$, le Size-Change Abstraction à un état associé est :

$$\begin{cases} a_1 = \{x'_1 < x_1,\ x'_2 \leq x_2\} \\ a_2 = \{x'_2 < x_2\} \end{cases}$$

— Pour $\mathcal{A}'_{1,k+1}$, $n \in \mathbb{N}$ ? On observe qu'on a rajouté une transition $a_{k+1}$ et une variable $x_{k+1}$, telles que, si $\mathcal{T}_k$, avec pour ensemble de transitions $\Sigma_k$, est le Size-Change Abstraction de $\mathcal{A}_{1,k}$, celui $\mathcal{T}_{k+1}$ de $\mathcal{A}_{1,k+1}$ est :

$$\begin{cases} \forall i \leq k,\ a_k^{\mathcal{T}_{k+1}} = a_k^{\mathcal{T}_k} \cup \{x'_{k+1} \leq x_{k+1}\} \\ a_{k+1}^{\mathcal{T}_{k+1}} = \{x'_{k+1} < x_{k+1}\} \end{cases}$$

Ainsi, on peut également définir les programmes par induction.

**Programmes associés.** On définit les programmes par récurrence.

— Pour $\mathcal{A}_{1,2}$, un programme associé est facile à trouver, par exemple :

```
int 1_2 (int x1,int x2,int m) {
    x1=x2=m;
    while (x1>=0 && x2>=0) {
        if (random()) {
            x1=x1-1;
        }
        else {
```

```
                x1=m;
                x2=x2-1;
            };
        };
    return 0;
}
```

— Pour $\mathcal{A}_{1,k+1}$, $k \in \mathbb{N}$? On observe que si le programme pour $\mathcal{A}_{1,k}$ est de la forme :

```
int 1_k (int x1,int x2,...,int xk, int m) {
    x1=x2=...=xk=m;
    while (x1>=0 && x2>=0 && ... && xk>=0) {
        if (random()) {
            bloc_1
        }
        ...
        else {
            bloc_k
        };
    };
    return 0;
}
```

Alors on peut le modifier pour en faire un programme associé à $\mathcal{A}_{1,k+1}$. Le programme s'écrit alors :

```
int 1_(k+1) (int x1,int x2,...,int xk, int x(k+1), int m) {
    x1=x2=...=xk=x(k+1)=m;
    while (x1>=0 && x2>=0 && ... && xk>=0 && x(k+1)>=0) {
        if (random()) {
            bloc_1
        }
        ...
        else if (random()) {
            bloc_k
        }
        else {
            x1=m;
            ...
            xk=m;
            x(k+1)=x(k+1)-1;
        };
    };
    return 0;
}
```

On a écrit un programme Ocaml permettant de générer cette famille de programmes automatiquement afin de les tester avec l'algorithme utilisant les Integer-Interpreted Automata . Il prend en entier $n > 2$, et produit, moyennant la présence d'un fichier de la bonne forme pour $n-1$, le programme de la bonne forme pour $n$. L'initialisation se fait à 2.

```
1 open Sys
2
3 let main n=
4   if n>2 then
5     let fichier=open_in ("1_"^(string_of_int (n-1))^".c")  in
6     let s=ref "" and b=ref true in
7     let nouveau=open_out ("1_"^(string_of_int n)^".c") in
8     begin
```

```
9
10 (*Declarations de variables a modifier*)
11
12       s:=input_line fichier;
13       let l=String.length (!s) in
14       Printf.fprintf nouveau "%s, int x%n) {\n" (String.sub (!s) 0 (l-3)) n;
15
16 (*Initialisations a modifier*)
17
18       s:=input_line fichier;
19       let l=String.length (!s) in
20       Printf.fprintf nouveau "%sx%n=m;\n" (String.sub (!s) 0 (l-2)) n;
21
22 (*On change les conditions du while*)
23
24       s:=input_line fichier;
25       let l=String.length (!s) in
26       Printf.fprintf nouveau "%s && x%n>=0) {\n" (String.sub (!s) 0 (l-3)) n;
27
28 (*On passe toutes les boucles (on connait la longueur de chacune en fonction de n)*)
29
30       for i=1 to (2*(n-2)+(((n-2)*(n-1))/2)) do
31         s:=input_line fichier;
32         Printf.fprintf nouveau "%s\n" (!s);
33       done;
34
35 (*Sauf la derniere ou il faut remplacer le else par un else if (random())*)
36
37       s:=input_line fichier;
38       let l=String.length (!s) in
39       Printf.fprintf nouveau "%sif (random()) {\n" (String.sub (!s) 0 (l-1));
40       for i=1 to (n-1) do
41         s:=input_line fichier;
42         Printf.fprintf nouveau "%s\n" (!s);
43       done;
44       s:=input_line fichier;
45       Printf.fprintf nouveau "\t\t}\n";
46
47 (*Enfin on rajoute la derniere boucle*)
48
49        Printf.fprintf nouveau "\t\telse {\n";
50        for i=1 to (n-1) do
51          Printf.fprintf nouveau "\t\t\tx%n=m;\n" i;
52        done;
53        Printf.fprintf nouveau "\t\t\tx%n=x%n-1;\n\t\t};\n" n n;
54
55 (*Et on termine le fichier*)
56
57       while !b do
58         try
59           s:=input_line fichier;
60           Printf.fprintf nouveau "%s\n" (!s);
61         with
62           End_of_file ->
63             begin
64               close_in fichier;
```

```
                close_out nouveau;
                b:=false;
              end;
      done;
    end

let _=try
        let a=(argv.(1)) in main (int_of_string a);
  with Failure("int_of_string")-> Printf.printf "Entrez un entier\n"
```

La complexité donnée par cette technique est bien de $k$ pour l'automate $\mathcal{A}'_{1,k}$, lorsque $k \leq 4$. Au-delà, l'outil produit l'erreur `Too Much Variables`.

## D.2 Cas de l'automate max-plus $\mathcal{A}_{3,2}$

Cette section présente la recherche d'un programme se traduisant en un automate max-plus de complexité $\frac{3}{2}$, représenté à la Figure 9 par exemple. On va construire un automate max-plus de même complexité avec le Théorème 3 et le Lemme 2.

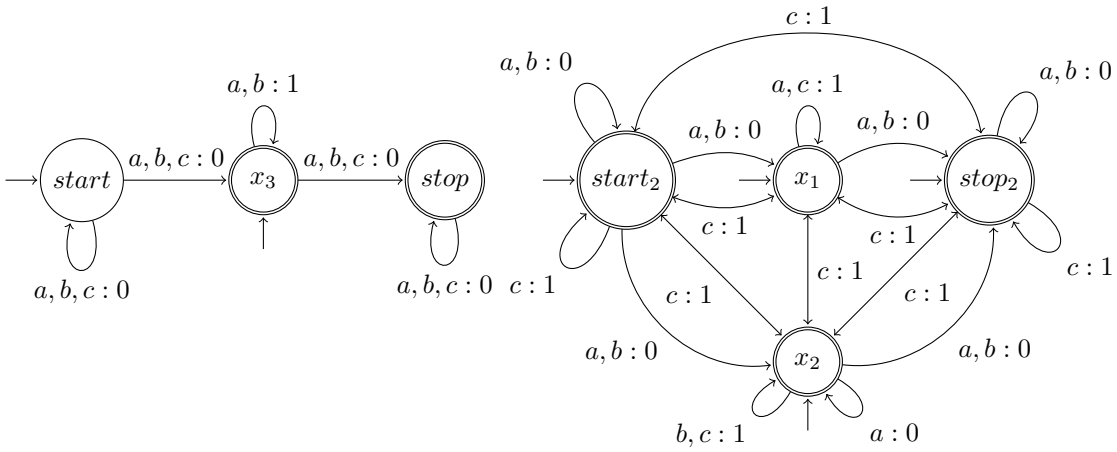



FIGURE 17 – $\mathcal{A}_{3,2}$

**Size-Change Abstraction associé.** On souhaiterai lui trouver un automate équivalent de même complexité qui puisse être l'abstraction d'un Size-Change Abstraction. On rajoute pour cela à l'automate de la Figure 17 des transitions de telle sorte que les transitions partant de *start*, respectivement de tout état, et arrivant à tout état, repectivement *stop*, étiquetées par $a, b, c : 0$ existent. On a alors le Size-Change Abstraction correspondant suivant :

$$\begin{cases} a = & \{x'_1 < x_1,\ x'_2 \leq x_2, x'_3 < x_3, x'_1 \leq start_2, stop'_2 \leq x_1, x'_2 \leq start_2, start'_2 \leq start_2, stop'_2 \leq stop_2, stop'_2 \leq x_2\} \\ b = & \{x'_2 < x_2, x'_3 < x_3, x'_1 \leq start_2, stop'_2 \leq x_1, x'_2 \leq start_2, start'_2 \leq start_2, stop'_2 \leq stop_2, stop'_2 \leq x_2\} \\ c = & \{x'_2 < x_2, start'_2 < start_2, stop'_2 < stop_2, x'_1 < x_1, x'_2 < x_1, x'_2 < stop_2, x'_2 < start_2, x'_1 < x_2, \\ & x'_1 < start_2, x'_1 < stop_2, start'_2 < x_1, start'_2 < x_2, start'_2 < stop_2, stop'_2 < x_1, stop'_2 < x_2, stop'_2 < start_2, \} \end{cases}$$

**Programme associé.** Un programme qui peut se traduire en ce Size-Change Abstraction est :

```
int main (int x1, int x2, int x3, int start2, int stop2, int m) {
        x1=x2=x3=start2=stop2=m;
        while (x1>=0 && x2>=0 && x3>=0 && start2>=0 && stop2>=0) {
                if (random()) {
                        if (x1<=x2 && stop2>x1) {
                                stop2=x1;
                        }
```

```
                else if (stop2>x2) {
                        stop2=x2;
                };
                x1=x1-1;
                x3=x3-1;
                if (x1>start2) {
                        x1=start2;
                };
                if (x2>start2) {
                        x2=start2;
                };
        }
        else if (random()) {
                if (x1<=x2 && stop2>x1) {
                        stop2=x1;
                }
                else if (stop2>x2) {
                        stop2=x2;
                };
                x1=m;
                x2=x2-1;
                x3=x3-1;
                if (x1>start2) {
                        x1=start2;
                };
                if (x2>start2) {
                        x2=start2;
                };
        }
         else {
                x3=m;
                if (x1<=x2 && x1<=start2 && x1<=stop2) {
                        start2=x1-1;
                        stop2=x1-1;
                        x2=x1-1;
                        x1=x1-1;
                }
                else if (start2<=x2 && start2<=x1 && start2<=stop2) {
                        x1=start2-1;
                        stop2=start2-1;
                        x2=start2-1;
                        start2=start2-1;
                }
                else if (stop2<=x2 && stop2<=x1 && stop2<=start2) {
                        x1=stop2-1;
                        start2=stop2-1;
                        x2=stop2-1;
                        stop2=stop2-1;
                }
                else {
                        start2=x2-1;
                        stop2=x2-1;
                        x1=x2-1;
                        x2=x2-1;
                };
        };
```

```
        };
        return 0;
}
```

Cependant, ce programme met en péril l'outil algorithmique utilisé : il produit l'erreur `Too much variables`. On a donc essayé de modifier les fichiers utilisés par cet outil, en simplifiant les fichiers intermédiaires. Pour cela, on a tout d'abord simplifier le pré-traitement du fichier, qui génère des inégalités sur les variables, en élargissant ces inégalités à $0 \leq x \leq m$ pour tout état de l'Integer-Interpreted Automaton et toute variable du programme. L'erreur était encore présente. On a donc généré à la main un Integer-Interpreted Automaton plus simple, puis laisser le pré-traitement s'opérer. Celui-ci étant encore trop précis pour effectuer les calculs, on a élargit les invariants comme précedemment, pour obtenir encore une fois la même erreur. Voici le fichier généré à la main pour cette étude :

```
//Fast File Modified for ranking stuff - 2016, Jul 12,14:31:11

model main{

//parameters m__o, start2__o, stop2__o, x1__o, x2__o, x3__o;

  var m, start2, stop2, x1, x2, x3, m__o, start2__o, stop2__o, x1__o, x2__o, x3__o;

  states stop, start, aff, ____start;

  transition t_1 :={
   from     := start ;
   to       := stop ;
   guard    := m<=0;
   action   := start2' = m, stop2' = m, x1' = m, x2' = m, x3' = m;
  };


  transition t_2 :={
   from     := start ;
   to       := aff ;
   guard    := m>=1;
   action   := start2' = m, stop2' = m, x1' = m, x2' = m, x3' = m;
  };


  transition t_3 :={
   from     := aff ;
   to       := aff ;
   guard    := m>=0 && start2>=x2 && stop2>=x1 && x1>=1 && x3>=1 && x1<=x2;
   action   := x1' = x1 - 1, stop2' = x1, x3' = x3 - 1;
  };


  transition t_4 :={
   from     := aff ;
   to       := aff ;
   guard    := m>=0 && start2>=x2 && stop2>=1 && x3>=1 && x1<=x2 && stop2+1<=x1;
   action   := x1' = x1 - 1, x3' = x3 - 1;
  };


  transition t_5 :={
```

```
 from     := aff ;
 to       := aff ;
 guard    := m>=0 && start2>=x1 && stop2>=x2+1 && x1>=x2+1 && x2>=1 && x3>=1;
 action   := x1' = x1 - 1,x3' = x3 - 1,stop2' = x2;
};


transition t_6 :={
 from     := aff ;
 to       := aff ;
 guard    := m>=0 && start2>=x1 && stop2>=1 && x1>=x2+1 && x3>=1 && stop2<=x2;
 action   := x1' = x1 - 1,x3' = x3 - 1;
};


transition t_7 :={
 from     := aff ;
 to       := aff ;
 guard    := m>=0 && start2>=1 && stop2>=1 && x3>=1 && x1<=x2 && stop2+1<=x1 &&
             start2+1<=x1;
 action   := x1' = start2,x2' = start2,x3' = x3 - 1;
};


transition t_8 :={
 from     := aff ;
 to       := aff ;
 guard    := m>=0 && start2>=1 && stop2>=x1 && x3>=1 && x1<=x2 && start2+1<=x1;
 action   := x1' = start2,x2' = start2,x3' = x3 - 1,stop2' = x1;
};


transition t_9 :={
 from     := aff ;
 to       := aff ;
 guard    := m>=0 && start2>=1 && stop2>=1 && x1>=x2+1 && x3>=1 && stop2<=x2 &&
             start2+1<=x2;
 action   := x1' = start2,x2' = start2,x3' = x3 - 1;
};


transition t_10 :={
 from     := aff ;
 to       := aff ;
 guard    := m>=0 && start2>=1 && stop2>=x2+1 && x1>=x2+1 && x3>=1 && start2+1<=x2;
 action   := x1' = start2,x2' = start2,x3' = x3 - 1,stop2' = x2;
};


transition t_11 :={
 from     := aff ;
 to       := aff ;
 guard    := m>=0 && start2>=x1 && stop2>=1 && x3>=1 && stop2+1<=x1 && start2<=x2;
 action   := x1' = x1 - 1,x3' = x3 - 1,x2' = start2;
};
```

```
transition t_12 :={
 from    := aff ;
 to      := aff ;
 guard   := m>=0 && start2>=x1 && stop2>=x1 && x1>=1 && x3>=1 && start2<=x2;
 action  := x1' = x1 - 1,x3' = x3 - 1,x2' = start2,stop2' = x1;
};


transition t_13 :={
 from    := aff ;
 to      := aff ;
 guard   := m>=0 && start2>=x2 && stop2>=1 && x3>=1 && stop2<=x2 && start2+1<=x1;
 action  := x3' = x3 - 1,x1' = start2;
};


transition t_14 :={
 from    := aff ;
 to      := aff ;
 guard   := m>=0 && start2>=x2 && stop2>=x2+1 && x2>=1 && x3>=1 && start2+1<=x1;
 action  := x3' = x3 - 1,x1' = start2,stop2' = x2;
};


transition t_15 :={
 from    := aff ;
 to      := aff ;
 guard   := m>=0 && start2>=x2 && stop2>=x2 && x1>=x2 && x2>=1 && x3>=1;
 action  := x1' = start2,stop2' = x2,x3' = x3 - 1,x2' = x2 - 1;
};


transition t_16 :={
 from    := aff ;
 to      := aff ;
 guard   := m>=0 && start2>=x2 && stop2>=1 && x1>=x2 && x3>=1 && stop2<=x2;
 action  := x1' = start2,x3' = x3 - 1,x2' = x2 - 1;
};


transition t_17 :={
 from    := aff ;
 to      := aff ;
 guard   := m>=0 && start2>=1 && stop2>=x2 && x1>=x2 && x3>=1 && start2+1<=x2;
 action  := x1' = start2,x3' = x3 - 1,stop2' = x2,x2' = start2;
};


transition t_18 :={
 from    := aff ;
 to      := aff ;
 guard   := m>=0 && start2>=1 && stop2>=1 && x1>=x2 && x3>=1 && stop2<=x2 &&
            start2+1<=x2;
 action  := x1' = start2,x3' = x3 - 1,x2' = start2;
};
```

```
transition t_19 :={
 from    := aff ;
 to      := aff ;
 guard   := m>=0 && start2>=x2 && stop2>=x1 && x1>=1 && x3>=1 && x1<=x2;
 action  := x1' = start2,stop2' = x1,x3' = x3 - 1,x2' = x2 - 1;
};


transition t_20 :={
 from    := aff ;
 to      := aff ;
 guard   := m>=0 && start2>=x2 && stop2>=1 && x3>=1 && x1<=x2 && stop2<=x1;
 action  := x1' = start2,x3' = x3 - 1,x2' = x2 - 1;
};


transition t_21 :={
 from    := aff ;
 to      := aff ;
 guard   := m>=0 && start2>=1 && stop2>=x1 && x1>=1 && x3>=1 && x1<=x2 &&
            start2+1<=x2;
 action  := x1' = start2,x3' = x3 - 1,stop2' = x1,x2' = start2;
};


transition t_22 :={
 from    := aff ;
 to      := aff ;
 guard   := m>=0 && start2>=1 && stop2>=1 && x3>=1 && x1<=x2 && stop2<=x1 &&
            start2+1<=x2;
 action  := x1' = start2,x3' = x3 - 1,x2' = start2;
};


transition t_23 :={
 from    := aff ;
 to      := aff ;
 guard   := start2>=x1 && stop2>=x1 && x1>=1 && x3>=1 && x1<=x2;
 action  := x3' = m,stop2' = x1 - 1,start2' = x1 - 1,x1' = x1 - 1,x2' = x1 - 1;
};


transition t_24 :={
 from    := aff ;
 to      := aff ;
 guard   := start2>=x2 && stop2>=x2 && x1>=x2 && x2>=1 && x3>=1;
 action  := x3' = m,stop2' = x2 - 1,start2' = x2 - 1,x1' = x2 - 1,x2' = x2 - 1;
};


transition t_25 :={
 from    := aff ;
 to      := aff ;
 guard   := start2>=1 && x3>=1 && start2<=stop2 && start2<=x1 && start2<=x2;
```

```
    action  := x3' = m,stop2' = start2 - 1,start2' = start2 - 1,x1' = start2 - 1,
               x2' = start2 - 1;
  };


  transition t_26 :={
   from    := aff ;
   to      := aff ;
   guard   := start2>=stop2 && stop2>=1 && x3>=1 && stop2<=x1 && stop2<=x2;
   action  := x3' = m,stop2' = stop2 - 1,stop2' = stop2 - 1,x1' = stop2 - 1,
               x2' = stop2 - 1;
  };


  transition t_27__tsplit0 :={
   from    := aff ;
   to      := stop ;
   guard   := m>=0 && stop2<=0;
   action  := ;
  };


  transition t_27__tsplit1 :={
   from    := aff ;
   to      := stop ;
   guard   := m>=0 && x1<=0;
   action  := ;
  };


  transition t_27__tsplit2 :={
   from    := aff ;
   to      := stop ;
   guard   := m>=0 && x2<=0;
   action  := ;
  };


  transition t_27__tsplit3 :={
   from    := aff ;
   to      := stop ;
   guard   := m>=0 && start2<=0;
   action  := ;
  };


  transition ____inittransition :={
   from    := ____start ;
   to      := start ;
   guard   := true;
   action  := m' = m__o,start2' = start2__o,stop2' = stop2__o,x1' = x1__o,
               x2' = x2__o,x3' = x3__o;
  };


}
```

```
strategy xx {

Region init:={ state = ____start && true};

}


// Result of Analysis
// invariant stop := start2<=m && stop2<=m && x2<=m && x1<=m && x3<=m && m=m__o ;
// invariant start := x3=x3__o && x2=x2__o && x1=x1__o && stop2=stop2__o
                      && start2=start2__o && m=m__o ;
// invariant aff := start2<=m && stop2<=m && x2<=m && x1<=m && x3<=m && start2>=0
                      && stop2>=0 && x2>=0 && x1>=0 && x3>=0 &&m=m__o ;
// invariant ____start := true ;

//widening aff;
```

Ce programme met donc l'outil en difficulté, les essais pratiques ne sont pas concluants.