École Normale Supérieure de Cachan
Master Parisien de Recherche en Informatique (MPRI)

Channel machines

Florent Martin

Encadré par James Worrell
Oxford University (Oxford Computing Lab)
Stage du 29 Mai au 10 Août 2007

Dans une première partie, en français, je présenterai les grandes orientations qui ont guidées mon stage, sans entrer dans les détails techniques. Dans la deuxième partie, redigée en anglais, on peut trouver les définitions (en particulier en ce qui concerne la fair-terminaison qui constitue le point de départ de mon stage), et les résultats que j'ai obtenus en collaboration avec James Worrell.

1 Le sujet

J'ai effectue mon stage à Oxford sous la direction de James Worrell. Le sujet qu'il m'a proposé consistait d'abord à étudier une certaine classe de machines à canaux (avec renommage) avec insertion, plus precisément le problème de la fair-terminaison. Dans un deuxième temps, je me suis concentré sur l' étude du même problème en supprimant la primitive de renommage. Dans tous les cas, je ne me suis intérressé qu'à des machines avec un unique canal.
Les machines à canaux ont étés introduites il y à plusieurs années, et représentent une modèlisation pertinante d'une exécution parallèle. Cependant, les machines à canaux sont de manière non surprenantes Turing-complète. De maniere surprenante en revanche, en autorisant des insertions (ou des suppressions), le modèle devient en grande partie analysable.
Pour faire cours, une machine à un canal est la donnée d'un automate fini tel que chaque transition d'un état à un autre est étiquetée par une action
$!a$ qui signifie que l'on doit ajouter un a à la queue du canal
ou $?a$ qui signifie que l'on doit lire (c'est à dire supprimer) un a à la tête du canal.
Une configuration est donc la donnée d'un état courant de l'automate, et la donnée du contenu du canal, c'est à dire d'une file. La primitive de l'insertion rend possible d'ajouter à tout moment et à n'importe quel endroit du canal et sans limite, des symboles.
Dans [FS01], des résultats généraux concernants ces deux modèles(avec insertion ou suppression) sont donnés, dans un cadre général qui met bien en perspective le rôle joué par le fait de pouvoir insérer (ou supprimer) des symboles. Le problème de l'accessibilte (étant donné deux configurations, existe-t-il un chemin qui mène de l'une a l'autre) qui est décidable, s'avère en fait avoir une complexité non primitive-récursive [Sch02]. Certaines idées présentes dans cet article (en particulier en ce qui concerne les simulations) se retrouvent dans la preuve de PSPACE-complètude plus-bas.
A la suite de l'article [OW06], mon superviseur, avait en fait pu ramener le problème de la satisfaisabilité de safety-MTL au problème de la fair-terminaison pour les machines à canaux avec insertion. Avec [OW05] et [BMOW07] cela établit donc un nouveau lien entre les machines à canaux à insertion et la logique MTL.
Le problème de la terminaison pour les machines à insertion ne présente en fait aucun intérêt, car il ne fait que mettre en lumière l'existence ou non d'un chemin infini dans la graphe de l'automate. En particulier, la réponse à la question ne dépend que de l'état de la configuration, et non de son contenu. Presenté rapidement, une execution infinie sera dite fair si tous les symboles qui sont ajoutés au canal sont ultèrieurement lus.

2 Le déroulement du stage

Apres une première période de lecture et de familiarisation avec le sujet, j'ai pu m'atteler au problème de la fair-terminaison des machines à insertion avec renommage. La solution du problème n'était en fait pas tres eloignée. Comme me l'avait présenté mon superviseur, en utilisant le lemme de Higman, l'ensemble des configurations qui ne sont pas terminantes peut se caractériser comme le plus grand point fixe d'une certaine fonction Φ, et le lemme de Highman permettait d'assurer que ce plus grand point fixe pouvait être obtenu en effectuant un nombre fini d'itérations de Φ. La double contrainte du renommage et d'une execution fair et infinie, rendait la fonction Φ complexe, et le principal problème a été de prouver que cette fonction était calculabe en un certains sens. Cependant, en utilisant des techniques d'automate fini, on a pu montrer comment étant donné un langage régulier $\mathcal{L}$ calculer $\Phi(\mathcal{L})$. Cela nous a donc permis de conclure que le problème était décidable.
La suite logique de ce problème consistait à s'intéresser à la complexite du problème de la fair-termination. La primitive du renommage etant en quelque sorte moins naturelle que le problème (moins general) de la fair-terminaison pour les machines à insertion "standard", c'est à ce dernier que je me suis attelé. Le problème de la fair-terminaison pour une machine à un canal avec insertion étant un cas particulier de la fair-terminaison pour une machine à canal avec insertion et avec renommage, le résultat de décidabilité présenté ci-dessus se transportait naturellement. Notre preuve reposait inéluctablement sur le lemme de Higman, et comme déjà mentionné, le résultat de [Sch02] tend à faire croire que, sans autre outil extérieur, il est impossible de prouver une complexite récursive-primitive. Malgré ce résultat quasi rédibitoire, et le sceptissisme de mon superviseur qui m'encourageait malgré tout à suivre mon instinct, je me suis intéressé pour un temps au lemme de Higman du point de vue de la complexité, c'est à dire à considérer quelle était la taille maximale d'une séquence de mots qui ne contredisaient pas le lemme de Higman, en supposant la taille des mots controlée. J'en arrivais finalement à la conclusion la plus naturelle, bien que ce ne soit pas celle que j'attendais, à savoir que cette taille n'est pas récursive-primitive.
Puis j'ai consacré le reste de mon stage a étudier le problème de la fair terminaison pour les machines à insertion à un canal. Le résultat obtenu est alors que le probleme la terminaison est PSPACE-complet pour un alphabet de trois lettres ou plus, et NP-complet dans le cas d'un alphabet de deux lettres. Dans tous les cas, l'idée directrice est d'introduire ce que j'ai appelé des bonnes composantes connexes de l'automate associé à notre machine á un canal. Une bonne composante connexe d'une machine à canal est une composante connexe de l'automate qui est tel que tout symbole a qui est ecrit dans cette composante (i.e. pour toute transition de la forme $!a$) on doit pouvoir lire ce même symbole dans cette composante (i.e. il doit y avoir une transition étiquetée par $?a$). Il est alors facile de voir qu'une configuration qui est dans un état d'une telle bonne composante, avec un canal composé uniquement des "bonnes" lettres que la composante pourra lire, est une configuration d' où il sera possible de faire une execution infinie et fair (en restant précisemment dans cette bonne composante). Réciproquement, si on considère une execution infinie, on peut définir la composante connexe qui sera visitée infiniement souvent. Il s'agit aussi d'une bonne composante connexe. A partir de là, l'idée consiste à dire que si on effectue un certains nombres de cycles au cours d'une execution, on a

nécessairement du passer dans une bonne composante connexe, avec une bonne configuration.
Une nouvelle question qui surgissait alors était de considérer le problème de la terminaison, en ajoutant en plus des contraintes sue chaque états. Précisément, l'idée consiste à associer à chaque états de l'automate un ensemble de mots clos par $\sqsubseteq$, et d'éxiger que pour qu'une execution soit considérée correcte, elle doit à chaque fois verifier la contrainte associée avec chaque état. Il s'agit d'une extension des machines à canaux avec test d'occurence.Or ce problm̀e a déjà été identifié comme un problème nécessitant une tour linéaire d'exponnentiel. Mon sentiment était et est toujours que ce résultat doit en fait pouvoir s'étendre au cas des contraintes ensemblistes. Malheureusement, aucune piste réelle sur le chemin de la solution ne s'est pas présentée.

3 Introduction

3.0.1 The general framework

Channel machines is a quite natural framework that allows to describe infinite transition systems. Not surprisingly though, this model is equivalent to the model of the Turing machines, and all non-trivial problem are there undecidable. More surprisingly, if we allow in the model insertion (or deletion), the transition system ranges in the category of the weel-structured transition systems described in [FS01] and most of the probleme become then decidable (in particular reachability, and non termination for the lossy channel machines). These general decidability results hold on Higman's Lemma. Petri Net is another exemple of infinite system which general decidabilty results, which also can enter in the framework of [FS01], using this time Dickson's lemma instead of Higman's Lemma. In both, this is the property of well-quasi order sets which is the key point.
We are here going to focus on insertion channel machine with only one channel, and introduce for it the notion of an infinte fair computation (in fact, the "classical" notion of non termination for an insertion channel machine is irrelevant). We'll first obtain a decidability result with the primitive of renaming. But our main contribution is to prove that the set of configuration that are non (fairly) terminating is a regular-set (actually a down-set for the order $\sqsubseteq$) that can be calculated in PSPACE, and that the membership problem is NP-complete for an alphabet of two letters, and PSPACE-complete for an alphabet of three letters (and remains in PSPACE if the length of the alphabet is bigger).

4 CAROTS

4.1 the subwordorderer and Higman's Lemma

Definition 1 *For two words $u, v \in \Sigma^*$ we write $u \sqsubseteq w$ if w can be obtained by u in adding some letters. Formally, if $u = u_1...u_n$ where the u_i are some letters of Σ, then there exist some words $w_0, \ldots, w_n$ such that $v = w_0.u_1.w_1.u_2...u_n.w_n$.*

$\sqsubseteq$ is then an order on Σ^*, which is obviously non total, and has the remarkable property to be a well-quasi order set :

Proposition 1 (Higman's Lemma) *Let $(w_i)_{i\in\mathbb{N}}$ be a sequence of elements of Σ^*, then there exists $i < j$ such that $w_i \sqsubseteq w_j$.*

Definition 2 *A subset L of Σ^* will be called an upper set, if for every $w \in L$ and for every v such that $w \sqsubseteq v$ then we have $v \in L$. Dually, L we'll be said to be a down, or a lower set if for every $w \in L$ and for every $v \sqsubseteq w$ we have $v \in L$.*

The simplest example of upper sets that come to mind are sets that we would be tempted to call principal upper sets : $\uparrow w \overset{def}{=} \{u \in \Sigma^* \mid w \sqsubseteq u\}$ where w is then a word of Σ^*. More generally, any finite union of principal upper sets is again an upper set. A direct consequence of Higman's Lemma is that every upper set L is in fact a finite union of principal upper sets. In that case, if $L = \bigcup_{i=1,\dots,n} \uparrow w_i$ we'll say that the set $\{w_1, \dots, w_n\}$ forms a *basis* of L. Conversely since the complement of every down set is an upper set, we can finitely describe a down set by giving a finite basis of its complement, such a basis will be called a *cobasis*.
Here is another consequence of Higman's Lemma

Proposition 2 *Any increasing sequence of upper sets $L_1 \subseteq L_2 \subseteq \dots \subseteq L_i \subseteq \dots$ must be stationary, and then dually, any decreasing sequence of lower set is also stationary.*

4.2 Definitions

Definition 3 *A* Channel Automaton with Renaming and Occurrence Testing *(CAROT) is a tuple $\mathcal{C} = (S, s_0, \Sigma, \Delta, F)$, where S is a finite set of control states, $s_0 \in S$ is the initial control state, $F \subseteq S$ is a set of accepting control states, Σ is a finite channel alphabet and $\Delta \subseteq S \times Op \times S$ is the set of transition rules, with $Op = \{\sigma!, \sigma? \mid \sigma \in \Sigma\} \cup \{zero(\sigma) \mid \sigma \in \Sigma\} \cup \{R \mid R \subseteq \Sigma \times \Sigma_\varepsilon\}$ the set of operations. Given a rule $\tau \in \Delta$, we denote the corresponding operation $op(\tau)$. Intuitively, $zero(\sigma) \in Op$ guards against the occurrence of σ in the channel, and $R \in Op$ is interpreted as a global renaming (where renaming to ε corresponds to deletion).*

A *global state* or *configuration* of $\mathcal{C}$ is a pair $\gamma = (s, x)$, where $s \in S$ is the control state and $x \in \Sigma^*$ is the channel contents. Define $Conf$ to be set of all configurations of $\mathcal{C}$. The rules in Δ induce a transition relation on the set of global states according to the following table, where, given $x = x_1 \dots x_n \in \Sigma^*$ and $R \subseteq \Sigma \times \Sigma_\varepsilon$, $R(x) \overset{\text{def}}{=} \{y_1 \cdots y_n \in \Sigma^* : x_i \; R \; y_i\}$.

Rule	Transition
$(s, \sigma!, t)$	$(s, x) \to (t, x \cdot \sigma)$
$(s, \sigma?, t)$	$(s, \sigma \cdot x) \to (t, x)$
$(s, zero(\sigma), t)$	$(s, x) \to (t, x)$, if $\sigma \notin x$
(s, R, t)	$(s, x) \to (t, y)$, if $y \in R(x)$

A computation of $\mathcal{C}$ is a (finite or infinite) sequence of transitions $\gamma_0 \to \gamma_1 \to \gamma_2 \to \cdots$ with $\gamma_0 = (s_0, \epsilon)$. In fact, a zero-testing $zero(\sigma)$ can be simulated by the relation $R_\sigma = \{(x, x) \mid x \in \Sigma \text{ and } x \neq \sigma\}$. Indeed, $u \; R_\sigma \; v$ if and only if

$u = v$ and $\sigma \notin u$. Hence, $(s,u) \xrightarrow{zero(\sigma)} (t,v)$ if and only if $(s,u) \xrightarrow{R_\sigma} (t,v)$, and then, if one replace all the transitions $s \xrightarrow{zero(\sigma)} t$ by a new one, $s \xrightarrow{R_\sigma} t$, one can effectively simulate a general CAROT by a channel automaton with only, read, write, and renaming operations. Now, associate some relations to every transition of $\mathcal{C}$

$op(\tau)$	$\sigma!, \sigma?$	R
R_τ	Id	$R \cup \{\varepsilon, \varepsilon\}$

And we define the submonoid of the relations over Σ generated by these relations $R_{\mathcal{C}}$. In fact, as previously said, even without the primitive of the renaming, a channel machine is Turing-Powerfull, but two variants have already been studied : Lossy channels Machines, and Insertion Channel machines, for which some classical problems are decidable. We are going to focus, and hence explain the second model. In the insertion model, one allows transition of the form $(s,u) \to (t,v)$ provided that there exist u' and v' such that $u \sqsubseteq u'$ and $v' \sqsubseteq v$ and that $(s,u') \to (t,v')$ in the model previously defined. Until the end, we are in fact going to work on an intermediate model : the lazy model. The restriction we are going to make is that one will be allowed to add a letter on the tape only if this letter is added on the head of the tape and is consumed by the next transition. Hence one could also define the semantic of the lazy model of a Channel machine with renaming as the perfect one, except that every transition of the form $s \xrightarrow{?\sigma} t$ can be taken, without any consequence on the tape.

Definition 4 *An infinite fair computation of* $\mathcal{C}$ *, is an infinite computation* $(s_0,x_0) \to (s_1,x_1) \to \ldots$ *such that every symbol that is written on the tape is eventually erased from the tape. Define then*

$$INF = \{(s,x) \mid \textit{there exists an infinite fair computation starting at } (s,x)\}$$

Our aim is to show that the set INF is regular and computable.

4.3 The fixed-point characterization

We are now going to divide the possible transitions $(s,u) \to (t,v)$ in two categories

- The progressive transitions where the symbol at the head of the tape has *really* been read. That is to say, whether a *read transition* $s \xrightarrow{?\sigma} t$ where we are considering a perfect transition, whether a transition $(s,u) \xrightarrow{R} (t,v)$ where $(\sigma,\epsilon) \in R$ and a σ has effectively been deleted.
- We call conservative transitions every transitions that are not some progressive transitions of the form $s \xrightarrow{?\sigma} t$, that is to say , all the transitions of the form $s \xrightarrow{!\sigma} t$, $s \xrightarrow{R} t$ or $s \xrightarrow{?\sigma} t$ where this last transition was token in the lazy model.

It is then clear that given an infinite computation $(s_0,x_0) \to (s_1,x_1) \to \ldots$, this is an infinite fair computation, if and only if it contains indefinitely many progressive transitions. And hence,

Proposition 3 *Given a Channel Machine with renaming $\mathcal{C}$ and a configuration (s,u), there is an infinite fair computation starting at (s,u), that's to say $(s,u) \in INF$ if and only if there is an infinite computation starting at (s,u) with indefinitely many progressive transitions*

Definition 5 *Let (s,u) be a configuration of a Renaming channel machine $\mathcal{C}$ define*

$$Pred_+(t,v) = \left\{ \begin{array}{l} (s,x) \textit{s.t. there exists a computation} \\ (s,x) = \gamma_0 \to t_1\gamma_1 \to t_2 ... \to t_n\gamma_n = (t,v) \\ \textit{ with } n \geq 1 \textit{ , and } t_i \textit{ is a conservative transition for } i < n \\ \textit{and a progressive transition for } i = n. \end{array} \right\}$$

$Pred_+$ can be extended to a monotonous function of $(Conf, \subseteq)$ on itself, and then has a greatest fixed-point. It's convenient to represent a set of configurations $\mathcal{L}$ as the union of sets of configurations in the same state, i.e. to consider that $\mathcal{L} = \underset{s\in S}{\cup}\mathcal{L}_s$. Then we will abusively confuse the set of configurations $\mathcal{L}_s$ with the language $\mathcal{L}_s \overset{def}{=} \{x \in \Sigma^* \mid (s,u) \in \mathcal{L}\}$ What we mean then by a lower set (resp. a regular set) of configuration $\mathcal{L}$, is a set of configuration $\mathcal{L} = \underset{s\in S}{\cup}\mathcal{L}_s$, where every $\mathcal{L}_s$ is a lower set (resp. a regular set).

Proposition 4 *INF is a lower-set and the greatest fixed-point of $Pred_+$*

Proof :First, if $(s,u) \in INF$ then one can find an infinite fair computation

$$(s,u) = (s_0,u_0) \to (s_1,u_1) \to ... \qquad (1)$$

Now, if $\gamma \sqsubseteq (s,u)$, that is to say, if $\gamma = (s,v)$ with $v \sqsubseteq u$ then there exists an infinite computation

$$(s,v) = (s_0,v_0) \to (s_1,v_1) \to ... \qquad (2)$$

where the $s_i's$ are really the same than in the computation (1). This is possible because if $x \sqsubseteq y$ and $(s,y) \to (t,y')$, then in the lazy model, there's also a computation $(s,x) \to (t,x')$ with $x' \sqsubseteq y'$ .Hence one can show (2) by induction. Now, since (1) is a fair computation, there exists a step n such that at that moment all the letters that were initially in the tape have been consumed, that is to say, all the letters of u have been read or deleted. Then all the letters that appear on the u_i for $i \geq n$ have been added to the tape by some $q \xrightarrow{!\sigma} q'$ transitions. And then after this $n-th$ step, one can assume that the simulation in (2) is perfect, i.e. $\forall i \geq n \ v_i = u_i$. And hence (2) is also an infinite fair computation.
Let's show now that $Pred_+(INF) = INF$. First, if $(t,v) \in INF$ then if $(s,u) \in Pred_+(INF)$ there exists a computation

$$(s,u) \xrightarrow{*} (t,u)$$

Since there is an infinite fair computation starting at (t,u), there's also an infinite fair computation starting at (s,u), and then $Pred_+(INF) \subseteq INF$. Conversely, if $(s,u) \in INF$ then there exists an infinite fair computation

$$(s,u) = \gamma_0 \xrightarrow{*} \gamma_1' \xrightarrow{t_1} \gamma_1 \xrightarrow{*} \gamma_2' \xrightarrow{t_2} \gamma_2 ...$$

where all the t_i are some progressive transition and $\gamma_i \xrightarrow{*} \gamma'_{i+1}$ is a sequence of conservative transitions. Then, $(s,u) \in Pred_+(\gamma_1)$ and it's clear that $\gamma_1 \in INF$. Then $(s,u) \in Pred_+(INF)$ and finally $INF = Pred_+(INF)$. Now if F is a fixed-point of $Pred_+$, then let $(s,u) \in F$. One can then construct inductively an infinite fair computation

$$(s,u) = \gamma_0 \xrightarrow{+} \gamma_1 \xrightarrow{+} \gamma_2 ...$$

with $\gamma_i \in F \; \forall \; i$. All you have to do is to define γ_{i+1} as an antecedent of γ_i for $Pred_+$ in F . $\square$

Our goal is to compute the set INF.Then, we would like to be able to say that the decreasing sequence $Pred_+^n(Conf)$ eventually stabilizes and then, this would produce the greatest fixed-point of $Pred_+$, INF. But this sequence might not stabilize. Hence we are going to define another function, very close to $Pred_+$

Definition 6 *For $X \subseteq Conf$ define*

$$\Phi(X) = Conf \backslash \uparrow (Conf \backslash Pred_+(X))$$

Equivalently, $\Phi(X)$ is the least lower set contained in $Pred_+$.

Φ is also a monotonous, and moreover, $\forall X \in Conf \; \Phi(X)$ is a lower set. Hence $\Phi^n(Conf)$ is a decreasing sequence of lower and then must stabilize. Moreover, if one write GFP for Greatest fixed point :

Proposition 5 *GFP Φ = GFP $Pred_+$*

Proof : First

$$\Phi(GFPPred_+) = Conf \backslash \uparrow (Conf \backslash Pred_+(GFPPred_+))$$

$$= Conf \backslash \uparrow (Conf \backslash GFPPred_+)$$

$$= GFPPred_+ \quad \text{since } INF = GFPPred_+ \text{ is downward-closed}$$

$GFPPred_+$ is then a fixed-point of Φ. Moreover,since $\forall X \in Conf \quad \Phi(X) \subset Pred_+(X)$ if F is a fixed-point of Φ , then $F \subseteq Pred_+(F)$ and then $F \subseteq GFPPred_+$. It's then true that $GFP(\Phi) = GFP(Pred_+)$. $\square$

Since we now know that the increasing sequence $\Phi^n(Conf)$ will stabilize at some point, we only need to be able to compute this sequence. By definition of Φ , whatever the set X is, $\Phi(X)$ is a lower set. A consequence of Higman's lemma is that an upper set X is in fact the open closure of a finite set; we call such finite sets basis of X, hence, every lower set is also regular, and can then be presented, as a regular (with an automaton for instance), or with a co-basis, that's to say by a basis of the upper set consisting of the complement of X. Our aim, is now to show that given a lower set I, presented in one of this way, we are able to compute a presentation of $Pred_+(I)$, and hence a presentation of $\Phi(I)$.

Definition 7 *If X is a state of configuration, define*

$$Pred_{prog}(X) = \{\gamma \in Conf \mid \textit{ there is a progressive transition } \gamma \to \gamma' \;, \gamma' \in X\}$$

and

$Pred_{cons}(X) = \{\gamma \in Conf \mid$ *there is a conservative transition* $\gamma \rightarrow \gamma'$, $\gamma' \in X\}$

Then for $X \in Conf$*, the following holds :*

$$Pred_{+}(X) = Pred^{*}_{cons} \circ Pred_{prog}(X)$$

Proposition 6 *Given a regular language* $\mathcal{L}$ *, then* $Pred_{prog}(\mathcal{L})$ *is a regular language that can effectively be calculated.*

Proof : Write $\mathcal{L} = \underset{s \in S}{\cup} \mathcal{L}_s$.
For every $s \in S$ if we have in $\mathcal{C}$ the transition

$$\alpha \quad t \xrightarrow{?\sigma} s$$

define then, abusively $\mathcal{L}^{\alpha}_{s} = (t, \mathcal{L}_s.\{\sigma\})$.
An the other hand, if we have a transition $\alpha \quad : \quad t \xrightarrow{R} s$ then if we define $\Sigma^R = \{\sigma \in \Sigma^* \mid \sigma \ R \ \epsilon\}$ then define $L^{\alpha}_{s}= (t, R^{-1} \ (\mathcal{L}_s).\Sigma^R$ which can then be empty, if Σ^R is empty, i.e. if R induces no deletion.
Then $Pred_{prog}(\mathcal{L}_s) = \underset{\alpha = \ s \rightarrow t}{\cup} \mathcal{L}^{\alpha}_{s}$.

Proposition 7 $(s, x) \in Pred^{*}_{cons}(t, y)$ *i.e there exists a computation* $(s, x) \xrightarrow{cons^*} (t, y)$ *, if and only if* $\exists \ x'$ *and* $z \ \in \Sigma^*$ *and* $R{\in}R_{\mathcal{C}}$ *s.t.* $y = x'.z$ *and* $x \ R \ x'$ *and there exists a (maybe empty) computation*

$$(s, \epsilon) = (s_0, z_0) \xrightarrow{\alpha_1} (s_1, z_1) \xrightarrow{\alpha_2} ... \xrightarrow{\alpha_n} (s_n, z_n) = (t, z)$$

where every α_i *is a conservative transition and* $R_{\alpha_n} \circ R_{\alpha_{n-1}} \ ... \ R_{\alpha_1}{=}R$ *.*

Proof : If $x \ R \ x'$ then since $R{=}R_{\alpha_n} \circ R_{\alpha_{n-1}} \ ... \ R_{\alpha_1}$ there exists a sequence $(x_i)_{i=0..n}$ with $x_0 = x$ and $x_n = x'$ and $x_{i-1} \ R \ x_i$ for $1 \leq i \leq n$. Then it is easy to check that

$$(s_0, x_0.z_0) \xrightarrow{\alpha_1} (s_1, x_1.z_1) \xrightarrow{\alpha_2} ... \xrightarrow{\alpha_n} (s_n, x_n.z_n)$$

is a computation of $\mathcal{C}$ with only conservative transitions.
Conversely, if $(s, x) \in Pred^{*}_{cons}(t, y)$ we can find a computation made of conservative transitions :

$$(s, x) \xrightarrow{\alpha_1} ... \xrightarrow{\alpha_n} (t, y)$$

Since conservative transitions can only rename the tape, or add a letter on the right of the tape, we can then define a sequence $x_0, x_1..., x_n$ which will represent the content of the tape at the $i-th$ step corresponding to the letters initially belonging to x . Hence we'll have $x_0 = x$ and $x_{i-1} \ R_{\alpha_i} \ x_i$ and if we define too $z_0, .., z_n$ to be the string that will have been added to the tape till the beginning of the computation(hence $z_0 = \epsilon$), we have $(s, \epsilon) \xrightarrow{\alpha_1} ... \xrightarrow{\alpha_n} (t, z_n)$ and finally $y = x_n.z_n$ where x_n plays the role of x' in the proposition, and this ends our proof. □

Proposition 8 *Given a regular set of configuration* $\mathcal{L}$*, one can effectively calculate* $Pred^{*}_{cons}(\mathcal{L})$

Proof : First, if $\mathcal{L} = \underset{s\in S}{\cup}\mathcal{L}_s$, then

$$Pred^*_{cons}(\mathcal{L} = \underset{s\in S}{\cup} Pred^*_{cons}(\mathcal{L}_s))$$

Now, for s and t in S , and for $R{\in}R_{\mathcal{C}}$, define

$$\mathcal{L}_{R,\ s\to t} = \left\{ \begin{array}{c} x \in \Sigma^* \quad \text{such that there exists a computation} \\ (s,\epsilon) = (s_0, x_0) \xrightarrow{\alpha_1} (s_1, x_1) \xrightarrow{\alpha_2} ... \xrightarrow{\alpha_n} (s_n, x_n) = (t, x)\ (P) \\ \text{with } R_{\alpha_n} \circ ... \circ R_{\alpha_1} {=} R \\ \text{and all the } \alpha_i \text{ are conservative transitions} \end{array} \right\}$$

We are going to show that this set is regular. In order to do that we define a finite non deterministic state automaton $\mathcal{A} = (R_{\mathcal{C}} \times S, , \Sigma, \Delta')$
We now define Δ' :

- if $q \xrightarrow{!\sigma} q' \in \Delta$ then $\forall\ R \in R_{\mathcal{C}}$ and $\forall \tau \in \Sigma$ s.t. $\sigma\ R\ \tau$,$(R,q) \xrightarrow{\tau} (R, q') \in \Delta'$. Note that τ can possibly equal ϵ.
- if $q \xrightarrow{?\sigma} q' \in \Delta$ then $\forall\ R \in R_{\mathcal{C}}$ $(R,q) \xrightarrow{\epsilon} (R,q') \in \Delta'$
- if $q \xrightarrow{S} q'$ for some $S \in R_{\mathcal{C}}$
then $\forall\ R, R' {\in} R_{\mathcal{C}}$ $s.t.\ R{=}R' \circ S$ $(R,q) \xrightarrow{\epsilon} (R',q) \in \Delta'$

We didn't precise neither an initial state, nor a set of final states, because we are going to show that :

$$\mathcal{L}_{R,\ s\to t} = \mathcal{L}(\mathcal{A})_{(R,s)\to(Id,t)} \tag{3}$$

Where the last language is the language of the words accepted by $\mathcal{A}$, starting in the state (R,s) , and arriving in the state (Id,t).
Before showing that properly, just note that the automaton $\mathcal{A}$ *simulates* the behavior of $\mathcal{C}$, starting with an empty tape, and taking only conservative transitions.Moreover, the states of $\mathcal{A}$ allow us to guess at each step of our simulation what will be the global renaming that is going to come until the end. This allows us to know, when a letter is written on the tape, what it will look like at the end of the sequence of conservative transitions.

We now show (3) :

First, if $x \in \mathcal{L}_{R,\ s\to t}$ then by definition, there exists a computation

$$(s,\epsilon) = (s_0, x_0) \xrightarrow{\alpha_1} (s_1, x_1) \xrightarrow{\alpha_2} ... \xrightarrow{\alpha_n} (s_n, x_n) = (t, x)$$

And $R_{\alpha_n} \circ ... \circ R_{\alpha_1} {=} R$.

Then if one define for every $0 \le i \le n$ $R_i{=}R_{\alpha_n} \circ ... \circ R_{\alpha_{i+1}}$ and then by convention $R_n{=}Id$.
Now, for $i = 0$ to n and $j = 1$ to n we define
$l_i^j \overset{def}{=}$ the letter (possibly empty) that has been added to the channel by the $j-th$ transition, i.e. α_j, but as it appears on the channel at the $i-th$ step of (P), that is to say, as it appears in x_i. Hence, l_i^i corresponds to the letter

that has been really written on the channel at the $i-th$ step, which can then by empty if the transition α_i wrote nothing on the channel. Necessarily, for $i<j$, we have $l_i^j=\epsilon$, and for every $i\in\{1..n\}$ $x_i=l_i^1.l_i^2...l_i^n$ and in particular, $x=x_n=l_n^1...l_n^n$. Note also that for $i\geq j$ $l_i^j\ R_{\alpha_{i+1}}\ l_{i+1}^j$, and hence a finite induction yields to $l_i^i\ R_{i-1}\ l_i^n$ accordingly to our previous definition.

Then it can be shown by a finite induction that the following path is a valid computation of $\mathcal{A}$:

$$((R_0,s_0),\epsilon)\xrightarrow{\widetilde{\alpha_1}}((R_1,s_1),l_n^1)\xrightarrow{\widetilde{\alpha_2}}((R_2,s_2),l_n^1.l_n^2)...\xrightarrow{\widetilde{\alpha_n}}((R_n,s_n),l_n^1.l_n^2...l_n^n)=((Id,t),x) \tag{4}$$

Where the transitions $\widetilde{\alpha_i}$ of $\mathcal{A}$ are naturally deduced from the definition we gave of Δ' above.

if we have the following conservative transition α_i in $\mathcal{C}$:

- $$\alpha_i\ :\ (s_{i-1},x_{i-1})\xrightarrow{!\sigma}(s_i,x_i.\sigma) \tag{5}$$

  we will define the transition $\widetilde{\alpha_i}$ this way :
  accordingly to our definitions, $\sigma=l_i^i$ and moreover, we have $l_i^i\ R_{i-1}\ l_n^i$ and then since (5) was a transition of $\mathcal{C}$ we must have in $\mathcal{A}$

  $$(R_{i-1},s_{i-1})\xrightarrow{l_n^i}(R_i,s_i)$$

  which justifies the $i-th$ transition of (4).

- $$\alpha_i\ :\ (s_{i-1},x_{i-1})\xrightarrow{?\sigma}(s_i,x_i) \tag{6}$$

  then since we assume that this is a conservative transition, nothing has been written on the channel, then $l_i^i=l_n^i=\epsilon$, moreover, here $R_{\alpha_i}{=}R_{?\sigma}{=}Id$ and then
  $$R_{i-1}{=}R_{\alpha_n}\circ...\circ R_{\alpha_i}{=}R_{\alpha_n}\circ...\circ R_{\alpha_{i-1}}{=}R_i$$
  and accordingly to our definition of Δ' since (6) is a transition of $\mathcal{C}$, the following is a transition of $\mathcal{A}$: $(R_{i-1},s_{i-1})\xrightarrow{\epsilon}(R_i,s_i)$

- $$\alpha_i\ :\ (s_{i-1},x_{i-1})\xrightarrow{S}(s_i,x_i) \tag{7}$$

  Then accordingly to our definition of the R_j we must have $R_{i-1}{=}S\circ R_i$ and then accordingly to our definition of Δ' and since we assume that (7) belongs to Δ we must have in $\mathcal{A}$

  $$(R_{i-1},s_{i-1})\xrightarrow{\epsilon}(R_i,s_i)$$

  Then, nothing has been added to the channel, so $(l_j^i)_j$ is the ϵ-sequence as in the previous case, and then this relation justifies the $i-th$ transition of (4)

Which proves that $\mathcal{L}_{R,\ s\to t} \subseteq \mathcal{L}(\mathcal{A})_{(R,s)\to(Id,t)}$

Conversely, suppose that $x \in \mathcal{L}(\mathcal{A})_{(R,s)\to(Id,t)}$ that is to say, that we can find a path in $\mathcal{A}$ which looks like

$$((R_0,s_0),\epsilon) \xrightarrow{\beta_1} ((R_1,s_1),l^1) \xrightarrow{\beta_2} ((R_2,s_2),l^1.l^2)... \xrightarrow{\beta_n} ((R_n,s_n),l^1.l^2...l^n) = ((Id,t),x)$$

where we assume that l^i is the letter that was induced by β_i (hence, $l^i = \epsilon$ if β_i is en ϵ-transition). Now, for every transition β_i, define $\widetilde{\beta_i}$ to be an associated transition of $\mathcal{A}$ accordingly to definition we gave of Δ'. Moreover, remind that $R_n{=}Id$. Then every thing has been done when we defined Δ' in order that the following holds :

$$\forall i \geq 1\ \ R_i \circ R_{\widetilde{\beta_i}}{=}R_{i-1}\ \forall n{-}1 \geq i \geq 0\ \ \ R_i{=}R_n \circ R_{\widetilde{\beta_n}} \circ ... \circ R_{\widetilde{\beta_{i+1}}}{=}R_{\widetilde{\beta_n}} \circ ... \circ R_{\widetilde{\beta_{i+1}}}$$

Hence during the transition $((R_{i-1},s_{i-1}),l^1...l^{i-1}) \xrightarrow{\widetilde{\beta_i}} ((R_i,s_i),l^1...l^i)$ the letter l^i (possibly empty)has been added which means that

- If $\widetilde{\beta_i}$ is of the form : $s_{i-1} \xrightarrow{!\sigma} s_i$ then we must have that $\sigma\ R_{i-1}\ \ l^i$ and hence then that $\sigma\ R_{\widetilde{\beta_n}} \circ ... \circ R_{\widetilde{\beta_i}}\ \ l^i$ but since in this case, $R_{\widetilde{\beta_i}}{=}Id$ we have in fact that $\sigma\ R_{\widetilde{\beta_n}} \circ ... \circ R_{\widetilde{\beta_{i+1}}}\ \ l^i$ and then we can define a sequence of letters

$$(l^i_j)_{0\leq j\leq n}\ by\ l^i_j = \begin{cases} \epsilon & \text{if } j < i \\ l^i_i = \sigma & \\ l^i_j\ R_{\widetilde{\beta_{j+1}}}\ l^i_{j+1} & \text{if } i \leq j < n \\ l^i_n = l^i & \end{cases}$$

- If $\widetilde{\beta_i}$ is of the form $s_{i-1} \xrightarrow{?\sigma} s_i$ or $s_{i-1} \xrightarrow{S} s_i$, we define the sequence $l^i_j = \epsilon$ for $0 \leq j \leq n$

Now we can affirm that (8) is a valid computation of $\mathcal{C}$:

$$(s,\epsilon) \xrightarrow{\alpha_1} (s_1,l^1_1) \xrightarrow{\alpha_2} (s_2,l^1_2.l^2_2) \xrightarrow{\alpha_3} ... \xrightarrow{\alpha_n} (s_n,l^1_n.l^2_n..l^n_n) = (t,x) \qquad (8)$$

Which then finishes to prove (3).

We now finish the proof of the proposition : for each $t \in S$, if we write

$$Pred^*_{cons}(\mathcal{L}_t) = \bigcup_{s\in S} \mathcal{L}_{s\to t} \quad \text{where}$$

$$\mathcal{L}_{s\to t} \overset{def}{=} \{(s,x) \mid (s,x) \in Pred^*_{cons}(\mathcal{L}_t)\}$$

Now, if we abusively confuse the set of configurations $\mathcal{L}_{s\to t}$ with a language of Σ^*, proposition 5 tels us that

$$\mathcal{L}_{s\to t} = \{x \in \Sigma^* \mid \exists x', z \in \Sigma^*,\ \exists y \in \mathcal{L}_t\ ,\ \exists\ R{\in}R_{\mathcal{C}}\ \ s.t.\ x\ R\ x'\ ,\ y = x'.z \text{ and } z \in \mathcal{L}_{R,\ s\to t}\}$$

$$= \bigcup_{R\in R_{\mathcal{C}}} \{x \in \Sigma^* \mid \exists x', z \in \Sigma^*,\ \exists y \in \mathcal{L}_t\ \text{ s.t. } x\ R\ x'\ ,\ y = x'.z\ ,\ \text{and } z \in \mathcal{L}_{R,\ s\to t}\}$$

$$= \bigcup_{R \in R_{\mathcal{C}}} \{x \in \Sigma^* \mid \exists x' \in \Sigma^* \text{ s.t. } x \, R \, x' \text{ , and } \; x \in \mathcal{L}_t \setminus \mathcal{L}_{R,\ s \to t}\}$$

But this exactly means that

$$\mathcal{L}_{s \to t} = \bigcup_{R \in R_{\mathcal{C}}} R^{-1} \left(\mathcal{L}_t \setminus \mathcal{L}_{R,\ s \to t}\right)$$

which is a finite union of regular set we are able to calculate. □

Corollary 1 *INF is effectively computable*

We showed at the proposition 3 that INF was the limit of the decreasing sequence of lower-sets $\Phi^n(conf)$, and Propositions 4 and 6 allow us to claim that , given a regular set of configurations $\mathcal{L}$, $Pred_+(\mathcal{L})$ is regular and computable, hence $\Phi(\mathcal{L})$ is regular and computable too. Here we use the fact that if $\mathcal{L}$ is a *real* regular language, then $\uparrow \mathcal{L}$ is also regular and computable (for instance, if $\mathcal{L}$ is recognized by the finite automaton $\mathcal{A}$, $\uparrow \mathcal{L}$ is recognized by $\widetilde{\mathcal{A}}$ where we added loops for all the letters of Σ on every states of $\mathcal{A}$). Hence, we can effectively compute the sequence $\Phi^n(conf)$ and we just have to check at every step of our computation if $\Phi^n(conf) = \Phi^{n+1}(conf)$, which will eventually arrive, and then will give us INF.

5 The bound on the cycles

We now start to focus on insertion channel machines (without renaming), and we are going to prove the results of PSPACE and NP completeness.

Lemma 1 *Let $n \in \mathbb{N}^*$. Then, if $X_1, X_2, .., X_{2n+4}$ is a sequence of subsets of $\{1 \ldots n\}$, there exists $1 \le i < j \le 2n+4$ such that*

$$\bigcup_{k=i}^{k=j} X_k = \bigcup_{k=i+1}^{k=j+1} X_k \ .$$

Proof : We use the fact that in a sequence $U_1, U_2, ...U_{n+2}$, there must exists an $i \in \{1 \ldots n+1\}$ such that

$$U_i \subseteq \bigcup_{k=i+1..n+2} U_k \tag{9}$$

Otherwise, if for each $i \in \{1..n+1\}$ $U_i \nsubseteq \bigcup_{k=i+1..n+2} U_k$ then there exists an integer $\alpha_i \in U_i$ such that $\alpha_i \notin \bigcup_{k=i+1..n+2} U_k$. Hence if for any i (9) doesn't hold, we can define an α_i for all $i \in \{1..n+1\}$. And then, we must have $r < s \Rightarrow \alpha_r \neq \alpha_s$ since $\alpha_s \in U_s$ and $\alpha_r \notin U_s$ (because $r < s$). So $\{\alpha_1, ..\alpha_n\} = \{1, .., n\}$ and necesserily $U_{n+1} = \emptyset$. And then $U_{n+1} \subseteq U_{n+2}$.

Applying (9) to the sequence $V_1 = U_{n+2}, V_2 = U_{n+1}, .., V_{n+2} = U_1$ (which is the reverse of our original sequence) tells us that

$$\exists i \in \{2..n+2\} \text{ s.t. } V_i \subseteq \bigcup_{k=1..i-1} V_k \ . \tag{10}$$

Now, if we consider our sequence
$X_1, .., X_{n+2}, X_{n+3}, .., X_{2n+4}$
applying (9) to $X_1, .., X_{n+2}$ and (10) to $X_{n+3}, .., X_{2n+4}$ allows us to find $i < n+2$ and $n+3 < j$ such that
$X_i \subseteq \bigcup_{k=i+1..n+2} X_k$ and $\bigcup_{k=n+3..j-1} X_k \supseteq X_j$.
And finally

$$\bigcup_{k=i..j-1} X_k = \bigcup_{k=i+1..j} X_k.$$

□

Definition 8 *By a sub-channel machine of $\mathcal{C} = (Q, \Sigma, \Delta)$, we mean a channel machine $\mathcal{C}' = (Q', \Sigma, \Delta')$ with $Q' \subseteq Q$ and $\Delta' \subseteq \Delta$.*

If $\rho = (q_0, w_0) \rightarrow (q_1, w_1) \rightarrow ...$ is an infinite fair computation, then if $\rho_\infty \overset{def}{=}$ {states and transitions that are infinitely often visited by ρ}
it is clear that ρ_∞ forms a *sub-channel machine* of $\mathcal{C}$, which is connected. By connected, we mean that if you look at ρ_∞ as a directed graph whose edges are the states of ρ_∞, and whose vertexes are naturally induced by the transitions, i.e. for every transition $q \xrightarrow{!a} r$ or $q \xrightarrow{?a} r$, you have the edge $q \rightarrow r$, then, this graph must be connected.
Since we suppose that $\rho \in INF$, necessarily, for every transition of ρ_∞ of the form $\xrightarrow{!\sigma}$, there must be a transition of the form $\xrightarrow{?\sigma}$, and there exists at least one transition $\xrightarrow{!\sigma}$ for one letter σ, otherwise the computation couldn't be infinite and fair.

Moreover, by definition of ρ_∞, eventually, the computation ρ must have reached after a certain number of steps, say n, a configuration (q_n, w_n) , such that the infinite computation which is a suffix of ρ :

$$(q_n, w_n) \rightarrow (q_{n+1}, w_{n+1}) \rightarrow ... \qquad (11)$$

must remain in ρ_∞. In particular, since the computation (11) must also be fair, all the letters of w_n will eventually be read, and so, they will be read by some *read-transition* $\xrightarrow{?\sigma}$, of ρ_{inf}. Hence, w_n must contain only such letters that can be read in ρ_∞.

Conversely, if you can find a *sub-channel machine* of $\mathcal{C}$, say $\mathcal{C}'$, which is connected, and such that everything you can write can be read (and such that you can write something), then, if from a state (q, w) you can reach this *sub-channel machine,* with a tape which contains only letter that can be read in it, this ensures that $(q, w) \in INF$. More formally :

Definition 9 *Let G be a sub-channel machine of $\mathcal{C}$. We'll say that G is a good connected component of $\mathcal{C}$, if*

- *considered as a directed graph as we made previously it is a connected graph*
- *there exists a read transition in G, i.e. a transition of the form $q \xrightarrow{!a} r$*

- *everything you can write in G can be read in G. That is to say, for every $\sigma \in \Sigma$, if there is a transition $q \xrightarrow{!\sigma} r$ in G, there must exist a transition $s \xrightarrow{?\sigma} t$ in G.*

With each good connected component$G$, we associate the good alphabet of G, Σ_G to be the set of letters (necessarily non-empty), that can be read in G.
If G is a good connected componentand q is a state of G, and $w \in \Sigma_G^*$, we'll say that (q, w) is a good configuration.

Proposition 9 *$(q, w) \in INF$ if and only if there exists a good connected component G and a computation*

$$(q, w) \xrightarrow{*} (q'w') \text{ with } q' \text{ a state of } G \text{ and } w' \in \Sigma_G^*. \tag{12}$$

In other words, $(q, w) \in INF$ if and only if we can reach a good configuration from (q, w).

Definition 10 *Let ρ be a finite computation :*

$$\rho = (q_0, w_0) \to (q_1, w_1) \to ... \to (q_n, w_n).$$

There is an unique way to decompose ρ in the following way, which we call the cyclic-decomposition :
$\rho = (r'_0, c'_0) \xrightarrow{+} (r_1, c_1) \xrightarrow{\alpha_1} (r'_1, c'_1) \xrightarrow{*} (r_2, c_2) \xrightarrow{\alpha_2} (r'_2, c'_2)...$
$(r_{m-1}, c_{m-1}) \xrightarrow{\alpha_m} (r'_{m-1}, c'_{m-1}) \xrightarrow{*} (r_m, c_m)$
such that $(r'_0, c'_0) = (q_0, w_0)$, $(r_m, c_m) = (q_n, w_n)$, and during $(r'_i, c'_i) \xrightarrow{*} (r_{i+1}, c_{i+1})$, only letters from c'_i will have been read, and $(r_i, c_i) \xrightarrow{\alpha_i} (r'_i, c'_i)$ is a real read-transition that has read a letter that didn't belong to c'_{i-1}, so it is the first read-transition that reads a letter not belonging to c'_{i-1}, or in other words, the first read-transition that read a letter belonging to c_i. We call the integer m the number of cycles of ρ.

It is easy to check that if ρ_1 and ρ_2 are two computations, with respectively n_1 and n_2 cycles, then $\rho_1.\rho_2$ has at most $n_1 + n_2$ cycles. Moreover, if we consider a cyclic-decomposition

$$\rho = (r'_0, c'_0) \xrightarrow{+} (r_1, c_1) \xrightarrow{\alpha_1} (r'_1, c'_1) \xrightarrow{*} (r_2, c_2) \xrightarrow{\alpha_2} (r'_2, c'_2) \dots (r_{m-1}, c_{m-1}) \xrightarrow{\alpha_m} (r'_{m-1}, c'_{m-1}) \xrightarrow{*} (r_m, c_m)$$

it is convenient to note that each computation

$$(r'_0, c'_0) \xrightarrow{+} (r_1, c_1) \dots (r_1, c_1) \xrightarrow{+} (r_2, c_2) \dots (r_{m-1}, c_{m-1}) \xrightarrow{+} (r_m, c_m)$$

is a one cycle-computation, and hence the cyclic decomposition allows us to decompose a m-cycle computation in a concatenation of m 1-cycle computations.
If ρ is an infinite fair computation, we can also define a cyclic-decomposition
$\rho = (r'_0, c'_0) \xrightarrow{+} (r_1, c_1) \xrightarrow{\alpha_1} (r'_1, c'_1) \xrightarrow{*} (r_2, c_2) \xrightarrow{\alpha_2} (r'_2, c'_2)...$

$$(r_{i-1}, c_{i-1}) \xrightarrow{\alpha_i} (r'_{i-1}, c'_{i-1}) \xrightarrow{*} (r_i, c_i) \to ... \tag{13}$$

It has to be remarked that in (13) there are infinitely many steps, in other words, infinitely many cycles, and moreover that, such a decomposition would be impossible if we hadn't assumed that ρ was an infinite fair computation.

Proposition 10 *$(q, w) \in INF$ if and only if there exists a computation ρ starting from (q, w) whose number of cycles is $\geq K$, for $K = |Q|(2|\Sigma| + 4) + 1$.*

Proof : First, if $(q, w) \in INF$, we can find an infinite fair computation ρ, starting from (q, w), and with the previous remark, we know that we can find some prefix of ρ with as many cycles as we wish. This proves the implication from the left to the right.

Now, if ρ is a computation with K cycles, we look at its cyclic-decomposition :

$$(r'_0, c'_0) \xrightarrow{+} (r_1, c_1) \xrightarrow{+} ... \xrightarrow{+} (r_K, c_K). \tag{14}$$

Then, if for a word w we define Σ_w to be the sub-alphabet of the letters of w (Σ_w is then empty if $w = \epsilon$), we can roughly write :

$$(r'_0, c'_0) \xrightarrow[!\Sigma_{c_1}]{?\Sigma_{c'_0}} (r_1, c_1) \xrightarrow[!\Sigma_{c_2}]{?\Sigma_{c_1}} ... \xrightarrow[!\Sigma_{c_K}]{?\Sigma_{c_{K-1}}} (r_K, c_K) \tag{15}$$

Where by $(q, w) \xrightarrow[!\Sigma_2]{?\Sigma_1} (r, x)$, we mean a sequence of transitions, where we'll have written all the letters of Σ_2 and only them, and really read all the letters of Σ_1 and only them. Hence if we've got two computations :
$(q, w) \xrightarrow[!\Sigma_2]{?\Sigma_1} (r, x) \xrightarrow[!\Sigma_3]{?\Sigma_2} (s, y)$, we can conclude that we also have

$$(q, w) \xrightarrow[!\Sigma_2 \cup \Sigma_3]{?\Sigma_1 \cup \Sigma_2} (s, y) \tag{16}$$

Now, we are going to use the pigeon-hole principle, in (14). Since $K \geq |Q|(2|\Sigma| + 4) + 1$, there exists a certain state, say q, that appears $(2|\Sigma| + 4) + 1$ times in (14), which leads, using (16) to a sub computation of (15) which looks like

$$(q, z_1) \xrightarrow[!\Sigma_2]{?\Sigma_1} (q, z_2) \xrightarrow[!\Sigma_3]{?\Sigma_2} ... \xrightarrow[!\Sigma_{(2|\Sigma|+4)+1}]{?\Sigma_{(2|\Sigma|+4)}} (q, z_{(2|\Sigma|+4)+1}) \tag{17}$$

Now using lemma (1), we can find $i < j$ such that

$$(q, z_i) \xrightarrow[!\Sigma_{i+1}]{?\Sigma_i} (q, z_{i+1}) \xrightarrow[!\Sigma_{i+2}]{?\Sigma_{i+1}} ... \xrightarrow[!\Sigma_{j+1}]{?\Sigma_j} (q, z_{j+1}) \tag{18}$$

and

$$\bigcup_{k=i}^{k=j} \Sigma_k = \bigcup_{k=i+1}^{k=j+1} \Sigma_k \ .$$

Then, the path (18) induces a *sub-channel machine* of $\mathcal{C}$, G, which is a *good connected component.* $\square$

Proposition 11 *Let $\mathcal{A} = (P, \Sigma, q_0, F)$ be a finite (non-deterministic) automaton, $p, p' \in P$, then*

$$Pred^{p \to p'}_{1cycle}(\mathcal{L}(\mathcal{A})) \stackrel{def}{=} \{w \mid \ \exists \ (p, w) \xrightarrow{1cycle} (p', w') \ \textit{with} \ w' \in \mathcal{L}(\mathcal{A})\}$$

is regular and computable.

Proof : If $\mathcal{C} = (Q, \Delta, \Sigma)$ then define $\mathcal{B} = (P \times Q, \Delta', \Sigma, (q_0, p), F \times \{p'\})$ where

- for each transition $r \xrightarrow{!a} s$ in Δ and $t \xrightarrow{a} u$ in P, we have $(t,r) \xrightarrow{\epsilon} (u,s)$ in Δ'.
- for each transition $r \xrightarrow{?a} s$ in Δ then for each $q \in P$ we have
  - $(q,r) \xrightarrow{\epsilon} (q,s)$ in Δ' (which corresponds to a false-read)
  - $(q,r) \xrightarrow{a} (q,s)$ in Δ' (which corresponds to a perfect read).

We now claim that $\mathcal{L}(\mathcal{B}) = Pred^{p \to p'}_{1cycle}(\mathcal{L}(\mathcal{A}))$. □

Corollary 2 *Given a channel machine $\mathcal{C}$, and a state q of $\mathcal{C}$, we can construct in* PSPACE *an (non-deterministic) automaton which recognizes INF_q.*

Corollary 3 *Given a channel machine $\mathcal{C}$, it can be decided in* NLOGSPACE *whether INF (or INF_q for a state q) is universal.*

Proof :

- INF is universal if and only if from every states, we can reach a *good connected component* where all the letters can be read, or in other words, if every minimal (for the reachability relation) connected component is a *good connected component* where every letters can be read. Indeed the sense from the right to the left is clear, whereas, if there exists a minimal connected component where a letter, say σ cannot be read, then the configuration (q, σ), for q a state of this connected component, cannot belong to INF.
- INF_q is universal if and only if from q we can reach a *good connected component* where every letters can be read. Here, the right to the left implication is the same as above, whereas if INF_q is universal, then if $\Sigma = \{\sigma_1, .., \sigma_n\}$, then $(q, (\sigma_1.\sigma_2...\sigma_n)^{|Q|+1}) \in INF$, and then there exists an infinite fair computation π starting in this configuration. But then using the pigeon hole principle, we can find a state r that must appear just before reading at least $|Q|+1$ letters of $w = (\sigma_1.\sigma_2...\sigma_n)^{|Q|+1}$. But w is such a word that it implies that the loop of $\mathcal{C}$ starting a the first, and ending at the last of these $|Q|+1$ occurrences of r in π must be a *good connected component* where every thing can be read.
- Now, given a state q, it can be clearly checked in $NLOGSPACE$ whether in $\mathcal{C}$ there exists a path leading to a loop where all the letters can be read. □

6 PSPACE-completeness

Theorem 1 *Given a channel-machine $\mathcal{C}$, and a configuration (q, w), we can decide in* PSPACE *wheither $(q, w) \in INF$.*

Proof : We just use the fact that the membership of a word to an automaton is in NLOGSPACE, that PSPACE=N-PSPACE, and corollary (2). □

Theorem 2 *Given a channel machine $\mathcal{C}$, and a configuration (q, w), it is* PSPACE-*hard to decide weither $(q, w) \in INF$.*

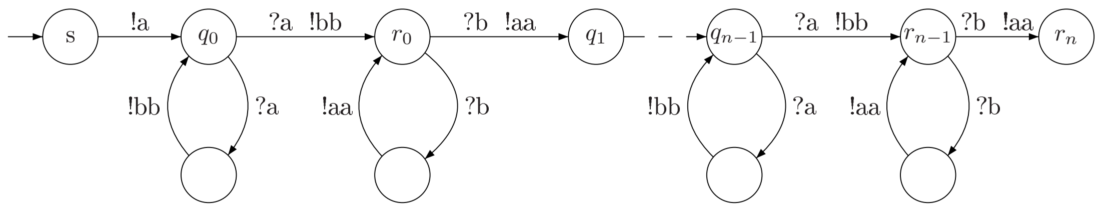
Proof : let $\mathcal{M} = (\Sigma, Z, z_0, z_f, \Gamma)$ be a *linear spacebounded* Turing Machine with states Z, initial state z_0, accepting state z_f, transitions Γ. And let m be a word on this alphabet of length $|m| = M$. We are going to build a channel machine $\mathcal{C}$ on the alphabet $\{a, b, c\}$, such that there will be a state s such that $(s, \epsilon) \in INF$ if and only if m is accepted by $\mathcal{M}$.

First, quite classically, we will encode an accepting computation of $\mathcal{M}$ on m as a string w of the form $m_1 * m_2 * ... m_{2^M} *$, where the m_i will be words on the alphabet $\Sigma \cup Z \cup \{*\}$, and $c_1 = z_0 \cdot m$. On this coding, m_i is supposed to represent the content of the tape, and the location of the head at the i-th step of the computation of $\mathcal{M}$. Hence, m_{2^M} must be in state z_f. It has to be noticed that this is not a restriction to suppose that this computation contains 2^M steps. This alphabet, $\Sigma \cup Z \cup \{*\}$, can be encoded on the alphabet $\{0, 1\}$. Here we encode $\Sigma \cup S \cup \{*\}$ with words on $\{0, 1\}$, of length 2^γ, all endings with a 0. Clearly, with these constraints, the length of γ can remain linear in our input. Moreover, this is also not a restriction to suppose that the input m has a length $2^\tau - 2$, so that on the alphabet $\Sigma \cup S \cup \{*\}$, the string m_i* has length 2^τ, and finally, coded with our coding on $\{0, 1\}$, the string will have a length $2^M \times 2^\tau \times 2^\gamma = 2^\beta$ if $\beta = M + \tau + \gamma$.
Hence, β remains polynomial in the size of our input. Moreover, this is still not a restriction to suppose β even, say $\beta = 2 \times n$.

Now, we give an idea of what $\mathcal{C}$ will do :
First, $\mathcal{C} = (\{a, b, c\}, Q, \Delta)$ will have only one *good connected component*, with only one state, namely F, and on the alphabet $\{b\}$. Hence we'll be allowed to say that $(s, \epsilon) \in INF$ if and only if there exists a path to a good configuration (F, b^λ). In fact we'll show that there exists a path from (s, ϵ) to a good configuration (F, b^λ) if and only if this path is a perfect computation of $\mathcal{C}$.
$\mathcal{C}$ will have such a shape that (if in a perfect behavior) it will in a first time guess a string $w = m_1 * m_2 * ... m_{2^M} *$ of length 2^β, supposed to encode a correct computation of $\mathcal{M}$ on m as described above.
Then, in a second phase, it will check that $w = m_1 * m_2 * ... m_{2^M} *$, is a correct simulation of $\mathcal{M}$. What has to be checked is that

1. the $'*'$ (coded in $\{0, 1\}$) appears at the good positions, that is to say, every $2^{\gamma+\tau}$ symbols. This can be checked in one cycle of the channel.
2. Then it has to be checked that in each m_i there is only one symbol which represents a letter of Z, that is to say there is exactly one head-tape position. This also can be checked in one cycle of the channel
3. finally, it has to be check that every triple of letters (seen on $\Sigma \cup S \cup \{*\}$) $(l_1, l_2, l_3), (l'_1, l'_2, l'_3)$ that appears in say the k-th position (with $2 \leq k \leq 2^M - 1$) of m_i and m_{i+1} satisfy the rules of Γ, the transitions of $\mathcal{M}$.

The first part of $\mathcal{C}$ will be



First we remark that this automaton doesn't possess a *good connected component.* Then we recall that we are interested in the question $(s, \epsilon) \in INF$. Here it appears that starting from (s, ϵ), there are various ways to reach the state r_n. One of these, which corresponds to a perfect run of $\mathcal{C}$, is the following sequence of transitions :

$$(s, \epsilon) \to (q_0, a) \xrightarrow{+} (r_0, bb) \xrightarrow{+} (q_1, aaaa) \xrightarrow{+} ... \xrightarrow{+} (r_{n-1}, b^{2.4^{n-1}}) \xrightarrow{+} (q_n, a^{4^n}) \tag{19}$$

In fact we have got

Lemma 2 *let π be a computation of $\mathcal{C}$ of the form*
$(s, \epsilon) \xrightarrow{*} (q_n, x)$. *Then*

- *if $x = a^\lambda$ with $\lambda \in \mathbb{N}$,necessarily $\lambda \geq 4^n$*
- *Otherwise $ba \sqsubseteq x$*

Proof :
□

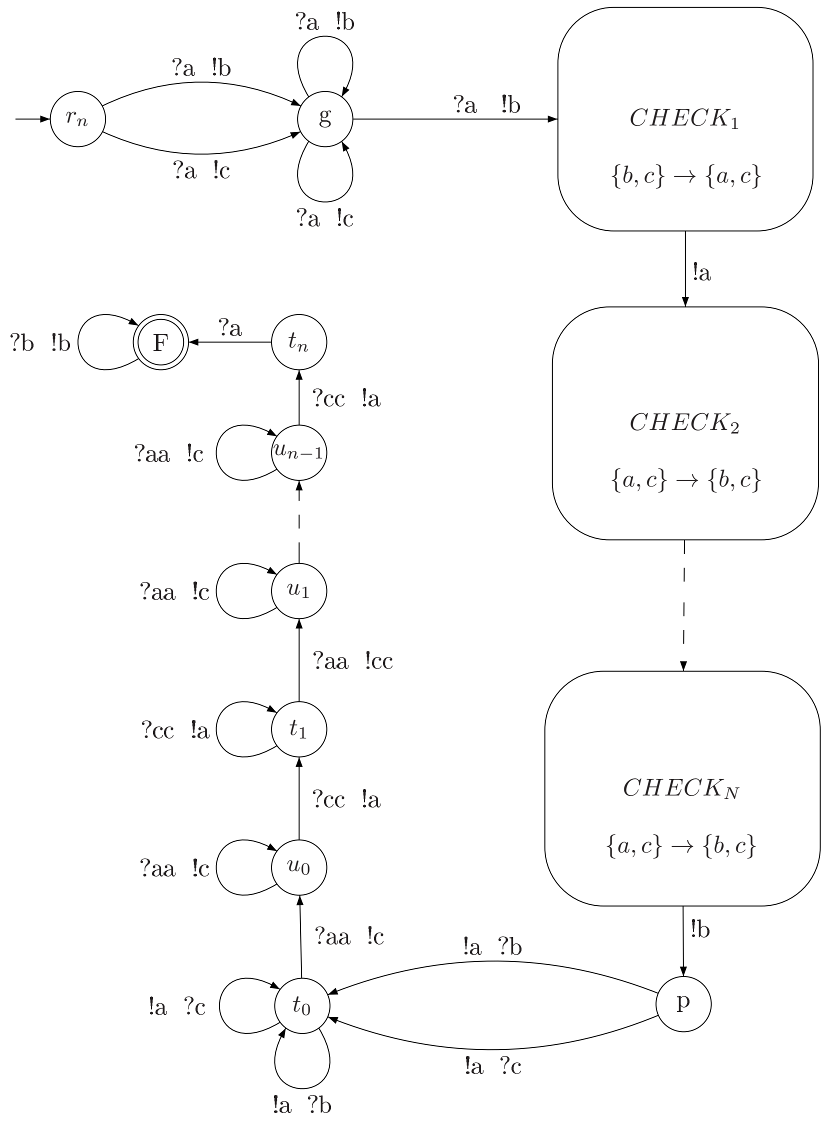
Hence when a run π reaches the state r_n, in a configuration (r_n, x) there are three possibilities :

perfect case : the run π was perfect and $x = a^{4^n} = a^{2^\beta}$

fat case : (the run wasn't perfect) and $x = a^\lambda$ with $\lambda > 2^\beta$ and we will in fact only remind that $|x| > 2^\beta$

ba $\sqsubseteq$ **case** : $ab \sqsubseteq x$ (note that the run until here can have been perfect).

Then the second and last part of $\mathcal{C}$ will be



Here's the philosophy. We described above how to encode a trace of a computation of $\mathcal{M}$ as a string of $\{0,1\}^{2^\beta}$. Here, a and b will alternatively play the role of 0, and c will always play the role of 1.
We Analise what can happen in the states r_n and g in function of the 3 cases described above :

perfect case : π was a perfect run to r_n and reached the configuration (r_n, a^{2^β}).
Then, the transitions of r_n and g (g for guessing) allows us to completely

cycle the channel and reach the part of the automaton $CHECK_1$ with a string x of $\{b,c\}^{2^\beta}$, the only constraint being that it ends with a $'b'$. We now see this string as a trace of the computation of $\mathcal{M}$ on m, b playing the role of 0, and c the role of 1. Hence, we can see that the fact that w ends with a b is not a problem, since we made the assumption that in our coding of $\Sigma \cup S \cup \{*\}$ by $\{0,1\}$, all the $\{0,1\}$-codes end with a 0, i.e. in the current context, with a b.
Hence, starting in the perfect case, we can either reach $CHECK_1$ with a string $y \in \{b,c\}^{2^\beta}$ ending with a b (which means that the simulation kept on being perfect), y is then a potential trace of $\mathcal{M}$.
Or there has been a non-perfect step and necessarily $|y| > 2^\beta$, since in this part of the automaton, every time we read something we afterward write something else.

fat case : for the same reason, if we reached r_n with a channel x , s.t. $|x| > 2^\beta$, then when reaching $CHECK_1$, with the channel y , we must have $|y| > 2^\beta$.

***ba* $\sqsubseteq$ case** : here, we arrived in r_n with a channel of the form $x_0.b.x_1.a.x_2$. Now since until $CHECK_1$ the only letter that can be read is a and since the last transition is necessarily $\xrightarrow{!b}$, we can claim, that, when reaching $CHECK_1$, the content of the channel y will necessarily contain the subword bab, and so also ab. We call this the ***ab* $\sqsubseteq$case**.

To summarize, when we reach $CHECK_1$ with the tape x , there are three possibilities :

$\{b,c\}$ perfect case : $x \in \{b,c\}^{2^\beta}$ and ends with a $'b'$. In that case we name $w \overset{def}{=} x$

fat case : $|x| > 2^\beta$

***ab* $\sqsubseteq$ case** : $ab \sqsubseteq x$

Now, if by induction we suppose that when we reach the part $CHECK_{2.i+1}$ of the automaton, we are in one of the three cases listed above : perfect, fat, or $ab \sqsubseteq$.
What $CHECK_{2.i+1}$ is going to do is checking one of the properties of the trace w listed above. In order to do that, we consider that the content of the tape is a string x of $\{b,c\}^{2^\beta}$, (true in the perfect case), and that the letter b (resp. c) plays the same role as 0 (resp. 1) in the way to encode a trace of $\mathcal{M}$. Then, in $CHECK_{2.i+1}$, the tape is going to be cycled, the automaton reads only b and c , and check the desired property on the associated word on $\{0,1\}$. But instead of making a real-cycle on the alphabet $\{b,c\}$, it changes the b in a and the c remains a c. That is to say that in $CHECK_{2.i+1}$, there are only transitions of the form
$\theta_1 \xrightarrow{?b} \theta_2 \xrightarrow{!a} \theta_3$ or
$\theta_1 \xrightarrow{?c} \theta_2 \xrightarrow{!c} \theta_3$.
We then are sure that there are no *good connected component*in $CHECK_{2.i+1}$. Hence if we analyze what happens when we'll leave $CHECK_{2.i+1}$ to enter in $CHECK_{2.(i+1)}$ in accordance with the three possibilities we had (by hypothesis) when we entered in $CHECK_{2.i+1}$, we have :

$\{b, c\}$ perfect case : then if we reach $CHECK_{2.(i+1)}$ having made a perfect run and a complete cycle of , the content of the tape y must be the same as the one when we enter x, if you replace the b by some a , and the c remain c, and moreover, we are sure that the property we wanted to check was true. Otherwise there are two possibilities. Either there has been one false-read, and in that case, we will enter the fat case anyway, or there hasn't been any false-read, but we left $CHECK_{2.i+1}$ before to have read all the b and c. Then we necessarily enter in the $ba \sqsubseteq$ case (here, the fact that in our coding of$\Sigma \cup Z \cup \{*\}$ on $\{0, 1\}$ we supposed that all the codes ended with a 0 has to be use).

fat case : since the length of the channel cannot decrease, we will reach $CHECK_{2.(i+1)}$ remaining in the fat-case.

$ab \sqsubseteq$ case : here we know that we won't be able to read the a in $CHECK_{2.i+1}$, and so neither the b , but we'll have to write an a , so we'll necessarily reach $CHECK_{2.(i+1)}$ in the $ab \sqsubseteq$ case.

Hence we can conclude that we'll reach $CHECK_{2.(i+1)}$ in one of these three possibilities :

$\{a, c\}$ perfect case

fat case

$ba \sqsubseteq$ case

In the same way, we can show that if we start in $CHECK_{2.i}$ in one of the three cases

$\{a, c\}$perfect case

fat case

$ba \sqsubseteq$ case

then we will necessarily reach $CHECK_{2.i+1}$ in of these cases

$\{b, c\}$ perfect case

fat case

$ab \sqsubseteq$ case

Which is then enough for an induction. And then, if we assume that N is odd, which is not a problem since we can anyway had an *empty-check*, we know that when we'll leave $CHECK_N$ and reach the state p, we'll be in one of these possibility's

$\{b, c\}$ perfect case : and m is accepted by $\mathcal{M}$

fat case

$ab \sqsubseteq$ case

Then when we'll leave the state t_0,say in the configuration (t_0, x), we'll have to be in one of these situations :

***'a'*-perfect case** : $x = a^{2^\beta}$.

fat case

***ba* $\sqsubseteq$ case**

Now we have

Lemma 3 *If* $(t_0, x) \xrightarrow{*} (F, y)$ *with* $y \in \{b\}^*$ *, then in fact* $y = \epsilon$ *and* $x \in \{a\}^{\leq 4^n}$

Proof : □

Hence, the only possibility to reach a good configuration in F , was that the run was perfect, guessed a trace of $\mathcal{M}$, and really cycled the tape during the various checks, and hence that the trace of $\mathcal{M}$ that had been guessed at the beginning of our path was correct. □

7 NP-completeness with a two-letter alphabet

Lemma 4 *Given two letters* a*,* b *(that might be equal) , an integer* i *in binary, a channel machine* $\mathcal{C}$ *, and two of its states* q *and* r*, define*

$$\mathcal{P}_{q,r} = \left\{ \begin{array}{l} \textit{paths } p \textit{ in } \mathcal{C} \textit{ starting in } q \textit{ and ending in } r \textit{ such that} \\ \textit{you only read some } a\textit{, but at least } i\textit{, and you only write some } b \end{array} \right\}$$

Then, we can calculate in polynomial time
$j \overset{def}{=} \min_{p \in \mathcal{P}_{q,r}} \{\textit{number of } b \textit{ that we write in } p\} \overset{def}{=} H_{a,b}(q, r, i)$.
Since this number could be not defined if no such path exists, i.e. $\mathcal{P}_{q,r} = \emptyset$*, we more precisely require that we could know in polynomial time if this number is defined, and in that case what's its value.*

Proof : We first remark that for $m, n > 0$, we've got

$$H_{a,b}(q, s, m + n) = \min_{r \in Q} H_{a,b}(q, r, m) + H_{a,b}(r, s, n) \tag{20}$$

Where we state that
$undefined + whatever = undefined$ and that
$min\{undefined, x\} = x$.
$undefined$, plays quite a similar role than ∞. Now

- for every q, r $H_{a.b}(q, r, 0)$ can be calculated in polynomial time. It suffices to find, if it exists, a path using only $read - a$ and $write - b$ edges, that contains the less number of $write - b$ edges.
- for every q, r, it's clear that $H_{a.b}(q, r, 1) \leq 2|\Delta|$ or is $undefined$. Then , for all the states, q, r we calculate successively their values :
  - $H_{a.b}(q, r, 1) = 0$ if and only if there exists a non-empty $read\ a$ path from q to r.

- For $n \geq 0$, $H_{a.b}(q,r,1) = n+1$ if and only if $H_{a.b}(q,r,1) \nleq n$ and if there exists a transition $s \xrightarrow{!a} t$ and $H_{a.b}(q,s,0) = k$ and $H_{a.b}(t,r,0) = l$ with $k+l = n+1$. Hence in $2|\Delta|$ steps, we can compute all the values of $H_{a.b}(q,r,1)$.

- Now that we've got the $H_{a.b}(q,r,1)$, for every q and r, we can easily construct by induction the values of $H_{a.b}(q,r,2^j)$ for $j \leq \log i$ and for every q and r, using (20).
- Finally, using once more (20) , we can construct the value of $H_{a.b}(q,r,i)$, using the binary decomposition of i.

All this construction requires well only a polynomial time.□

Lemma 5

$$\textit{If} \quad (q,w) \xrightarrow{\leq K\ cycles} (q',w') \tag{21}$$

then there exists $w'' \sqsubseteq w'$ *and a computation*

$$(q,w) \to (q',w'') \tag{22}$$

such that the length of the computation (22) *is less than* $(|w|+1)(|Q|+1)^K$.

Proof : We make an induction on K.

- if $K = 1$, then (21) is a computation with less than one cycle. So if we write $w = w_1.w_2...w_n$, the beginning of the computation (21) must look like

$$(q^1,w) \xrightarrow{!a_1} (q^2,w.a_1) \xrightarrow{!a_2} ... \xrightarrow{!a_l} (q^l,w.a_1.a_2...a_l) \xrightarrow{?w_1} (q^{l+1},w_2...w_n.a_1...a_l) \tag{23}$$

  Now, we can then find in $\mathcal{C}$ a subpath of (23) of length $k \leq |Q|$ starting from q^1 and arriving in q^{l+1}. This allows us to say that there exists a computation of length $k \leq |Q|$:
  $(q^1,w) \to (q^{l+1},w_2...w_n.u_1)$
  where $u_1 \sqsubseteq a_1...a_n$.

  Now if we repeat this for all the letters of w that must be read in (21) we can found a simulation of length $k \leq |Q| \times (|w|+1)$ that leads to the a configuration (q',w'') with $w'' \sqsubseteq w'$.

- if $K+1 > 1$ then we can decompose the computation (21) in two parts :
  $(q,w) \xrightarrow{K-1\ cycles} (r,v) \xrightarrow{1\ cycle} (q,w')$
  Then by induction we can found $v' \sqsubseteq v$ such that we have a computation $(q,w) \to (r,v')$ of length $\leq |w|(|Q|+1)^K$.
  Then since we have got $(r,v) \xrightarrow{1cycle} (q,w')$
  it is straightforward to check that we also have for a $w'' \sqsubseteq w'$
  $(r,v') \xrightarrow{1cycle} (q,w'')$
  and then using the the property we've showed for $K = 1$ we have that this computation has a smaller length than $|w||Q|(|Q|+1)^K$ and then finally

the computation
$(q, w) \to (q', w'')$
has a length
$k \leq |w|(|Q|+1)^K + |w||\Delta|(|Q|+1)^K = |w|(|Q|+1)^{K+1}$. $\square$

Theorem 3 *If $\mathcal{C}$ has only two letters, then the membership in INF is in NP.*

Proof : we suppose that the two letters of $\mathcal{C}$ are a and b.

First, given a configuration (q, w) , we know that it belongs to INF if and only if from this configuration we can reach a good connected component of $\mathcal{C}$ where everything we can write can be read, and moreover, we must join this component with a tape containing these good letters that we can read. With two letters, there are two possibilities :

- Either from q we can reach in $\mathcal{C}$ a *good connected component$G$* where we can read the both letters a and b. In that case, it really doesn't depend on the initial content of w, because, whatever the content of the tape w' is when we reach $\mathcal{C}'$, we'll be able to read every thing. Hence we just have to decide the problem of the reachability in a graph to tackle this case, which can be done in $PTIME$.
- Otherwise, the only possibility for (q, w) to be in INF is that we can reach a state q' of a *good connected component$G$*, where, for instance we can read a, and the content of the tape is a word $w' \in \{a\}^*$. That is to say we have the following computation :

$$(q, w) \to (q', w') \text{ with } w' \in \{a\}^* \tag{24}$$

Now, since we here make the assumption that we cannot reach a *good connected component*where we're able to read the both letters, and using theorem 1 , we can suppose that we have a subcomputation of (24) (where maybe we have to replace a by b) , with less than Q cycles. Then using lemma 5 , we can suppose that we can find a computation

$$(q, w) \xrightarrow{\leq |Q| cycles} (q', w'') \text{ with } w'' \sqsubseteq w' \in \{a\}^* \tag{25}$$

Where the number of steps of this computation (25) is less than $|w| \times (|\Delta|+1)^{|Q|}$, and using once more theorem 1, we can correctly suppose that (25) is a computation with less than $|Q|$ cycles.
Consider now the word x that we would obtain if would have never really read a letter in (25) , in other word we consider the initial content of the tape w and we add to it, on the right, all the letters that will be written during (25). We can then obtain an unique decomposition of x
$x = a^{i_1}.b^{j_1}.a^{i_2}...a^{i_k}.b^{j_k}.a^{j_{k+1}}$
with i_l and $j_l > 0$ except maybe for i_1 and j_{k+1} .
Then we must have $k \leq |Q|$. Indeed, otherwise $k > |Q|$, and since we know that all the b of x have been read in (25)(because $w'' \in \{a\}^*$), we can consider for each $1 \leq h \leq k$ the transition which read the last b of b^{j_h}
:
$r_i \xrightarrow{?b} r'_i$.

Then, using the pigeon-hole principle, we can find $m < n$ such that $r_m = r_n$. But then, looking at the subcomputation in (24) starting in r_m and ending in r_n, this allows us to consider a cycle and hence a connected component, in G where we can read b but also a because between r_m and r_n , in (24) , the a^{i_h} for $n < h \leq m$ will have been read. This is then a contradiction with the fact that we supposed we couldn't reach a *good connected component*where we could read a and b. Hence $k \leq |Q|$.
Since in (25) there were less than $|w|(|\Delta|+1)^{|Q|}$ steps, this means that $|x| \leq |w| + |w|(|\Delta|+1)^{|Q|}$ which is exponential in the length of our input, and, if you remind that $x = a^{i_1}.b^{j_1}.a^{i_2}...a^{i_k}.b^{j_k}.a^{j_{k+1}}$, then since every i_l and j_l are less than $|x|$, every i_l and j_l have a polynomial size, if written in binary. Hence, a way we are going to be sure that $(q, w) \in INF$ is by guessing the successive steps of a computation like (25).

For doing that, we guess the number $K \leq |Q|$ of cycles of the computation, and then the successive contents of the channel $c_1,\ c_2, ..., c_{K+1}$ after each cycles, which means that for instance $c_1 = w$ and that $w'' = c'_K.c_{K+1}$ where c'_K is a suffix of c_K. Since in (25) x represents the concatenation $c_1..c_{K+1}$, and that we've seen that the number of alternation of a and b in x was less than $|Q|$, we can guess the contents of the channels c_m as words of a length smaller than $|x|$, and of the form $a^{i_1}.b^{j_1}.a^{i_2}...a^{i_k}.b^{j_k}.a^{j_{k+1}}$ with $k \leq |Q|$. In fact, we guess in binary the numbers i_h and j_h, which then makes a linear guess of polynomial guesses. After that, we have to be sure that the computation we guessed was correct.

For instance, if

$$c_l = a^f.b^g.a^h$$
$$c_{l+1} = b^i.a^j$$

we could for instance guess that we started read the whole block a^f , starting in state q and finishing in state r, and that during that we wrote 'i'' b, with $i = i' + i''$ (that has to be guessed too).
Then we guessed $g = g' + g''$ and that starting in state r, and finishing in state s we read the g' first a of a^g, and wrote the i'' remaining b of b^i. Then, guessing that $j = j' + j''$, we guess that starting in state s and ending in state t we read the g'' last b of b^g , and wrote j' a, and finally, that starting in state t and finishing in state u, we read a^h, and wrote $a^{j''}$.

In other words, what we did is just guessing this computation :

$$(q, a^f.b^{g'+g''}.a^h) \xrightarrow{*} (r, b^{g'+g''}.a^h.b^{i'}) \xrightarrow{*}$$
$$(s, b^{g''}.a^h.b^{i'+i''}) \xrightarrow{*} (t, a^h.bi' + i''.a^{j'}) \xrightarrow{*} (u, b^{i'+i''}.a^{j'+j''})$$

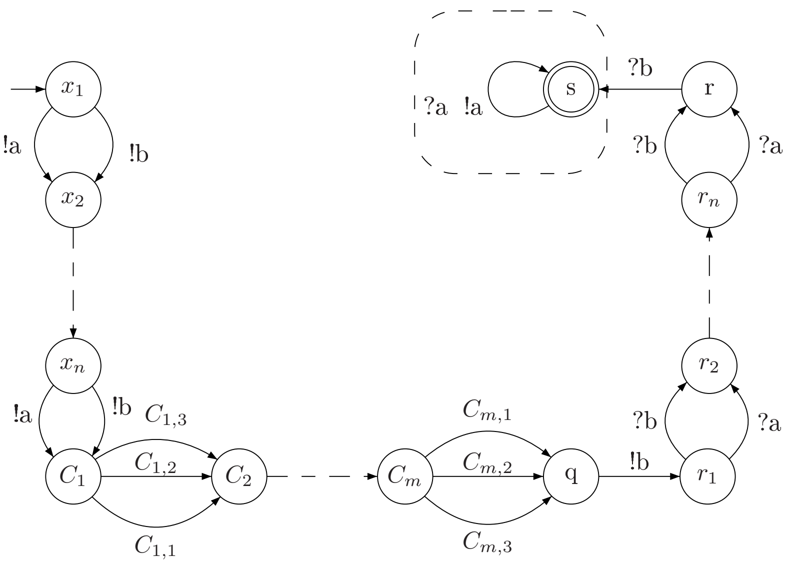
Hence, the only thing that remains to be checked, is that every time we supposed we could start in a state q, read i' and write j $'b'$ and end in state r, it was true. Instead of that, we just check that $H_{a,b}(q, r, i) \leq j$, and lemma 4 ensures that we can do it in polynomial time. Because indeed, if we check that we have $H_{a,b}(q, r, i) \leq j$, then using the fact that we've got a downward transition system, we can make an induction that finally tells us that we had a path
$(q, w) \to (q', w''')$ with $w''' \sqsubseteq w''$, and hence $w''' \in \{a\}^*$. $\square$

Theorem 4 *If $\mathcal{C}$ has only two letters, then the membership in INF is $NP-hard$.*

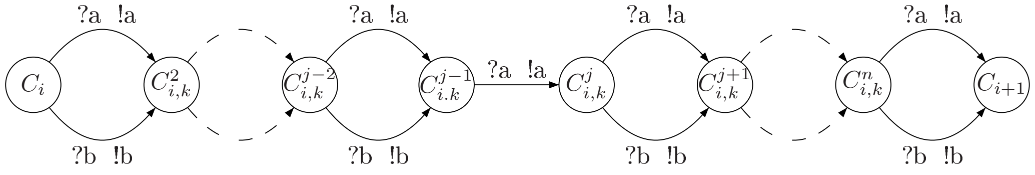
Proof : We reduce $3-SAT$. Let's then consider an instance ($\mathcal{P}$) of $3-SAT$ on n variables $x_1,\ x_2, ...x_n$ with m clauses.

$$\bigwedge_{i=1..m} C_i \text{ with } C_i = y_1^i \vee y_2^i \vee y_3^i \tag{26}$$

We assume then that each clause contains exactly 3 literals, which is not a restriction for the $NP-hardness$ of $3-SAT$. Then consider the following channel automata $\mathcal{C}$



What we mean by the transition $C_{i,k}$, for $1 \leq i \leq m$ and $k = 1, 2$ or 3 is the following. If $y_i^k = x_j$, then $C_{i,k}$ is of the form :



Note that if we had had $y_i^k = \neg x_j$, the block $c_{i,k}$ of $\mathcal{C}$ would have been the same except that there would have been
$C_{i,k}^{j-1} \xrightarrow{?b\ !b} C_{i,k}^j$
We claim that $(g_1, \epsilon) \in INF$ if and only if our instance of $3-SAT$ is satisfiable. The idea is that first, the only *good connected component* in the dashed part around the state s. Hence, $(g_1, \epsilon) \in INF$ if and only if it is possible to find a path from (g_1, ϵ) to (s, w) with $w \in \{c\}^*$.

We explain what is the idea of the coding, and at the same time prove that if $\mathcal{P}$ is satisfiable, say with a valuation V, then $(g_1, \epsilon) \in INF$. In order to do that we encode the valuation V of the variables $x_1, ..., x_n$ as a string $w_V = w_1...w_n$ in $\{a, b\}^n$, saying that x_i is true if and only if $w_i = a$.

Then, the first part of a path from (g_1, ϵ) to C_1 is just a guess of length n, and then we can reach C_1 with the content of the tape w_V. Then, if we suppose the computation perfect, i.e. every read transition really removes a letter of the tape, during the computation, the tape will always contain a n-letter word. Then, for each clause $C_i = y_1^i \vee y_1^i \vee y_3^i$, the passage from the state C_i to C_{i+1} just cycles the tape, and check that the literal associated with the path which has been chosen effectively valid our instance of $3 - SAT$, accordingly to w. Finally, the passage

$$q \xrightarrow{!b} r_1 \tag{27}$$

just adds a b on the tape, and then, the sequence $r_1 \rightarrow .. \rightarrow r$ reads the letters that correspond to w, and finally the transition $r \xrightarrow{?b} s$ reads the b we had written in (27). So we arrive in the configuration (s, ϵ) , which is a good configuration.

Now, if conversely $(x_1, \epsilon) \in INF$, let's consider a path π which leads to a good configuration, that is to say here, a configuration of the form (s, a^λ) with some $\lambda \in \mathbb{N}$.
Clearly, π must first pass through the states $g_1, ..g_n$ and then reach C_1 with a content of the tape, say w, which contains n letters. As above, we interpret this word as a valuation V of the variables x_i. Now, we prove that the part of the computation of π which corresponds to the passage from the state C_1 to q is perfect. Indeed, whatever this passage is, it must be a sequence of transitions of the form
$q \xrightarrow{?\sigma\ !\sigma} q'$.
Hence suppose the simulation hasn't been perfect, then the length of the content of the tape, say w' must be $> n$ when reaching the state q.
Afterward, we must have the transition $(q, w') \xrightarrow{!b} (r, w'.b)$, and after having passed through the states $r_1, r_2, ..., r_n$ we must reach the state r after having read n letters, if the computation was perfect, less otherwise, and then in any case, we are in the configuration $(r, w''.b)$, where w'' is a proper suffix of w' (since we supposed $|w'| > n$). Hence the transition $r \xrightarrow{?b} s$ won't allow us to read the last b of $w''.b$, and then π should reach a configuration $(s, w'''.b)$. But from s, it won't be possible to join another state, and it won't be either possible to read this b, which is then a contradiction.
Hence, necessarily, the part of the computation that crossed the states $C_1, .., C_m, q$ had to be perfect, which ensures the valuation V associated to w satisfied $\mathcal{P}$. $\Box$

8 Counter machines

We now consider insertion counter machines. This can been seen as an insertion channel machine with few channels, using only a single letter alphabet. We then fix a counter machine $\mathcal{C} = (Q, \Delta, c_1, \ldots, c_n)$, with states Q, transitions Δ and n counters. The notion of fairness is here irrelevant, since we have only one letter.

However, we can consider the problem of an infinite computation with some set constraints, that is to say, for each state $q \in Q$, we associate a lower-set of $\mathbb{N}^n$ L_q closed by $\leq$, that is to say, if $x \in L_q$ and $y \leq x$ then $y \in L_q$. According to these constraints, we define a correct configuration to be a configuration $\gamma = (q, c_1, \ldots, c_n)$ such that $(c_1, \ldots, c_n) \in L_q$. Then, INF, which we define as the set of all the configurations γ such that it is possible to start an infinite computation with only correct configurations is a lower-set, and computable, using the same methods as the ones we used to calculate INF for the $CAROTS$.

Proposition 12 *Giving an insertion counter machine $\mathcal{C}$, some constraints L_q, given with a cobasis in binary, and a configuration γ, we can decide in* EXPSPACE *if $\gamma \in INF$*

The cobasis which describes the L_q are giving say with elements of $\mathbb{N}^n$ of the form $(b_1, \ldots, b_\alpha)$. We define K to be the maximum of all the x_i that appears among all the elements of the basis of all the L_q. According to this we define $\phi_K(n)$ to be the length k of the longest computation $\pi = \gamma_0 \to \ldots \to \gamma_i \to \ldots \to \gamma_k$ such that $\gamma_0 \notin INF$, for a counter machine with the K as defined just above. We then are going to control $\phi_K(n)$ by induction on n

For $n = 1$, it is straightforward to see that the number of correct configurations is less than $|Q|.K$, so if we consider a correct computation

$$(q_0, c) \to \ldots \to (q_{|Q|.K})$$

then using the pigeon hole principle, we can find two identical configurations, and then $(q_0, c) \in INF$. So $\phi_K(n) \leq |Q|.K$.

If we have $n + 1$ channels, and the constraints $L_1, \ldots, L_{n+1}$, let's define $N = \phi_K(n) + K$, and let's show that

$$\phi_K(n+1) \leq |Q|.N^{n+1} = |Q|.(\phi_K(n) + K)^{n+1}. \tag{28}$$

Let's consider a correct computation

$$\pi = \gamma_0 \to \ldots \to \gamma_{|Q|.N^{n+1}}$$

Then, there exists a channel, say c_{n+1}, and an i such that the content of c_{n+1} in γ_i is greater than N (otherwise, using the pigeon hole principle, we could still find two similar configurations in π, and then would be able to find an infinite correct computation, but we implicitly supposed that $\gamma_0 \notin INF$.) Then around the index i, we can find a subcomputation of π :

$$\rho \;:\; \gamma_j \to \ldots \to \gamma_{j+N}.$$

and since we defined $N = \phi(n) + K$, and that in every transition a counter value can at most decrease by one, we know that the counter value of c_{n+1} in this subcomputation will always be greater than K. Hence during this path :

$$\rho \;: \gamma_j \to \ldots \to \gamma_{j+N}$$

for each $L_q = cobasis(b_1, \ldots, b_\alpha)$ (the b_k are here the elements -of $\mathbb{N}^{n+1}$- of the cobasis of L_q, and we'll refer as $b_1 = (b_1(1), \ldots, b_1(n+1))$ for instance), we

have for any $j \leq m \leq j+N$
if $\gamma_m = (q, c_1, \dots, c_{n+1})$

$$\begin{array}{ccc}
(c_1,\dots,c_{n+1}) \not\geq b_1 & \textit{but} & c_{n+1} > b_1(n+1) \\
(c_1,\dots,c_{n+1}) \not\geq b_2 & \textit{but} & c_{n+1} > b_2(n+1) \\
\dots & \dots & \dots \\
(c_1,\dots,c_{n+1}) \not\geq b_\alpha & \textit{but} & c_{n+1} > b_\alpha(n+1)
\end{array}$$

and so,

$$\begin{array}{c}
(c_1,\dots,c_n) \not\geq (b_1(1),\dots,b_1(n)) \\
(c_1,\dots,c_n) \not\geq (b_2(1),\dots,b_2(n)) \\
\dots \\
(c_1,\dots,c_n) \not\geq (b_\alpha(1),\dots,b_\alpha(n))
\end{array}$$

Now, if we define $\mathcal{C}' = (Q, \Delta', c_1, \dots, c_n)$, and the L'_q to be the projection of the n first coordinates of L_q on $\mathbb{N}^n$, and define Δ' to be the same as Δ, except that when there is a transition
$q \xrightarrow{!c_{n+1}}$ or $q \xrightarrow{?c_{n+1}}$ in Δ, we just consider in Δ' an ϵ-transition $q \to r$, i.e. a transition without any action. Then, it is a routine to check that the projection (we consider that the projection of $(q, c_1, \dots, c_{n+1})$ is $(q, c_1, \dots, c_n)$) of ρ , ρ' is a correct computation of $\mathcal{C}'$, and then by induction hypothesis, since the length of this computation is $\phi_K(n)$, we can find an infinite correct computation in $\mathcal{C}'$, τ, starting in the configuration that is the projection of γ_i. This also allows us to find an infinite correct computation in $\mathcal{C}'$ starting in the projection of γ_0, since we can reach correctly γ_i from γ_0 in $\mathcal{C}$, so we can also reach the projection of γ_i from the projection of γ_0 in $\mathcal{C}'$. But clearly, an infinite correct computation in $\mathcal{C}$ gives rise to another one in $\mathcal{C}$. So we have proved (28).

Now, if we keep in mind that

$$\begin{array}{rcl}
\phi_K(1) & \leq & |Q|.K \\
\phi_K(n+1) & \leq & |Q|.(2\phi_K(n))^{n+1}
\end{array}$$

because $\phi_K(n) \geq K$ we have

$$\begin{array}{rcl}
\phi_K(n+1) & \leq & |Q|(2\phi_K(n))^{n+1} \\
& \leq & |Q|(2|Q|(2\phi_K(n-1))^n)^{n+1} \\
& \leq & |Q|.(2|Q|)^{n+1}.(2\phi_K(n-1))^n)^{n+1} \\
& \leq & \dots \\
& \leq & |Q|.(2|Q|)^{n+1} \dots (2|Q|)^{(n+1).n\dots2.1}.\phi_K(1)^{(n+1).n\dots2.1} \\
& = & |Q|.(2|Q|)^{(n+1)!}.(|Q|.K)^{(n+1)!}
\end{array}$$

Which is doubly-exponential. So $\gamma \in INF$ if and only if a computation starting in γ of this length can be made. But, this every steps increases a counter by at most one, along such a computation, the size of the computation, stored in binary will be exponential. We can then conclude using that PSPACE=NPSPACE. □

We don't know if this result is optimal. Maybe the hardness results of [CLM76] could be applied, but anyway, we didn't see how to use them directly.

References

[BMOW07] Patricia Bouyer, Nicolas Markey, Joel Ouaknine, and James Worrell. The cost of punctuality. In *LICS '07: Proceedings of the 22nd*

Annual IEEE Symposium on Logic in Computer Science, pages 109–120, Washington, DC, USA, 2007. IEEE Computer Society.

[CLM76] E. Cardoza, Richard J. Lipton, and Albert R. Meyer. Exponential space complete problems for petri nets and commutative semigroups: Preliminary report. In *STOC*, pages 50–54, 1976.

[FS01] Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1-2):63–92, April 2001.

[OW05] Joël Ouaknine and James Worrell. On the decidability of metric temporal logic. In *LICS*, pages 188–197, 2005.

[OW06] Joël Ouaknine and James Worrell. Safety metric temporal logic is fully decidable. In *TACAS*, pages 411–425, 2006.

[Sch02] Philippe Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity. *Information Processing Letters*, 83(5):251–261, September 2002.