Février 98

Une preuve formelle du Bounded Retransmission Protocol dans le π-calcul

Brahim Mammass
LIP6, équipe SPI
Université Pierre et Marie Curie

Table des matières

Table des figures

1 Introduction

Développer des protocoles de communication, comme développer des systèmes complexes, est une tâche difficile [Hol91]. Pendant la conception de ces systèmes, des modèles sont souvent développés, par exemple sous forme de prototypes, et expérimentés par des tests ou des simulations. Les résultats obtenus permettent de déduire les propriétés du système qui sont souvent exprimées sous forme d'interactions avec l'environnement. Cependant, construire des modèles qui couvrent toutes les interactions possibles est un vrai challenge. Dans la plupart des cas, seule une partie des situations possibles est couverte.

La plupart des protocoles de communication sont encore décrits de manière informelle; plusieurs aspects de ces protocoles ne sont pas expliqués de façon précise et suffisamment abstraite. Le besoin de définitons formelles rigoureuses pour décrire les protocoles de communication s'exprime de plus en plus. L'avantage de cette approche est que les modèles formels sont plus abstraits que les programmes, les détails non importants pour les propriétés étudiées peuvent être facilement ignorés.

Le choix d'un formalisme dépend des systèmes modélisés et du type des propriétés étudiées. Les algèbres de processus sont convenables pour modéliser les interactions dans les systèmes coopératifs tels que les réseaux d'ordinateurs ou de télécommunication. Hormis l'analyse rigoureuse, les spécifications algébriques ont l'avantage de ne pas être ambigues. Cette propriété est très importante lors de la conception de systèmes composés de parties autonomes interagissant entre elles. Par exemple, le BRP (Bounded Retransmission Protocol), protocole de transfert de fichiers de Philips, se compose d'un émetteur et d'un récepteur. Ces deux composantes doivent interagir convenablement entre elles et avec l'environnement. Pour cela, les règles d'interaction doivent être spécifiées de manière non ambigue et doivent être respectées.

Dans ce papier, l'algèbre de processus utilisée est le π-calcul [MPW89a,MPW89b]. Ses constructeurs permettent de décrire le comportement des entités de plusieurs protocoles et les services attendus de ces protocoles. La vérification formelle se fait en prouvant que la composition parallèle des différentes entités est équivalente au service attendu du protocole.

La section 2 présente une introduction au π-calcul. La section suivante donne une définition informelle du BRP. Dans la section 5, une vue abstraite (spécification) du BRP dans le π-calcul est fournie. La section suivante donne une description formelle (implantation) du BRP dans le π-calcul. Une exécution du protocole pour transférer un fichier de trois messages est décrite afin d'illustrer le fait que lorsqu'un transfert se termine, le protocole revient à son état initial. La section 6 présente la théorie algébrique du π-calcul qui est utilisée pour prouver que la vue abstraite du protocole et son implantation sont équivalentes par bisimulation. La section 7 présente la méthode de preuve ainsi que la preuve complète de l'équivalence entre le protocole et sa spécification. Enfin, la dernière section présente quelques travaux réalisés autour du BRP et fait une comparaison du travail effectué dans ce papier avec ces différents travaux; elle fournit également les perspectives envisagées à la suite de ce travail.

2 Le π-calcul

2.1 Le π-calcul monoadique

2.1.1 Les processus mobiles

Le π-calcul permet de décrire des processus communiquants ayant une structure d'interconnexion qui évolue pendant l'exécution. Il étend CCS [Mil89] par le fait que les processus peuvent communiquer entre eux non seulement des valeurs mais également des noms de canaux.

Comme exemple, considérons trois processus P, Q et R. Supposons que P possède les canaux $\overline{a}$ et $\overline{b}$, Q possède le canal a et R possède le canal b (la barre sur un nom de canal indique que ce canal est utilisé en sortie; tandis qu'un canal non barré est utilisé en entrée). Dans la composition parallèle $(P \mid Q \mid R)$, les communications sont possibles entre P et Q (via le canal a) et entre P et R (via le canal b). Mais, aucune communication n'est possible entre Q et R. Cette situation est illustrée par la partie gauche de la figure 1.

Dans CCS, cette structure d'interconnexion entre les processus est statique dans le sens où aucune modification de la configuration du système n'est possible (i.e, aucun processus ne peut acquérir un nouveau canal suite à une communication).

En revanche, dans le π-calcul, le processus P peut transmettre son canal b à Q le long du canal a. Par exemple, si $P = \overline{a}b.P'$ (i.e, P envoie le nom du canal b sur le canal a à Q) et $Q = a(x).\overline{x}5.Q'$ (i.e, Q attend une donnée sur le canal a et désigne cette donnée par le paramètre x), alors au moment de la communication entre P et Q le long du canal a, x est lié à b. Après la communication, on obtient le processus $(P' \mid \overline{b}5.Q' \mid R)$ où Q a évolué en $\overline{b}5.Q'$; ce processus est prêt à envoyer la valeur 5 à R sur le canal b.

$P \mid Q \mid R \;\rightarrow\; P' \mid \overline{b}5.Q' \mid R$

Si on suppose que P' n'utilise plus le canal b, alors le système résultant est illustré par la partie droite de la figure 1. Cette nouvelle configuration peut être interprétée comme le déplacement de R du champ de communication de P à celui de Q. Ainsi, on peut dire que le π-calcul est dédié à la description de processus mobiles.

2.1.2 Syntaxe et sémantique informelle

Soient $x, y, z, u, v, \ldots \in N$ (ensemble des noms). Soient $A, B, \ldots$ des processus qui peuvent être les suivants:

- 0, un processus qui ne fait rien.
- $\overline{y}x.P$, un processus qui envoie le nom x sur le canal y, ensuite se comporte comme P. $\overline{y}x$ est appelé *préfixe négatif*.

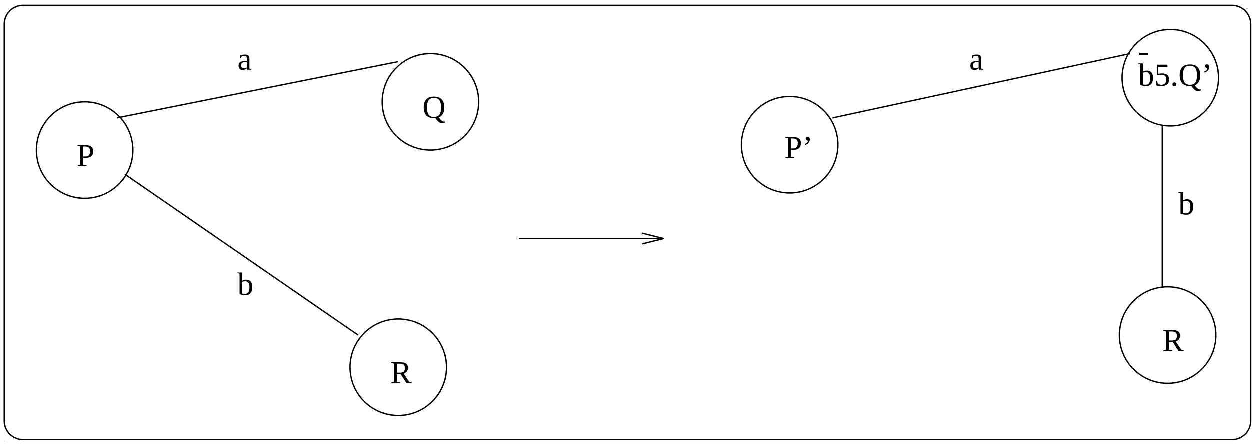

FIG. 1 - *La mobilité dans le π-calcul*

- $y(x).P$, un processus qui reçoit un nom sur le canal y, ensuite se comporte comme P dans lequel x est remplacé par le nom reçu. $y(x)$ est appelé *préfixe positif*.

- $\tau.P$, un processus qui fait une transition silencieuse τ et se comporte ensuite comme P.

- $P + Q$, un processus qui se comporte comme P ou comme Q.

- $P \mid Q$, un processus représentant la composition parallèle de P et Q. Les processus P et Q peuvent évoluer séparement. De plus, les communications entre P et Q peuvent se produire si l'un des processus émet sur un canal et l'autre reçoit sur le même canal. Ces communications se traduisent par des transitions silencieuses τ.

- $(\nu\ x)P$, un processus qui agit comme P mais le nom x est privé à P; x ne peut être utilisé comme canal dans les communications avec l'environnement du processus (i.e, avec les autres processus). x est donc lié dans P (i.e, x est un nom local dont la portée est P).

- $[x = y]P$, si x et y sont identiques, le processus se comporte comme P, sinon il ne fait rien.

- $A(y_1, ..., y_n)$ est un processus si A est un identificateur de processus d'arité n défini par une équation de la forme $P = A(x_1, ..., x_n)$, où les noms x_1, ..., x_n sont des noms distincts et sont les seuls noms libres dans P. Le processus $A(y_1, ..., y_n)$ se comporte comme P dans lequel chaque x_i a été substitué par y_i ($i = 1...n$). Les x_i peuvent être considérés comme les paramètres formels de A; tandis que les y_i sont les paramètres d'appel dans $A(y_1, ..., y_n)$.

  Ainsi, il est possible de coder une fonction ayant un nombre fini de paramètres par un processus dans le π-calcul. De plus, les identificateurs de processus fournissent la récursion car l'équation de définition de A peut contenir A.

Dans les préfixes $\overline{y}x$ et $y(x)$, on dit que y est le *sujet* et que x est *l'objet* ou le paramètre. Dans ce qui suit, on écrira $(\nu\ x_1...x_n)P$ au lieu de $(\nu\ x_1)...(\nu\ x_n)P$.

La sémantique opérationnelle formelle des processus est définie dans [MPW89b].

2.1.3 Portée des noms

On note par $n(P)$ l'ensemble des noms apparaissant dans le processus P.

Les noms *liés* dans un processus P notés $bn(P)$ sont les noms liés par un input ou par l'opérateur de restriction. Ces noms ne sont pas importants pour son environnement, ils traduisent uniquement les calculs internes.

On écrit $P \equiv Q$ pour dire que P et Q sont alpha-équivalents (i.e, ils diffèrent seulement par les noms liés). Dans la suite de ce papier, aucune distinction ne sera faite entre les processus alpha-équivalents.

Les noms *libres* dans un processus P notés $fn(P)$ sont les noms qui ne sont pas liés dans P. Ces noms représentent sa connaissance de l'envrionnement ou sa liaison avec les autres processus de son environnement. Réciproquement, ils représentent ce que l'environnement sait sur ce processus.

On dit qu'une *action est libre* si tous les noms apparaissant dans cette action sont libres. Par exemple, l'action $\overline{x}y$ est une action libre.

On note par $P\{y/x\}$ le résultat de la substitution de toutes les occurrences libres de x par y dans P, avec un renommage des noms liés si nécessaire pour éviter la capture de y (i.e, la liaison de y) dans P.

Dans l'exemple qui suit, y est lié dans $(\nu\ y)\overline{x}y.0$ mais y est libre dans $y(x).\overline{x}y.0$. Par contre, x est libre dans $(\nu\ y)\overline{x}y.0$ mais x est lié dans $y(x).\overline{x}y.0$. Ainsi:
$((\nu\ y)\overline{x}y.0 + y(x).\overline{x}y.0)\{y/x\} \equiv (\nu\ y')\overline{y}y'.0 + y(x).\overline{x}y.0$, où y' est un nom différent de y.

Considérons le processus $(\overline{x}y.P \mid (\nu\ x)x(z).Q)$. Le nom x apparait dans les 2 parties de la composition parallèle, mais ne représente aucune liaison entre $\overline{x}y.P$ et $(\nu\ x)x(z).Q$ car x est libre dans $\overline{x}y.P$ mais lié dans $(\nu\ x)x(z).Q$. En particulier, ces processus ne peuvent communiquer le long du canal x.

Même si le nom x est transmis comme objet à un processus possédant lui même un nom local x (phénomène appelé *scope intrusion*), les deux x demeurent distincts comme le montre la transition suivante:

$$\overline{y}x.P \ \mid \ y(z).(\nu\ x)Q \ \rightarrow \ P \mid (\nu\ x')Q\{x'/x\}\{x/z\}$$

Le renommage du x local par x' est une conséquence de la définition de la substitution; les noms liés sont renommés si nécessaire afin d'éviter les captures.

La seule situation où l'environnement de $(\nu\ x)P$ peut connaître le nom local x est lorsque P émet x comme objet dans une communication à l'environnement. La portée de x croît

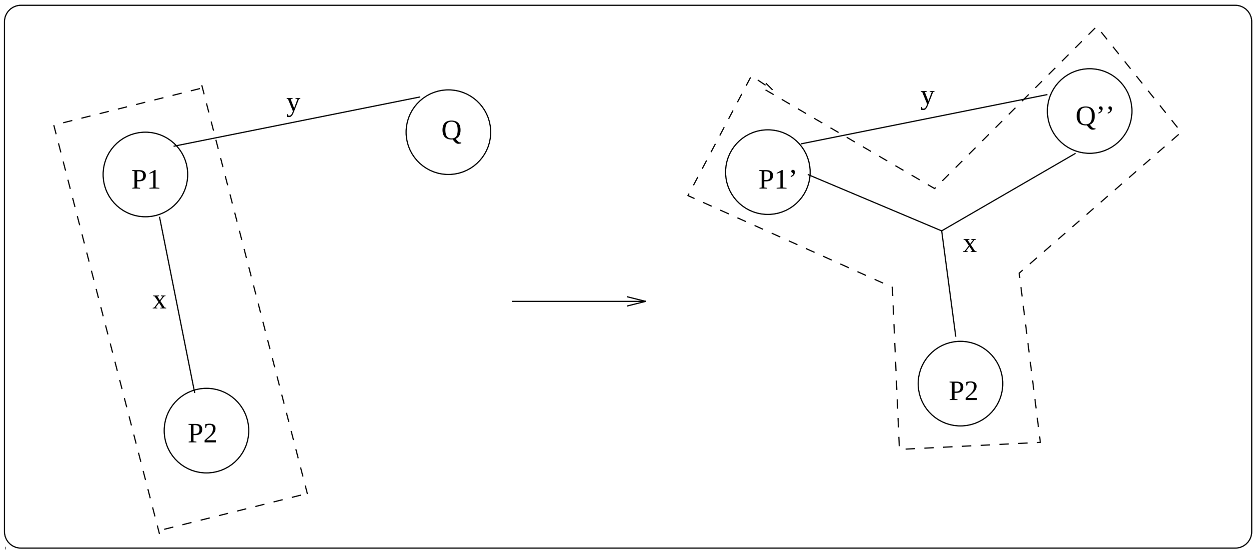

FIG. 2 - *Scope extrusion*

alors en s'étendant au récepteur de x (mais pas aux autres processus). Ce phénomène est appelé *scope extrusion* et est illustré par l'exemple suivant.

Soient $P1$ et $P2$ deux processus utilisant le nom x (x est libre dans $P1$ et dans $P2$). Le nom x dans $P = (\nu\ x)(P1 \mid P2)$ représente une liaison privée entre $P1$ et $P2$. Le nom x n'est pas libre dans P. Soit Q un autre processus. Supposons que $P1$ et Q communiquent via un canal y. Cette configuration est illustrée par la partie gauche de la figure 2.

Supposons que $P1 = \overline{y}x.P1'$ et $Q = y(z).Q'$.
Le canal x est donc transmis à Q le long du canal y par $P1$. Or, comme x est local à P, sa portée s'étend à Q. Cette nouvelle configuration est illustrée par la partie droite de la figure 2 où Q'' représente $Q'\{x/z\}$.

$(\nu\ \ x)(P1 \mid P2) \mid Q\ \rightarrow\ (\nu\ \ x)(P1' \mid P2 \mid Q'\{x/z\})$

De plus, si $P1'$ n'utilise plus x (i.e, x n'est plus libre dans $P1'$), on peut utiliser la loi suivante qui dit que si x n'est pas libre dans P alors $(\nu\ x)(P \mid Q) = P \mid (\nu\ x)Q$.
Donc, $(\nu\ x)(P1' \mid P2 \mid Q'\{x/z\}) = P1' \mid (\nu\ x)(P2 \mid Q'\{x/z\})$. Cette nouvelle configuration est illustrée par la figure 3 où Q'' représente $Q'\{x/z\}$. On peut dire ici que la portée de x a migré de $P1$ à Q.

2.2 Le π-calcul polyadique

Si on veut coder un ensemble fini de valeurs dans le π-calcul monoadique, il suffit de les coder par un ensemble fini de noms appelés constantes. Une *constante* est un nom qui ne

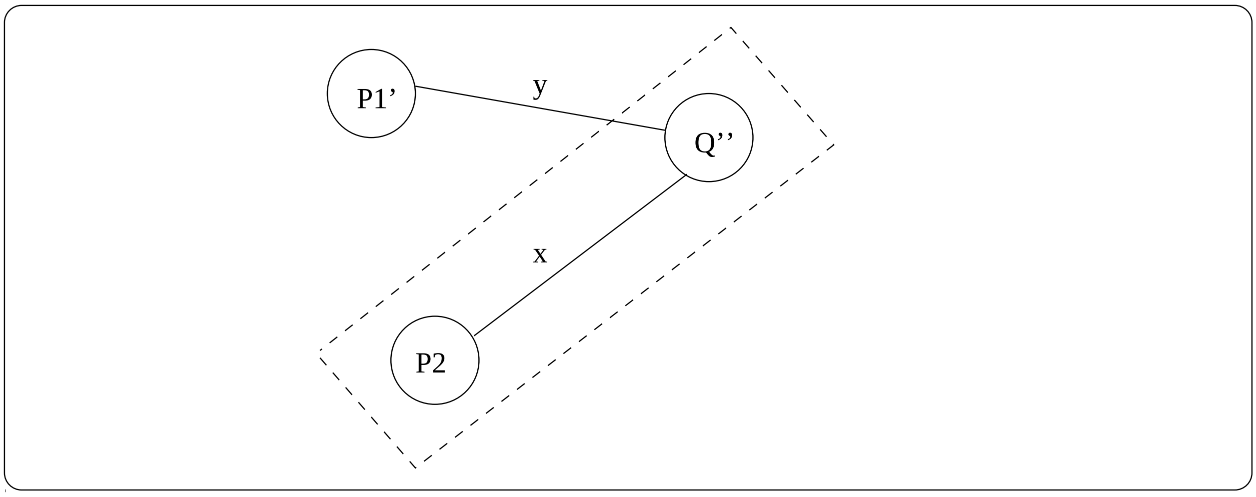

FIG. 3 - *Migration de la portée d'un canal*

peut être lié dans un input ou par l'opérateur de restriction. Par exemple, les constantes t et f peuvent représenter les valeurs booléennes $True$ et $False$. Mais, cette méthode ne peut être appliquée à un ensemble infini de valeurs. Toutefois, on peut coder un ensemble fini ou infini de valeurs sans utiliser des noms de constantes dans le π-calcul polyadique. C'est cette dernière approche qui est adoptée dans la suite pour coder quelques types usuels ainsi que certaines fonctions sur ces types nécessaires pour la modélisation du protocole.

2.2.1 Syntaxe et sémantique informelle

Dans le π-calcul monoadique, les processus peuvent communiquer un seul message à la fois; tandis que dans le π-calcul polyadique [Mil91], les processus peuvent émettre ou recevoir plusieurs messages en même temps sur le même canal. On introduit l'abbréviation suivante:
$\widetilde{y} = y_1\ y_2\ ...\ y_n$ ($\widetilde{y}$ symbolise le n-uplet $y_1\ y_2\ ...\ y_n$).
Maintenant, une action atomique peut être:

- $x(\widetilde{y})$, réception d'un n-uplet $\widetilde{y}$ sur le canal x. Le processus $x(\widetilde{y}).P$ reçoit un n-uplet $\widetilde{z}$ sur le canal x; ensuite se comporte comme P dans lequel chaque occurrence de y_i est substituée par z_i.

- $\overline{x}\widetilde{y}$, émission d'un n-uplet $\widetilde{y}$ sur le canal x. Le processus $\overline{x}\widetilde{y}.P$ envoie le n-uplet $\widetilde{y}$ sur le canal x; ensuite se comporte comme P.

- τ, une transition silencieuse.

On introduit un nouveau préfixe:
$\overline{x}(\widetilde{y}).P = (\nu\ \widetilde{y})\overline{x}\widetilde{y}.P$ (on introduit un nouveau n-uplet $\widetilde{y}$ et on émet $\widetilde{y}$ sur le canal x).

2.2.2 Codage des booléens

Le codage est fait de manière analogue au codage des booléens dans le λ-calcul où $True$ est codée par la fonction $\lambda x\ y.x$ et $False$ est codée par la fonction $\lambda x\ y.y$.

- La valeur $True$ est codée par le processus $[\![True]\!](b_t)$ qui attend deux noms de canaux t et f sur le canal de nom b_t, émet un signal sur le premier canal t (ci-dessous, $\overline{t}$ désigne l'émission d'un signal sur le canal t, c'est une émission sans paramètre) et revient à son état initial. On dira que la valeur booléenne $True$ est pointée par le canal de nom b_t.

  $[\![True]\!](b_t) \equiv b_t(t\ f).\overline{t}.[\![True]\!](b_t)$

  On peut faire l'analogie avec la fonction $\lambda x\ y.x$ qui appliquée à 2 valeurs a et b $((\lambda x\ y.x)a\ b)$ rend la valeur a. L'application dans le π-calcul est la composition parallèle de processus.

- La valeur $False$ est codée par le processus $[\![False]\!](b_f)$ qui attend deux noms de canaux t et f sur le canal de nom b_f, émet un signal sur le deuxième canal f et revient à son état initial. De même, on dira que la valeur booléenne $False$ est pointée par le canal de nom b_f.

  $[\![False]\!](b_f) \equiv b_f(t\ f).\overline{f}.[\![False]\!](b_f)$

Maintenant, on peut coder des fonctions sur les booléens.

- La fonction *copy_bool* ci-dessous copie la valeur booléenne pointée par un canal y (i.e, $True$ ou $False$) dans un autre canal z. Elle introduit deux nouveaux noms u et v et envoie ces deux noms sur le canal y. Si elle reçoit un signal sur le canal u, elle crée un processus $[\![True]\!](z)$. Sinon, elle crée un processus $[\![False]\!](z)$. L'envoi des deux canaux u et v sur le canal y et la réception d'un signal sur l'un de ces deux canaux correspond à l'évaluation de y.

  $$copy\_bool(y,z)\ =\ (\nu\ \ u\ \ v)\ \ \overline{y}\ \ u\ \ v.(u().[\![True]\!](z)\ +\ v().[\![False]\!](z))$$

  Supposons que le canal y pointe la valeur $True$, on a alors un processus de la forme $y(t\ f).\overline{t}$. Si on veut copier la valeur pointée par y dans un autre canal z, on doit composer ce processus avec le processus correspondant à la fonction *copy_bool* appliquée aux canaux y et z. On obtient alors:

  $[\![True]\!](y) \mid copy\_bool(y,z)$

  $\equiv y(t\ f).\overline{t}.[\![True]\!](y) \mid (\nu\ u\ v)\ \overline{y}\ u\ v.(u().[\![True]\!](z) + v().[\![False]\!](z))$

  Les noms u et v ne sont pas libres dans la partie gauche de la composition parallèle. On peut alors étendre leur portée à la composition parallèle.

  $\equiv (\nu\ u\ v)\ (y(t\ f).\overline{t}.[\![True]\!](y) \mid \overline{y}\ u\ v.(u().[\![True]\!](z) + v().[\![False]\!](z)))$

  On peut réaliser la communication le long du canal y, ce qui permet de substituer les noms t et f respectivement par u et v dans la partie gauche de la composition.

$\rightarrow \tau.(\nu\ u\ v)\ (\overline{u}.[\![True]\!](y) \mid (u().[\![True]\!](z) + v().[\![False]\!](z)))$

Une deuxième communication est possible le long du canal u.

$\rightarrow \tau.\tau.(\nu\ u\ v)\ ([\![True]\!](y) \mid [\![True]\!](z))$

Or u et v n'apparaissent plus dans la composition parallèle, on supprime alors la restriction sur u et v.

$\rightarrow \tau.\tau.([\![True]\!](y) \mid [\![True]\!](z))$

On remarque donc que le résultat de $copy\_bool(y, z)$ est la composition parallèle de deux processus. Le premier pointe toujours la valeur $True$ par le canal y. Le deuxième pointe la valeur $True$ par le canal z.

- La fonction $equal\_bool$ teste l'égalité de deux booléens pointés respectivement par les canaux b_1 et b_2 et copie la valeur pointée par le canal b_t ($True$) ou la valeur pointée par le canal b_f ($False$) dans le canal res.

$$\begin{aligned}
equal_bool(b_1, b_2, res) \;=\; & (\nu\ u\ v\ u'\ v')\ \overline{b_1}\ u\ v.\overline{b_2}\ u'\ v'. \\
& (u().u'().copy_bool(b_t, res) \\
& +\ v().v'().copy_bool(b_t, res) \\
& +\ u().v'().copy_bool(b_f, res) \\
& +\ v().u'().copy_bool(b_f, res))
\end{aligned}$$

- L'opérateur Not calcule la négation d'un booléen pointé par un canal b et copie la valeur pointée par le canal b_t ($True$) ou la valeur pointée par le canal b_f ($False$) dans le canal résultat nb.

$$Not(b, nb) = (\nu\ u\ v)\ \overline{b}\ u\ v.(u().copy\_bool(b_f, nb)\ +\ v().copy\_bool(b_t, nb))$$

- L'opérateur And calcule la conjonction de deux booléens pointés par les canaux b_1 et b_2 et copie la valeur pointée par le canal b_t ($True$) ou la valeur pointée par le canal b_f ($False$) dans le canal résultat res.

$$\begin{aligned}
And(b_1,b_2,res) \;=\; & (\nu\ u\ v\ u'\ v')\ \overline{b_1}\ u\ v.\overline{b_2}\ u'\ v'. \\
& (u().u'().copy_bool(b_t, res) \\
& +\ v().v'().copy_bool(b_f, res) \\
& +\ u().v'().copy_bool(b_f, res) \\
& +\ v().u'().copy_bool(b_f, res))
\end{aligned}$$

On peut coder de la même manière les autres opérateurs sur les booléens.

On peut également coder des instructions dans le π-calcul. Ainsi, l'instruction ($If\ e\ then\ P\ else\ Q$), e étant une expression booléenne, peut être codée par le processus:

$$(\nu\ b)([\![e]\!]\ b \mid (\nu\ u\ v)(\overline{b}\ u\ v.(u().P\ +\ v().Q)))$$

On évalue d'abord l'expression e, le résultat est pointé par un nouveau canal b. Parallèlement, on envoie deux nouveaux noms u et v sur b. Si on reçoit un signal sur u (ce qui veut dire que e vaut $True$), on exécute P sinon on exécute Q.

Ce codage des booléens s'appuie sur le fait qu'il y a un nombre fini de valeurs booléennes (un nom est réservé pour chaque valeur), ce qui permet d'utiliser la somme finie de processus pour traiter les différentes valeurs possibles.

2.2.3 Codage des entiers naturels

Les entiers naturels sont infinis, on ne peut les coder de la même manière que les booléens (i.e, en utilisant une somme finie de processus). On va les coder inductivement par le zéro et le successeur (par analogie aux entiers de Church).

- L'entier zéro est codé par le processus $[\![0]\!](l)$ qui attend deux noms de canaux z (pour zéro) et s (pour successeur) sur le canal de nom l, émet un signal sur le premier canal z et revient à son état initial. On dira que l'entier zéro est pointé par le canal l.

  $[\![0]\!](l) = l(z\ s).\overline{z}.[\![0]\!](l)$

- L'entier $succ(n)$ est codé par le processus $[\![succ(n)]\!](l)$. Ce processus est la composition parallèle d'un processus qui définit n ($[\![n]\!](l')$), n est pointé par un nouveau nom l'. L'autre processus attend sur l deux noms de canaux z et s et envoie l' sur le canal s. Autrement-dit, on envoie sur le canal s le canal l' qui pointe le prédécesseur n.

  $[\![succ(n)]\!](l) = (\nu\ l')(l(z\ s).\overline{s}\ l' \mid [\![n]\!](l'))$

Avec cette représentation, l'entier 1 est codé par le processus $(\nu\ l')(l(z\ s).\overline{s}\ l' \mid l'(z\ s).\overline{z})$.

En utilisant cette définition inductive, on peut coder les opérateurs arithmétiques sur les entiers naturels.

- La fonction *copy* effectue la copie d'un entier pointé par un canal l dans un autre canal m.

  $$copy(l,m) = (\nu\ u\ v)\overline{l}\ u\ v.(u().[\![0]\!](m) + v(l').(\nu\ m')(m(z\ s).\overline{s}\ m' \mid copy(l',m')))$$

  La fonction *copy* introduit deux nouveaux noms u et v et envoie ces deux noms sur le canal l. Si elle reçoit un signal sur le canal u (la valeur pointée par le canal l est zéro), elle crée un processus $[\![0]\!](m)$ (i.e, copie zéro dans m). Sinon, elle reçoit un nom l' sur le canal v (il s'agit d'un entier $succ(n)$, n étant pointé par le canal l'). Dans ce cas, elle introduit un nouveau nom m' dans lequel elle copie la valeur pointée par l'. Parallèlement, elle attend deux noms z et s sur le canal m et envoie m' sur s. Autrement-dit, $succ(n)$ est maintenant pointé par le canal m sachant que n est pointé par le canal m' (définition inductive de la copie).

– La fonction add effectue l'addition de deux entiers pointés respectivement par les canaux m et n, le résultat est pointé par le canal res.

$$\begin{aligned} add(m,n,res) = (\nu\ u\ v)\overline{m}\ u\ v.(&u().copy(n,res) \\ &+\ v(m').(\nu\ res')(res(z\ s).\overline{s}\ res' \mid add(m',n,res'))) \end{aligned}$$

La fonction add introduit deux nouveaux noms u et v et envoie ces deux noms sur le canal m. Si elle reçoit un signal sur le canal u (la valeur pointée par le canal m est zéro), elle effectue la copie de l'entier pointé par le canal n dans le canal res. Sinon, elle reçoit un nom m' sur le canal v (il s'agit d'un entier $succ(a)$, a étant pointé par le canal m'). Dans ce cas, elle introduit un nouveau nom res' qui pointera le résultat (b) de l'addition des entiers pointés par m' et n. Parallèlement, elle attend deux noms z et s sur le canal res et émet le canal res' sur s. Autrement-dit, $succ(b)$ est la valeur pointée par le canal res sachant que b est pointé par le canal res'.

– La fonction $equal$ teste l'égalité de deux entiers pointés respectivement par les canaux m et n, le résultat (booléen) est pointé par le canal res. Si les deux canaux pointent l'entier zéro, elle renvoie $True$. Sinon, si l'un des deux canaux pointe l'entier zéro et l'autre pas, elle renvoie $False$. Sinon, elle rappelle la fonction $equal$ avec les canaux pointant les prédécesseurs des entiers pointés par m et n.

$$\begin{aligned} equal(m,n,res) = (\nu\ u\ v\ u'\ v')\overline{m}\ u\ v.\overline{n}\ u'\ v'.& \\ (u().u'().copy\_bool(b_t,res)& \\ +\ u().v'(n').copy\_bool(b_f,res)& \\ +\ v(m').u'().copy\_bool(b_f,res)& \\ +\ v(m').v'(n').equal(m',n',res))& \end{aligned}$$

2.2.4 Codage des listes

– La valeur de la liste vide (Nil) est codée par le processus $[\![Nil]\!](n)$ qui attend sur le canal n deux noms de canaux v (pour vide) et p (pour pas vide) et envoie un signal sur le premier canal v. On dira que la valeur Nil est pointée par le canal n.

$[\![Nil]\!](n) = n(v\ p).\overline{v}.[\![Nil]\!](n)$

– La valeur de la liste $cons(a,l)$ est codée par le processus $[\![cons(a,l)]\!](c)$ qui est la composition parallèle de trois processus. Un processus qui évalue l'élément de tête de la liste (a), le résultat est pointé par le canal car. Le deuxième processus évalue le reste de la liste (l), le résultat est pointé par le canal cdr. Le dernier processus attend sur le canal c deux noms de canaux v et p et envoie car et cdr sur le canal p. Autrement-dit, le canal c pointera $cons(a,l)$.

$[\![cons(a,l)]\!](c) = (\nu\ car\ cdr)(c(v\ p).\overline{p}\ car\ cdr \mid [\![a]\!](car) \mid [\![l]\!](cdr))$

On peut coder des fonctions manipulant des listes. Supposons que les éléments des listes sont de type entier.

- La fonction *copy_list* effectue la copie de la liste pointée par le canal l dans une autre liste pointée par le canal m.

$$
\begin{aligned}
copy\_list(l,m) \;=\; (\nu \;\; x \;\; y)\overline{l} \;\; x \;\; y.(&x().[\![Nil]\!](m) \\
&+ \; y(d \;\; f).(\nu \;\; m')(m(v \;\; p).\overline{p} \;\; d \;\; m' \mid copy\_list(f,m'))
\end{aligned}
$$

- La fonction *head_list* retourne l'élément de tête d'une liste non vide pointée par le canal l. Le résultat est pointé par le canal h.

$$head\_list(l,h) \;=\; (\nu \;\; x \;\; y)\overline{l} \;\; x \;\; y.y(d \;\; f).copy(d,h)$$

  On introduit deux nouveaux noms de canaux x et y et on les envoie sur le canal l. Sachant que la liste est non vide, on reçoit sur le canal y deux noms d et f pointant la tête et le reste de la liste. Ensuite, on copie l'entier pointé par le canal d dans le canal h.

- La fonction *tail_list* retourne le reste de la liste pointée par le canal l. Le résultat est pointé par le canal r.

$$tail\_list(l,r) \;=\; (\nu \;\; x \;\; y)\overline{l} \;\; x \;\; y.(x().[\![Nil]\!](r) \; + \; y(d \;\; f).copy\_list(f,r))$$

- La fonction *equal_list* teste l'égalité de deux listes d'entiers pointées respectivement par les canaux m et n, le résultat (booléen) est pointé par le canal res. Si les deux canaux pointent Nil, elle renvoie $True$. Sinon, si l'un des deux canaux pointe Nil et l'autre pas, elle renvoie $False$. Sinon, en parallèle, elle rappelle la fonction *equal_list* avec en paramètres les canaux pointant les restes des listes pointées par m et n, teste l'égalité des éléments de tête des deux listes et ramène la conjonction des deux résultats.

$$
\begin{aligned}
equal\_list(m,n,res) \;=\; & (\nu \;\; u \;\; v \;\; u' \;\; v')\overline{m} \;\; u \;\; v.\overline{n} \;\; u' \;\; v'. \\
& (u().u'().copy\_bool(b_t,res) \\
& + \; u().v'(d',f').copy\_bool(b_f,res) \\
& + \; v(d,f).u'().copy\_bool(b_f,res) \\
& + \; v(d,f).v'(d',f').(\nu \;\; b_1 \;\; b_2)(equal(d,d',b_1) \mid equal\_list(f,f',b_2) \\
& \qquad\qquad\qquad\qquad \mid And(b_1,b_2,res)))
\end{aligned}
$$

D'autres exemples de codage de types de données et de fonctions sur ces types de données peuvent être consultés dans [MPW89a].

3 Description informelle du BRP

Le BRP est un protocole de communication développé dans le laboratoire de recherche de Philips. Il communique des messages d'un producteur à un consommateur sur une ligne physique non fiable (i.e, qui peut perdre les messages). C'est une extension non triviale du protocole du bit alterné qui utilise des timeouts et qui interrompt la transmission suite à un nombre borné de tentatives de retransmissions.

3.1 Environnement du protocole

L'environnement du protocole comprend un producteur et un consommateur. La vue globale du système est illustrée par la figure 4. Le système accepte des requêtes $Req(f)$ du producteur pour transmettre le fichier f au consommateur.

Quand la transmission est terminée ou interrompue, le producteur reçoit une confirmation $Conf(c)$ où c prend la valeur OK, $NOTOK$ ou $DONTKNOW$ indiquant respectivement que le fichier a été transmis complètement, que la transmission a été interrompue ou que le dernier message du fichier n'a pas été acquitté mais a peut être été reçu par le consommateur.

Le consommateur peut recevoir un signal Ind_err indiquant que la transmission du fichier a été interrompue ou un signal $Ind(m, i)$ où m est le message courant et i pouvant prendre la valeur $FIRST$, $LAST$ ou $INCOMPLETE$ indiquant qu'il s'agit respectivement du premier message, du dernier message ou d'un message intermédiaire du fichier.

3.2 Composantes du protocole

Le système se compose d'un émetteur et d'un récepteur liés par deux canaux K (pour les messages) et L (pour les acquittements) ayant chacun un buffer d'un seul message. Ces deux canaux ne sont pas fiables, ils peuvent perdre les messages ou les acquittements.

3.2.1 Comportement de l'émetteur

L'émetteur envoie chaque message sur le canal K, incrémente le nombre de retransmissions, déclenche son timer ($timer1$) et attend un acquittement sur le canal L. Ce timer est utilisé pour détecter la perte de message ou d'acquittement. Le temps associé à ce timer dépasse le temps nécessaire pour qu'un message soit transmis sur le canal K et que l'acquittement correspondant soit revenu sur le canal L.

Si un acquittement est reçu par l'émetteur avant l'expiration de $timer1$, $timer1$ est désactivé et le message suivant du fichier est transmis. Quand le transfert est achevé, l'émetteur envoie une confirmation OK au producteur sur le canal $Conf$.

Si aucun acquittement n'est reçu, $timer1$ expire et l'émetteur retransmet le message. Le nombre de retransmissions possibles est borné (MAX). Si le nombre maximum de retransmissions est atteint, l'émetteur interrompt le transfert et envoie une confirmation d'echec

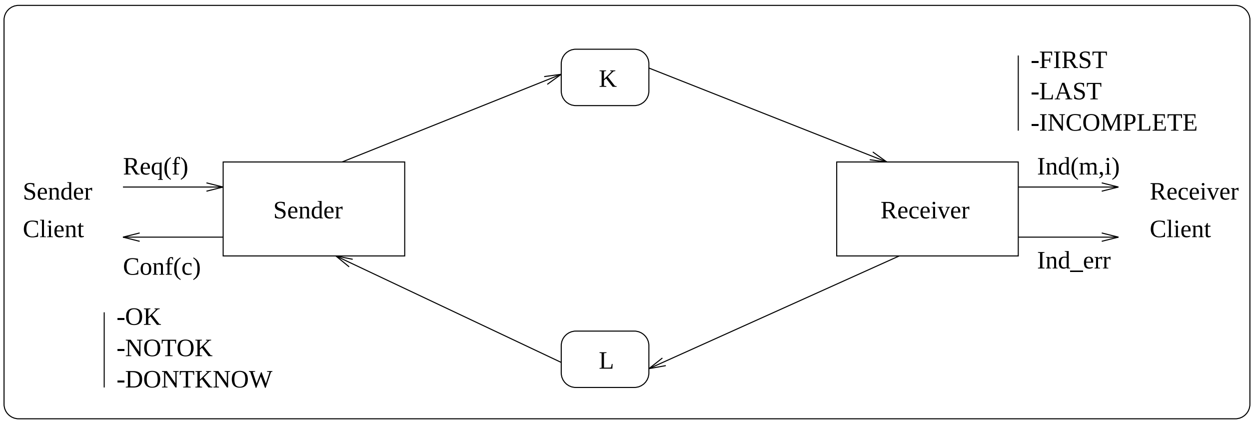

FIG. 4 - *Description informelle du BRP*

de transmission au producteur sur le canal $Conf$. La confirmation est $NOTOK$ si l'abort a eu lieu lors du transfert d'un message intermédiaire ou $DONTKNOW$ si l'abort a eu lieu pendant le transfert du dernier message du fichier.

3.2.2 Comportement du récepteur

Le récepteur attend chaque message sur le canal K.

Si le message a été déjà reçu, le récepteur retransmet un acquittement pour ce message sur le canal L. Le récepteur peut détecter une retransmission de message en comparant le bit alterné (tag) du message reçu à celui du message précédent qu'il mémorise localement.

S'il s'agit d'un nouveau message, le récepteur envoie une indication (msg, i) sur le canal Ind, le champ i pouvant prendre la valeur $FIRST$, $LAST$ ou $INCOMPLETE$. Il envoie ensuite un acquittement sur le canal L et déclenche son timer ($timer2$). Le temps associé à ce timer dépasse le temps nécessaire pour transmettre MAX fois un message (i.e, $timer2 > MAX * timer1$). Ce timer sert à resynchroniser l'émetteur et le récepteur suite à un abort de la transmission.

Si $timer2$ expire (i.e, aucun nouveau message n'est reçu par le récepteur), le récepteur envoie une indication d'abort de la transmission sur le canal Ind_err pour informer le consommateur.

4 Vue abstraite du protocole dans le π-calcul

4.1 Définition de la vue abstraite

La vue abstraite du protocole est le comportement observable par l'environnement au niveau des canaux externes Req, $Conf$, Ind et Ind_err, faisant abstraction sur les communications le long des canaux internes K et L. Le traitement interne du protocole est considéré donc comme une boîte noire. Seules les relations entre les entrées au niveau du canal Req et les sorties au niveau des canaux $Conf$, Ind et Ind_err sont spécifiées.

4.2 Spécification de la vue abstraite dans le π-calcul

Intuitivement, le système accepte des requêtes $Req(f)$ du producteur pour transmettre le fichier f au consommateur.

Quand la transmission est terminée ou interrompue, le producteur reçoit une confirmation $Conf(c)$ où le champ c peut prendre la valeur OK, $NOTOK$ ou $DONTKNOW$ indiquant respectivement que le fichier a été transmis complètement, que la transmission a été interrompue ou que le dernier message du fichier n'a pas été acquitté mais a peut être été délivré au consommateur.

Le consommateur peut recevoir un signal Ind_err indiquant que la transmission du fichier a été interrompue ou un signal $Ind(m, i)$ où m est le message courant et i pouvant prendre la valeur $FIRST$, $LAST$ ou $INCOMPLETE$ indiquant qu'il s'agit respectivement du premier message, du dernier message ou d'un message intermédiaire du fichier.

Noter que si le fichier contient un seul message, alors ce message doit être délivré avec une indication $LAST$ et non $FIRST$. De plus, si un abort se produit lors de la transmission de ce message, le producteur doit recevoir une confirmation $DONTKNOW$ et non $NOTOK$.

Ce comportement est illustré par la figure 5 et est décrit par les 3 équations suivantes où le fichier à transférer est modélisé par une liste de messages.

L'état S_0 est l'état initial. Si le fichier reçu est vide, on envoie tout de suite une confirmation OK au producteur sur le canal $Conf$ et on retourne à l'état initial S_0. Sinon, on va dans l'état S_1 pour transmettre le message de tête du fichier.

$$S_0 \stackrel{def}{=} Req(Nil).\overline{Conf}\ OK.S_0 \ + \ Req(cons(head, tail)).S_1(head, tail)$$

Dans l'état S_1, on suppose que le premier message du fichier arrive au moins une fois chez le récepteur. Sinon, l'émetteur peut interrompre le transfert sachant qu'il n'y a aucun moyen pour avertir le récepteur (qui n'est au courant de rien) ce qui conduirait à une situation de deadlock. Le premier message du fichier est délivré au consommateur avec une indication

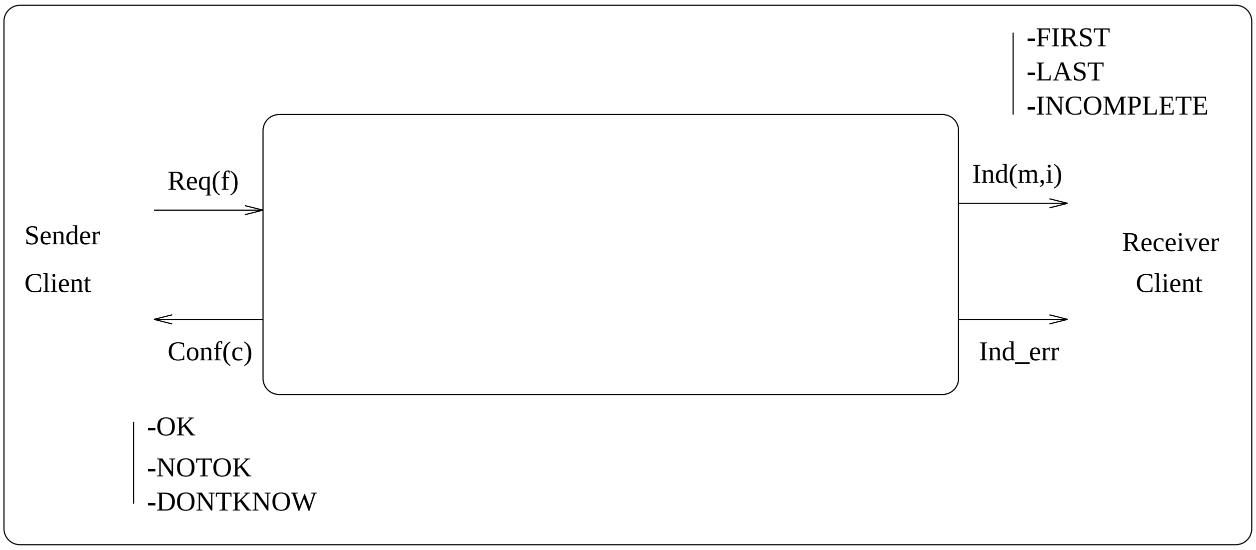

FIG. 5 - *Vue abstraite du BRP*

$FIRST$ si le fichier contient d'autres messages ou alors avec une indication $LAST$ si c'est le seul message du fichier.

Si l'acquittement correspondant au premier message est reçu par l'émetteur, si ce message est aussi le dernier message du fichier, on envoie une confirmation OK au producteur sur le canal $Conf$ et on retourne à l'état initial S_0. Sinon (le fichier contient d'autres messages), on va dans l'état S_2 pour transférer le reste du fichier. Dans S_2, hd et tl représentent respectivement la tête et le reste du fichier $tail$.

Si l'acquittement correspondant au premier message n'est pas reçu, si ce message n'est pas le dernier message du fichier, on envoie une confirmation $NOTOK$ au producteur et une indication d'erreur au consommateur. Sinon, une confirmation $DONTKNOW$ est envoyée au producteur. Dans tous les cas, on revient à l'état initial S_0.

Noter que l'action non observable τ dans S_1 correspond à l'émission d'un message par l'émetteur et à sa réception par le récepteur et à l'émission de l'acquittement correspondant par le récepteur et à sa réception ou pas par l'émetteur.

$$
\begin{aligned}
S_1(head,tail) \stackrel{def}{=} \; & \tau.\overline{Ind}\ head\ LAST.\overline{Conf}\ OK.S_0 \\
& + \tau.\overline{Ind}\ head\ FIRST.S_2(hd,tl) \\
& + \tau.\overline{Ind}\ head\ LAST.\overline{Conf}\ DONTKNOW.S_0 \\
& + \tau.\overline{Ind}\ head\ FIRST.\overline{Conf}\ NOTOK.\overline{Inderr}.S_0
\end{aligned}
$$

Dans l'état S_2, si le message suivant du fichier n'arrive jamais côté récepteur, on interrompt le transfert. Si on était en train de transférer le dernier message du fichier, on envoie une confirmation $DONTKNOW$ au producteur, sinon, on envoie une confirmation $NOTOK$ au producteur. Dans les deux cas, une indication d'erreur est envoyée au consommateur. Ensuite, on revient à l'état initial S_0.

Si le message suivant du fichier est reçu par le récepteur, on délivre le message au consommateur avec une indication $LAST$ ou $INCOMPLETE$ selon qu'il s'agissait du dernier message du fichier ou pas et on envoie un acquittement à l'émetteur.

Si l'acquittement est reçu par l'émetteur, s'il s'agit du dernier message du fichier, on envoie une confirmation OK au producteur et on revient à l'état initial S_0. S'il s'agit d'un message intermédiaire, on va dans l'état S_2 pour transférer le reste du fichier ($hdtl$ et $tltl$ représentent respectivement la tête et le reste du fichier tl).

Si l'acquittement n'est jamais reçu (i.e, le message a été retransmis MAX fois), on interrompt le transfert. Si on était en train de transférer le dernier message du fichier, on envoie une confirmation $DONTKNOW$ au producteur sans envoyer une indication d'erreur au consommateur car le message a déjà été délivré. Si l'interruption a eu lieu lors du transfert d'un message intermédiaire, on envoie une confirmation $NOTOK$ au producteur et une indication d'erreur au consommateur. Dans les deux cas, on revient à l'état initial S_0.

Noter que l'action non observable τ appraissant dans S_2 correspond:

- soit à plusieurs tentatives de transmission du message suivant du fichier, le message n'arrive jamais côté récepteur,
- soit à plusieurs tentatives de transmission du message suivant du fichier, le message finit par arriver chez le récepteur, mais l'acquittement n'est jamais reçu par l'émetteur.
- soit à plusieurs tentatives de transmission du message suivant du fichier, le message est arrivé chez le récepteur et l'acquittement est reçu par l'émetteur.

$$
\begin{aligned}
S_2(hd,tl) \stackrel{def}{=} \; & \tau.\overline{Conf}\ \ DONTKNOW.\overline{Ind\_err}.S_0 \\
+ \; & \tau.\overline{Conf}\ \ NOTOK.\overline{Ind\_err}.S_0 \\
+ \; & \tau.\overline{Ind}\ \ hd\ \ LAST.\overline{Conf}\ \ DONTKNOW.S_0 \\
+ \; & \tau.\overline{Ind}\ \ hd\ \ INCOMPLETE.\overline{Conf}\ \ NOTOK.\overline{Inderr}.S_0 \\
+ \; & \tau.\overline{Ind}\ \ hd\ \ LAST.\overline{Conf}\ \ OK.S_0 \\
+ \; & \tau.\overline{Ind}\ \ hd\ \ INCOMPLETE.S_2(hdtl,tltl)
\end{aligned}
$$

La spécification S_0 n'exprime pas explicitement la perte de message ou la perte d'acquittement. Alors, elle ne donne pas la garantie qu'une perte de message ou d'acquittement va se produire. Pour pouvoir faire cela, il faut rendre les actions non observables τ représentant l'émission d'un message sur le canal K par l'émetteur et sa réception par le récepteur, ou l'émission d'un acquittement sur le canal L par le récepteur et sa réception par l'émetteur, des actions observables. Toutefois, la spécification suppose que les pertes sont possibles.

Si on récapitule, on obtient les 3 équations suivantes:

$$S_0 \stackrel{def}{=} Req(Nil).\overline{Conf}\ \ OK.S_0 \; + \; Req(cons(head,tail)).S_1(head,tail)$$

$$
\begin{aligned}
S_1(head,tail) \stackrel{def}{=}\ & \tau.\overline{Ind}\ \ head\ \ LAST.\overline{Conf}\ \ OK.S_0 \\
& +\ \tau.\overline{Ind}\ \ head\ \ FIRST.S_2(hd,tl) \\
& +\ \tau.\overline{Ind}\ \ head\ \ LAST.\overline{Conf}\ \ DONTKNOW.S_0 \\
& +\ \tau.\overline{Ind}\ \ head\ \ FIRST.\overline{Conf}\ \ NOTOK.\overline{Inderr}.S_0
\end{aligned}
$$

$$
\begin{aligned}
S_2(hd,tl) \stackrel{def}{=}\ & \tau.\overline{Conf}\ \ DONTKNOW.\overline{Ind\_err}.S_0 \\
& +\ \tau.\overline{Conf}\ \ NOTOK.\overline{Ind\_err}.S_0 \\
& +\ \tau.\overline{Ind}\ \ hd\ \ LAST.\overline{Conf}\ \ DONTKNOW.S_0 \\
& +\ \tau.\overline{Ind}\ \ hd\ \ INCOMPLETE.\overline{Conf}\ \ NOTOK.\overline{Inderr}.S_0 \\
& +\ \tau.\overline{Ind}\ \ hd\ \ LAST.\overline{Conf}\ \ OK.S_0 \\
& +\ \tau.\overline{Ind}\ \ hd\ \ INCOMPLETE.S_2(hdtl,tltl)
\end{aligned}
$$

5 Description formelle du BRP dans le π-calcul

5.1 Les choix d'implantation

Le point de départ est la description informelle du protocole qui le décrit comme une composition parallèle de composantes s'exécutant séquentiellement. Ceci se traduit bien dans le π-calcul. Cependant, plusieurs points restent à éclaircir, parmi lesquels:

1. le contenu du fichier à transférer (le nombre de messages et les traitements spécifiques),
2. le fonctionnement de $timer2$ (est-il déclenché au démarrage? qu'arrive t-il en cas d'abort du transfert ou en cas de terminaison d'un transfert?) sachant que la seule phrase de la description informelle qui évoque $timer2$ est la suivante: "$timer2$ sert à la synchronisation suite à un abort de transfert",
3. la gestion du bit alterné (est-ce que l'inversion du bit alterné continue dans le cas où plusieurs requêtes de transfert de fichier arrivent les unes après les autres ou est-ce qu'il faut le réinitialiser avant chaque transfert?)
4. et l'émission d'une indication d'erreur par le récepteur vers le consommateur quand un abort se produit (dans quels cas le récepteur doit-il signaler l'abort au consommateur?).

La première hypothèse faite est que le fichier à transférer est disponible complètement chez l'émetteur avant que celui-ci commence à transférer les messages car on pourrait imaginer que les messages arrivent un par un de la couche supérieure et qu'ils sont traités au fur et à mesure de leur réception par l'émetteur. Cette hypothèse est faite dans la plupart des travaux réalisés autour du BRP. Le fichier à transférer sera modélisé par une *liste de messages de type quelconque* car les messages ne subiront aucun traitement pendant l'exécution du protocole.

On suppose que le fichier peut contenir 0, un ou plusieurs messages:

- si 0 message: l'émetteur envoie tout de suite une confirmation OK au producteur, demande au récepteur de se réinitialiser et revient à son état initial.
- si 1 message: ce message doit être considéré comme le dernier message du fichier et non comme le premier, sinon, c'est que l'on suppose qu'il y a d'autres messages qui vont arriver après. Le récepteur doit délivrer ce message avec une indication $LAST$ et non $FIRST$. De plus, si un abort de transfert se produit lors de la transmission de ce message, l'émetteur doit envoyer une confirmation $DONTKNOW$ (et non $NOTOK$) au producteur. On avait supposé que le premier message du fichier arrive au moins une fois chez le récepteur.
- si plusieurs messages: le premier message est délivré avec une indication $FIRST$, les messages intermédiaires sont délivrés avec une indication $INCOMPLETE$ et le dernier message est délivré avec une indication $LAST$.

L'hypothèse que $timer2 > (MAX * timer1)$ fait que l'émetteur réagit toujours le premier à un abort. Il peut alors recevoir une nouvelle requête de transfert de fichier et émettre le premier message de ce fichier alors que le récepteur n'a pas encore réagit à l'abort (i.e, $timer2$ n'a pas encore expiré). Le récepteur peut considérer ce message comme le message suivant du fichier précédent ou comme une duplication de message ce qui est faux. L'émetteur doit donc attendre que le récepteur réagisse proprement à l'abort. Mais, aucun mécanisme dans le BRP ne permet de notifier l'expiration du $timer2$ du récepteur à l'émetteur. D'autre part, il n'est pas évident comment l'émetteur pourrait savoir que le récepteur a réagit à l'abort surtout s'ils sont géographiquement éloignés.

Pour résoudre ce problème, l'émetteur doit attendre que le récepteur réagisse à l'abort en réalisant un rendez-vous sur le canal *abort* avec le récepteur. De plus, l'émetteur et le récepteur doivent réinitialiser leurs variables (i.e, doivent réinitialiser le schéma du bit alterné).

Après la terminaison correcte d'un transfert de fichier, deux situations sont possibles:

- La première est que l'émetteur peut recevoir une nouvelle requête de transfert de fichier et émettre le premier message de ce fichier alors que le récepteur n'a pas encore réinitialisé ses variables car $timer2$ n'a pas encore expiré (l'émetteur par contre a réinitialisé ses variables). Il peut donc considérer ce premier message comme une duplication du message précédent ce qui est faux.

- La deuxième situation est que aucune nouvelle requête n'arrive chez l'émetteur, alors $timer2$ expire et le récepteur attend le signal *abort* ce qui est également faux et conduit à une situation de deadlock.

Pour résoudre ces problèmes, l'émetteur doit signaler la fin du transfert d'un fichier à l'aide du canal *restart* au récepteur afin que ce dernier anticipe l'expiration de son timer et qu'ils réinitialisent tous les deux leurs variables (i.e, le schéma du bit alterné) avant de commencer un nouveau transfert.

Donc, que le transfert se termine ou soit interrompu:

- l'émetteur et le récepteur réinitialisent leurs variables (i.e, réinitialisent le schéma du bit alterné) et

- le $timer2$ du récepteur est désactivé.

De plus, l'émetteur et le récepteur se synchronisent sur le canal *abort* suite à un abort de transfert d'un fichier. De même, ils se synchronisent sur le canal *restart* suite à une bonne terminaison de transfert d'un fichier.

Enfin, si l'émetteur interrompt le transfert lors de la transmission d'un message intermédiaire du fichier, le récepteur doit émettre une indication d'erreur au consommateur. Par contre, si l'abort a eu lieu lors du transfert du dernier message du fichier, deux situations sont possibles côté récepteur:

- Soit ce message a été reçu et délivré par le récepteur; dans ce cas, le récepteur n'envoie pas d'indication d'erreur au consommateur.

- Soit ce message n'est jamais arrivé chez le récepteur (il a été perdu MAX fois) et dans ce cas, le récepteur doit envoyer une indication d'erreur au consommateur.

5.2 Variables de l'émetteur et du récepteur

L'émetteur manipule les variables suivantes:

- $first$: cette variable indique qu'il s'agit du premier message du fichier si elle vaut $True$. Elle est initialisée à $True$ et remise à $False$ dès que le récepteur reçoit l'acquittement correspondant au premier message.

- $last$: cette variable indique qu'il s'agit du dernier message du fichier si elle vaut $True$. Elle est initialisée à $False$ et posititionnée à $True$ dès que l'émetteur traite le dernier message du fichier.

- tag: bit alterné permettant au récepteur de détecter les retransmissions de messages. Cette variable est initialisée à $True$, elle est inversée à chaque fois que l'émetteur doit transmettre un nouveau message.

- rn: nombre de tentatives de retransmissions d'un message. Cette variable est initialisée à 0 à chaque fois que l'émetteur doit transmettre un nouveau message. Elle est incrémentée de 1 à chaque fois qu'il doit retransmettre ce message.

Chaque message transmis par l'émetteur est un quadruplet contenant les informations $first$, $last$, tag et une donnée du fichier.

L'émetteur utilise également une constante MAX qui contient le nombre maximum de tentatives de retransmissions d'un message.

Le récepteur gère les variables suivantes:

- $rtag$: variable mémorisant le bit alterné du message précédent. Elle permet au récepteur de détecter les retransmissions de messages dans le cas où le tag du message reçu est identique à $rtag$. Cette variable est initialisée à $False$ ce qui permet au récepteur de traiter le premier message reçu comme un nouveau message. Elle est inversée à chaque fois que le récepteur reçoit un nouveau message.

- end: cette variable indique que le récepteur a reçu le dernier message du fichier si elle vaut $True$. Elle est initialisée à $False$ et permet au récepteur d'envoyer une indication d'erreur au consommateur dans le cas où un abort se produit lors du transfert d'un message intermédiaire du fichier, ou dans le cas où l'abort a eu lieu lors du transfert du dernier message du fichier, lequel n'est jamais arrivé chez le récepteur (i.e, quand end vaut $False$).

- $t2enabled$: cette variable indique que $timer2$ est déclenché si elle vaut $True$. Elle est initialisée à $False$ et permet au récepteur de savoir s'il peut recevoir le signal $timeout2$ ou s'il peut désactiver $timer2$ après un abort ou une terminaison de transfert d'un fichier.

5.3 Description du protocole

5.3.1 Modélisation des canaux externes

Le comportement attendu du système est le suivant: toute réception de requête de transmission de fichier sur le canal Req conduira à:

- une confirmation sur la canal $Conf$,
- éventuellement une indication d'erreur sur le canal Ind_err,
- une ou plusieurs indications de livraison de message sur le canal Ind.

Les canaux Req, $Conf$, Ind et Ind_err seront considérés comme des constantes. Ils ne pourront jamais être liés et resteront fixes tout le long de l'exécution du système.

5.3.2 Modélisation des événements externes

Les événements externes de perte de message ou de perte d'acquittement peuvent se produire à tout moment. Ces deux événements sont modélisés de la façon suivante:

- l'événement externe de perte de message est modélisé par un processus qui intercepte simplement le message émis par l'émetteur sur le canal K; ensuite, il revient à son état initial.

  $$perte\_msg(K) \stackrel{def}{=} K(first\ \ last\ \ tag\ \ m).perte\_msg(K)$$

- l'événement externe de perte d'acquittement est modélisé par un processus qui intercepte simplement l'acquittement émis par le récepteur sur le canal L; ensuite, il revient à son état initial.

  $$perte\_ack(L) \stackrel{def}{=} L.perte\_ack(L)$$

5.3.3 Modélisation des timers

Le processus $Timer1$ modélise le timer de l'émetteur. Ce processus attend simplement un signal sur le canal $time1$, la réception de ce signal correspond à l'activation de $timer1$. Ensuite, il émet un signal sur le canal $timeout1$ pour signaler l'expiration du timer.

L'activation de ce timer par l'émetteur se fait en émettant un signal sur le canal $time1$. La désactivation de ce timer par l'émetteur ou son expiration est modélisée par la réception d'un signal sur le canal $timeout1$.

$$Timer1 \stackrel{def}{=} time1.\overline{timeout1}.Timer1$$

De même, le processus $Timer2$ modélise le timer du récepteur. Ce processus attend simplement un signal sur le canal $time2$, la réception de ce signal correspond à l'activation de $timer2$. Ensuite, il émet un signal sur le canal $timeout2$ pour signaler l'expiration du timer.

L'activation de ce timer par le récepteur se fait en émettant un signal sur le canal $time2$. La désactivation de ce timer par le récepteur ou son expiration est modélisée par la réception d'un signal sur le canal $timeout2$.

$$Timer2 \stackrel{def}{=} time2.\overline{timeout2}.Timer2$$

5.3.4 Composantes du protocole

L'émetteur et son timer constituent la composante P du système.

$$P(K, L, abort, restart) \stackrel{def}{=} (\nu\ \ time1\ \ timeout1)\ \ (Sender(K, L, abort, restart) \mid Timer1)$$

Quant au récepteur et son timer, ils forment la composante Q du système.

$$Q(K, L, abort, restart) \stackrel{def}{=} (\nu\ \ time2\ \ timeout2)\ \ (Receiver(K, L, abort, restart) \mid Timer2)$$

Ces deux composantes interagissent via l'émetteur et le récepteur. Ceci est illustré par la figure 6. L'émetteur et le récepteur sont liés par quatre canaux K, L, $abort$ et $restart$. Ces 4 canaux sont locaux dans la mesure où ils ne seront jamais transmis comme objets dans les communications avec l'extérieur. Ils ne sont accessibles par aucun processus de l'environnement du système. Les canaux K, L, $abort$ et $restart$ sont donc privés au système.

Le système est donc la composition parallèle des composantes P et Q et des événements externes $perte\_msg$ et $perte\_ack$.

$$\begin{aligned} System \equiv (\nu\ \ K\ \ L\ \ abort\ \ restart)(&P(K, L, abort, restart) \\ &\mid Q(K, L, abort, restart) \\ &\mid perte\_msg(K) \\ &\mid perte\_ack(L)) \end{aligned}$$

Pour ne pas encombrer le système, les événements externes $perte\_msg$ et $perte\_ack$ ne seront pas représentés dans la composition parallèle. Toutefois, ils seront utilisés si nécessaire pendant la preuve du protocole.

$$System \equiv (\nu\ \ K\ \ L\ \ abort\ \ restart)\ \ (P(K, L, abort, restart) \mid Q(K, L, abort, restart))$$

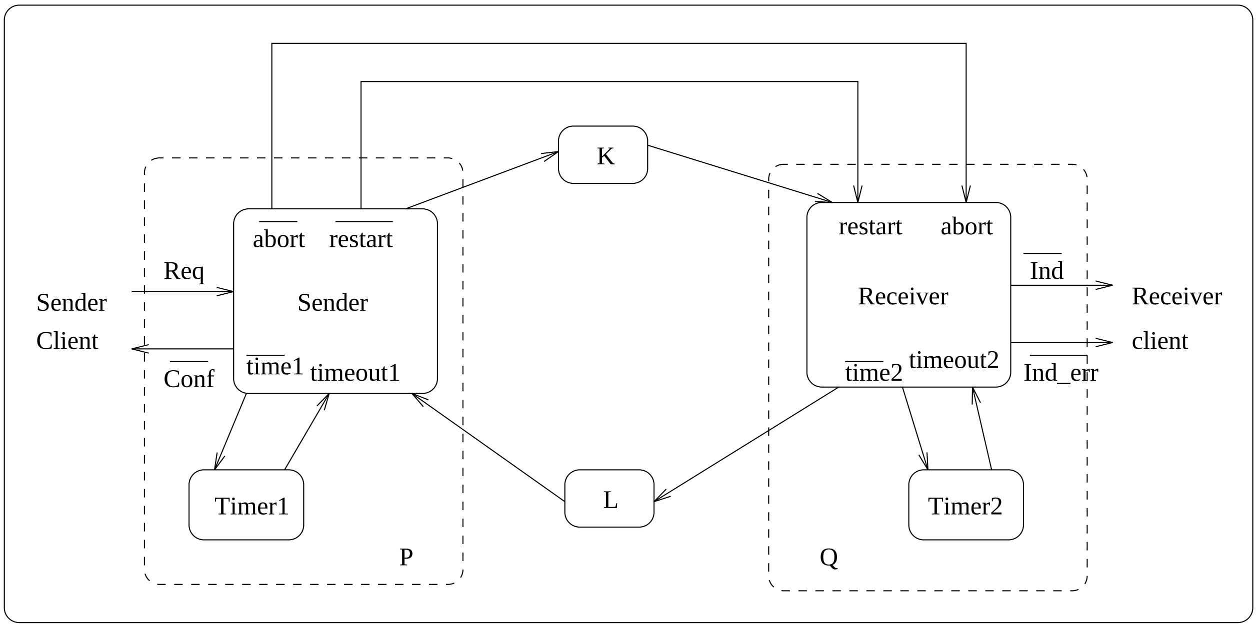

FIG. 6 - *Modélisation du BRP dans le π-calcul*

Comportement de l'émetteur L'émetteur effectue ses initialisations de variables et se met en attente d'une requête de transmission de fichier sur le canal *Req* de la part du producteur.

$$Sender(K, L, abort, restart) \stackrel{def}{=} [\![True]\!]first.[\![True]\!]tag.[\![0]\!]rn.[\![False]\!]last.$$
$$Attente\_req(K, L, abort, restart, first, last, tag, rn)$$

Quand l'émetteur reçoit une requête, il transfère le fichier.

$$Attente\_req(K, L, abort, restart, first, last, tag, rn) \stackrel{def}{=}$$
$$Req(f).Transfert(K, L, abort, restart, f, first, last, tag, rn)$$

À ce niveau, deux situations sont possibles:

- Soit le fichier reçu est vide, l'émetteur envoie tout de suite une confirmation OK au producteur. Ensuite, il envoie le signal $restart$ au récepteur et revient à son état initial.
- Soit le fichier est non vide, l'émetteur transmet au récepteur le message de tête du fichier accompagné des informations $first$ (premier message du fichier), $last$ (dernier message du fichier) et tag (valeur du bit alterné). Il déclenche ensuite $timer1$, incrémente le nombre de retransmissions du message (rn) et attend un acquittement de la part du récepteur sur le canal L.

$$Transfert(K,L,abort,restart,f,first,last,tag,rn) \stackrel{def}{=}$$
$$[f = Nil]\ \overline{Conf}\ \ OK.\overline{restart}.Sender(K,L,abort,restart)$$
$$+\ [f = cons(head,tail)]\ \ dernier(tail,last).\overline{K}\ \ first\ \ last\ \ tag\ \ head.\overline{time1}.$$
$$add(rn,1,rn).Attente\_ack(K,L,abort,restart,head,tail,first,last,tag,rn)$$

La fonction *dernier* permet de positionner la variable *last* selon qu'on est en train de traiter le dernier message du fichier ou pas.

$$dernier(tail,last) \stackrel{def}{=}$$
$$[tail = Nil]\ \ [\![True]\!]last$$
$$+\ [tail = cons(h,t)]\ \ [\![False]\!]last$$

Deux cas sont possibles:

- Soit un acquittement est reçu, l'émetteur désactive $timer1$, réinitialse rn, inverse le bit alterné et transfère le message suivant du fichier.
- Soit aucun acquittement n'est reçu (message ou acquittement perdu), $timer1$ expire. Dans ce cas, l'émetteur retransmet le message. Si le nombre maximum de retransmissions est atteint, l'émetteur envoie une confirmation $DONTKNOW$ ou $NOTOK$ au producteur selon que l'abort a eu lieu pendant le transfert du dernier message du fichier ou pendant le transfert d'un message intermédiaire du fichier. Ensuite, il envoie le signal *abort* au récepteur pour l'avertir et revient à son état initial.

$$Attente\_ack(K,L,abort,restart,head,tail,first,last,tag,rn) \stackrel{def}{=}$$
$$L.timeout1.[\![0]\!]rn.Not(tag).[\![False]\!]first.$$
$$Transfert(K,L,abort,restart,tail,first,last,tag,rn)$$
$$+\ timeout1.Retransmission(K,L,abort,restart,cons(head,tail),first,last,tag,rn)$$

$$Retransmission(K,L,abort,restart,f,first,last,tag,rn) \stackrel{def}{=}$$
$$If\ \ equal(rn,MAX)\ \ then$$
$$([last = True]\ \ \overline{Conf}\ \ DONTKNOW.\overline{abort}.Sender(K,L,abort,restart)$$
$$+\ [last = False]\ \ \overline{Conf}\ \ NOTOK.\overline{abort}.Sender(K,L,abort,restart))$$
$$else\ \ Transfert(K,L,abort,restart,f,first,last,tag,rn)$$

Si le fichier a été complètement transmis, l'émetteur envoie une confirmation OK au producteur, puis envoie le signal *restart* au récepteur pour que ce dernier anticipe l'expiration de $timer2$. Ensuite, l'émetteur revient à son état initial.

Comportement du récepteur Le récepteur fait ses initialisations et se met en attente de messages.

$$Receiver(K,L,abort,restart) \stackrel{def}{=} [\![False]\!]rtag.[\![False]\!]end.[\![False]\!]t2enabled.$$
$$Reception(K,L,abort,restart,rtag,end,t2enabled)$$

Le récepteur peut recevoir un message du fichier de la part de l'émetteur, le signal $timeout2$ d'expiration de son timer si $timer2$ est déclenché ou le signal *restart* pour se réinitialiser si le fichier qui doit être transféré par l'émetteur est vide. Ensuite, il réalise le traitement approprié.

$$
\begin{array}{l}
Reception(K,L,abort,restart,rtag,end,t2enabled) \stackrel{def}{=} \\
\quad K(first\ \ last\ \ tag\ \ m).Traitement(K,L,abort,restart,first,last,tag,m,rtag,end,t2enabled) \\
\quad +\ If\ \ equal\_bool(t2enabled,True)\ \ then \\
\qquad timeout2.abort.Abort(K,L,abort,restart,end) \\
\quad +\ restart. \\
\qquad If\ \ equal\_bool(t2enabled,True)\ \ then \\
\qquad\qquad timeout2.Receiver(K,L,abort,restart) \\
\qquad else\ \ Receiver(K,L,abort,restart)
\end{array}
$$

- Quand un message est reçu:
  - s'il s'agit d'une retransmission (le récepteur le sait en comparant le *tag* du message reçu au *tag* du message précédent qu'il mémorise dans *rtag*), il envoie un acquittement sur le canal L et se remet en attente de nouveau message.
  - S'il s'agit d'un nouveau message, le récepteur mémorise le *tag* associé et si ce n'est pas le premier message du fichier, il désactive $timer2$ ($timer2$ n'est pas encore déclenché quand le premier message arrive). Ensuite, il envoie une indication au consommateur. Cette indication peut être $FIRST$, $LAST$ ou $INCOMPLETE$ correspondant respectivement au premier message, dernier message ou à un message intermédiaire du fichier. Après, le récepteur envoie un acquittement sur le canal L, active $timer2$ et se remet en attente du message suivant du fichier.

$$
\begin{array}{l}
Traitement(K,L,abort,restart,first,last,tag,m,rtag,end,t2enabled) \stackrel{def}{=} \\
\quad If\ \ equal\_bool(tag,rtag)\ \ then \\
\qquad \overline{L}.Reception(K,L,abort,restart,rtag,end,t2enabled) \\
\quad else \\
\qquad If\ \ equal\_bool(first,True)\ \ then \\
\qquad\qquad copy\_bool(tag,rtag).Indication(K,L,abort,restart,first,last,m,rtag,end,t2enabled) \\
\qquad else\ \ copy\_bool(tag,rtag).timeout2. \\
\qquad\qquad\qquad Indication(K,L,abort,restart,first,last,m,rtag,end,t2enabled)
\end{array}
$$

$$
\begin{array}{l}
Indication(K,L,abort,restart,first,last,m,rtag,end,t2enabled) \stackrel{def}{=} \\
\qquad If\ \ equal\_bool(last,True)\ \ then \\
\qquad\quad [\![True]\!]end.\overline{Ind}\ \ m\ \ LAST.\overline{L}.\overline{time2}.[\![True]\!]t2enabled. \\
\qquad\qquad Reception(K,L,abort,restart,rtag,end,t2enabled) \\
\qquad else \\
\qquad\quad If\ \ equal\_bool(first,True)\ \ then \\
\qquad\qquad \overline{Ind}\ \ m\ \ FIRST.\overline{L}.\overline{time2}.[\![True]\!]t2enabled. \\
\qquad\qquad\quad Reception(K,L,abort,restart,rtag,end,t2enabled) \\
\qquad\quad else\ \ \overline{Ind}\ \ m\ \ INCOMPLETE.\overline{L}.\overline{time2}.[\![True]\!]t2enabled. \\
\qquad\qquad Reception(K,L,abort,restart,rtag,end,t2enabled)
\end{array}
$$

- En cas d'abort du transfert par l'émetteur, le récepteur reçoit le signal $timeout2$ et réalise un rendez-vous avec l'émetteur sur le canal *abort*. Il peut ensuite émettre une indication d'erreur au consommateur si l'abort n'a pas eu lieu lors du transfert du dernier message du fichier ou dans le cas où l'abort a eu lieu lors du transfert du dernier message et que ce message n'est jamais reçu par le récepteur. Ensuite, il revient à son état initial.

$$
\begin{aligned}
&Abort(K,L,abort,restart,end) \stackrel{def}{=} \\
&\quad If\ \ equal\_bool(end,True)\ \ then \\
&\qquad Receiver(K,L,abort,restart) \\
&\quad else \\
&\qquad \overline{Ind\_err}.Receiver(K,L,abort,restart)
\end{aligned}
$$

- Si au contraire, le fichier a été complètement transmis, le récepteur reçoit le signal $restart$. Dans ce cas, il désactive $timer2$ s'il est déclenché (il n'est pas déclenché dans le cas où l'émetteur a reçu une requête de transfert d'un fichier vide) et revient à son état initial.

On peut remarquer que la configuration du système ne change pas pendant l'exécution du protocole (i.e, les liaisons entre les processus restent inchangées).

Si on récapitule, on obtient les définitions suivantes:

$$
\begin{aligned}
&Sender(K,L,abort,restart) \stackrel{def}{=} [\![True]\!]first.[\![True]\!]tag.[\![0]\!]rn.[\![False]\!]last. \\
&\quad Attente\_req(K,L,abort,restart,first,last,tag,rn)
\end{aligned}
$$

$$
\begin{aligned}
&Attente\_req(K,L,abort,restart,first,last,tag,rn) \stackrel{def}{=} \\
&\quad Req(f).Transfert(K,L,abort,restart,f,first,last,tag,rn)
\end{aligned}
$$

$$
\begin{aligned}
&Transfert(K,L,abort,restart,f,first,last,tag,rn) \stackrel{def}{=} \\
&\quad [f=Nil]\ \ \overline{Conf}\ \ OK.\overline{restart}.Sender(K,L,abort,restart) \\
&\quad +\ [f=cons(head,tail)]\ \ dernier(tail,last).\overline{K}\ \ first\ \ last\ \ tag\ \ head.\overline{time1}. \\
&\qquad add(rn,1,rn).Attente\_ack(K,L,abort,restart,head,tail,first,last,tag,rn)
\end{aligned}
$$

$$
\begin{aligned}
&dernier(tail,last) \stackrel{def}{=} \\
&\qquad [tail=Nil]\ \ [\![True]\!]last \\
&\qquad +\ [tail=cons(h,t)]\ \ [\![False]\!]last
\end{aligned}
$$

$$
\begin{aligned}
&Attente\_ack(K,L,abort,restart,head,tail,first,last,tag,rn) \stackrel{def}{=} \\
&\quad L.timeout1.[\![0]\!]rn.Not(tag).[\![False]\!]first. \\
&\qquad Transfert(K,L,abort,restart,tail,first,last,tag,rn) \\
&\quad +\ timeout1.Retransmission(K,L,abort,restart,cons(head,tail),first,last,tag,rn)
\end{aligned}
$$

$$
\begin{aligned}
&Retransmission(K,L,abort,restart,f,first,last,tag,rn) \stackrel{def}{=} \\
&\quad If\ \ equal(rn,MAX)\ \ then \\
&\qquad ([last=True]\ \ \overline{Conf}\ \ DONTKNOW.\overline{abort}.Sender(K,L,abort,restart) \\
&\qquad +\ [last=False]\ \ \overline{Conf}\ \ NOTOK.\overline{abort}.Sender(K,L,abort,restart)) \\
&\quad else\ \ Transfert(K,L,abort,restart,f,first,last,tag,rn)
\end{aligned}
$$

$$Receiver(K,L,abort,restart) \stackrel{def}{=} [\![False]\!]rtag.[\![False]\!]end.[\![False]\!]t2enabled.$$
$$\quad Reception(K,L,abort,restart,rtag,end,t2enabled)$$

$$\begin{array}{l}
Reception(K,L,abort,restart,rtag,end,t2enabled) \stackrel{def}{=} \\
\quad K(first\ \ last\ \ tag\ \ m).Traitement(K,L,abort,restart,first,last,tag,m,rtag,end,t2enabled) \\
\quad +\ If\ \ equal_bool(t2enabled,True)\ \ then \\
\quad\quad timeout2.abort.Abort(K,L,abort,restart,end) \\
\quad +\ restart. \\
\quad\quad If\ \ equal_bool(t2enabled,True)\ \ then \\
\quad\quad\quad timeout2.Receiver(K,L,abort,restart) \\
\quad\quad else\ \ Receiver(K,L,abort,restart)
\end{array}$$

$$\begin{array}{l}
Traitement(K,L,abort,restart,first,last,tag,m,rtag,end,t2enabled) \stackrel{def}{=} \\
\quad If\ \ equal_bool(tag,rtag)\ \ then \\
\quad\quad \overline{L}.Reception(K,L,abort,restart,rtag,end,t2enabled) \\
\quad else \\
\quad\quad If\ \ equal_bool(first,True)\ \ then \\
\quad\quad\quad copy_bool(tag,rtag).Indication(K,L,abort,restart,first,last,m,rtag,end,t2enabled) \\
\quad\quad else\ \ copy_bool(tag,rtag).timeout2. \\
\quad\quad\quad\quad Indication(K,L,abort,restart,first,last,m,rtag,end,t2enabled)
\end{array}$$

$$\begin{array}{l}
Indication(K,L,abort,restart,first,last,m,rtag,end,t2enabled) \stackrel{def}{=} \\
\quad If\ \ equal_bool(last,True)\ \ then \\
\quad\quad [\![True]\!]end.\overline{Ind}\ \ m\ \ LAST.\overline{L}.\overline{time2}.[\![True]\!]t2enabled. \\
\quad\quad\quad Reception(K,L,abort,restart,rtag,end,t2enabled) \\
\quad else \\
\quad\quad If\ \ equal_bool(first,True)\ \ then \\
\quad\quad\quad \overline{Ind}\ \ m\ \ FIRST.\overline{L}.\overline{time2}.[\![True]\!]t2enabled. \\
\quad\quad\quad\quad Reception(K,L,abort,restart,rtag,end,t2enabled) \\
\quad\quad else\ \ \overline{Ind}\ \ m\ \ INCOMPLETE.\overline{L}.\overline{time2}.[\![True]\!]t2enabled. \\
\quad\quad\quad Reception(K,L,abort,restart,rtag,end,t2enabled)
\end{array}$$

$$\begin{array}{l}
Abort(K,L,abort,restart,end) \stackrel{def}{=} \\
\quad\quad If\ \ equal_bool(end,True)\ \ then \\
\quad\quad\quad Receiver(K,L,abort,restart) \\
\quad\quad else \\
\quad\quad\quad \overline{Ind_err}.Receiver(K,L,abort,restart)
\end{array}$$

$$Timer1 \stackrel{def}{=} time1.\overline{timeout1}.Timer1$$

$$Timer2 \stackrel{def}{=} time2.\overline{timeout2}.Timer2$$

$$P(K,L,abort,restart) \stackrel{def}{=} (\nu\ \ time1\ \ timeout1)\ \ (Sender(K,L,abort,restart) \mid Timer1)$$

$$Q(K,L,abort,restart) \stackrel{def}{=} (\nu\ \ time2\ \ timeout2)\ \ (Receiver(K,L,abort,restart) \mid Timer2)$$

$$System \equiv (\nu\ \ K\ \ L\ \ abort\ \ restart)\ \ (P(K,L,abort,restart) \mid Q(K,L,abort,restart))$$

5.4 Exécution du protocole

Nous allons exécuter le protocole pour transférer un fichier de 3 messages. Supposons que les messages sont de type entier. De plus, supposons qu'aucune perte de message ou d'acquittement ne se produit. Le point de départ est le système à l'état initial:

$$System \equiv (\nu\ K\ L\ abort\ restart)(P(K, L, abort, restart) \mid Q(K, L, abort, restart))$$

$$\begin{aligned} System \equiv\ & (\nu\ K\ L\ abort\ restart) \\ & ((\nu\ time1\ timeout1)(Sender(K, L, abort, restart) \mid Timer1) \\ & \mid (\nu\ time2\ timeout2)\ (Receiver(K, L, abort, restart) \mid Timer2)) \end{aligned}$$

Au démarrage, l'émetteur fait ses initialisations de variables et se met en attente de fichier sur le canal Req; le récepteur effectue ses initialisations et se met en attente de message sur le canal K.

La première action est une action externe qui consiste à émettre le fichier cons(1,cons(2,cons(3,Nil))) sur le canal Req par le producteur. La deuxième action est la réception du fichier par l'émetteur sur le canal Req. La situation du système après ces 2 actions est la suivante:

$$\begin{aligned} & (\nu\ K\ L\ abort\ restart) \\ & \quad (Transfert(K, L, abort, restart, cons(1, cons(2, cons(3, Nil))), True, False, True, 0) \\ & \quad \mid Reception(K, L, abort, restart, False, False, False)) \end{aligned}$$

Maintenant, l'émetteur transmet le premier message ($True$,$False$,$True$,1) sur le canal K (i.e, $first$ vaut $True$, $last$ vaut $False$, le bit alterné vaut $True$ et la donnée vaut 1). Ensuite, il active $timer1$ et incrémente le nombre de retransmissions (qui vaut maintenant 1). Enfin, l'émetteur se met en attente d'acquittement sur le canal L.

Le récepteur reçoit le message ($True$,$False$,$True$,1) sur le canal K. Sachant qu'il s'agit d'un nouveau message (i.e, le bit alterné reçu est différent du bit alterné local qui vaut $False$), le récepteur copie $True$ dans son bit alterné. Ensuite, puisque $first$ vaut $True$, il envoie une indication (1,$FIRST$) sur le canal Ind, envoie un acquittement sur le canal L, déclenche $timer2$ et met à $True$ la variable $t2enabled$. Enfin, il se met en attente du message suivant sur K.

L'émetteur reçoit alors un acquittement sur le canal L ($timer1$ est toujours actif). Il arrête $timer1$, remet à 0 le nombre de retransmissions, inverse son bit alterné (il vaut maintenant $False$) et met $False$ dans la variable $first$. Ensuite, l'émetteur s'apprête à transmettre le message suivant du fichier.

La nouvelle situation est la suivante:

$$\begin{aligned} & (\nu\ K\ L\ abort\ restart) \\ & \qquad (Transfert(K, L, abort, restart, cons(2, cons(3, Nil)), False, False, False, 0) \\ & \qquad \mid Reception(K, L, abort, restart, True, False, True)) \end{aligned}$$

Le même scénario se reproduit pour le deuxième message ($False$,$False$,$False$,2) du fichier. Mais cette fois-ci, le récepteur copie la valeur $False$ dans $rtag$, désactive $timer2$ qui est maintenant déclenché puisque $t2enabled$ vaut $True$ et envoie une indication (2,$INCOMPLETE$) sur la canal Ind car il s'agit d'un message intermédiaire du fichier. Ensuite, l'acquittement est transmis à l'émetteur et $timer2$ est redéclenché.

On se retrouve dans la situation suivante:

$$
\begin{aligned}
&(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\qquad\qquad (Transfert(K, L, abort, restart, cons(3, Nil), False, False, True, 0)\\
&\qquad\qquad \mid Reception(K, L, abort, restart, False, False, True))
\end{aligned}
$$

Le même scénario se reproduit pour le troisième message ($False$,$True$,$True$,3) du fichier sauf que cette fois-ci, il s'agit du dernier message ($last$ est positionné à $True$). Quand le récepteur reçoit ce message, il envoie une indication (3,$LAST$) sur le canal Ind et positionne end à $True$. L'acquittement est transmis à l'émetteur et $timer2$ est redéclenché.

On se retrouve dans la situation suivante:

$$
\begin{aligned}
&(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\qquad\qquad (Transfert(K, L, abort, restart, Nil, False, True, False, 0)\\
&\qquad\qquad \mid Reception(K, L, abort, restart, True, True, True))
\end{aligned}
$$

L'action suivante est faite par l'émetteur: sachant que maintenant le fichier est vide, il envoie une confirmation OK sur le canal $Conf$. Ensuite, il envoie un signal sur le canal $restart$ pour autoriser le récepteur à anticiper l'expiration de $timer2$ et se met en attente d'une nouvelle requête de transmission de fichier sur le canal Req.

Quant au récepteur, après la livraison du dernier message, il reçoit le signal $restart$. Ensuite, il arrête $timer2$ et se remet en attente d'un nouveau message sur le canal K.

L'émetteur et le récepteur sont ainsi synchronisés pour le transfert d'un nouveau fichier. La situation finale est la suivante:

$$(\nu\ \ K\ \ L\ \ abort\ \ restart)(P(K, L, abort, restart) \mid Q(K, L, abort, restart))$$

Appelons ce processus $System'$. Comme on peut le remarquer, le processus $System'$ résultant d'une exécution correcte (sans perte de message ou d'acquittement) du protocole est équivalent au processus de départ $System$.

Ainsi, on a démontré qu'une exécution correcte du protocole, partant de l'état initial du système, reconduit à cet état intial. Ce résultat est encore valable dans le cas où des pertes de messages ou d'acquittements se produisent et sera prouvé dans la section 7.

6 Les lois algébriques du π-calcul

La notion d'équivalence entre 2 agents est basée intuitivement sur l'idée qu'on peut les distinguer si une différence peut être détectée par un troisième agent (externe) interagissant avec eux. Ce troisième agent peut être considéré comme nous-mêmes (l'observateur).

6.1 La notion de bisimulation

- Une relation binaire S est une *simulation* entre processus si $P\ S\ Q$ (i.e, P peut être simulé par Q) implique:

  si $P \stackrel{\alpha}{\rightarrow} P'$ et α est une action libre (i.e, ne lie aucun nom), alors $\exists\ Q'$ tel que $Q \stackrel{\alpha}{\rightarrow} Q'$ et $P'\ S\ Q'$.

  Ceci signifie que toute transition à partir de P doit être simulée par une transition à partir de Q telle que le processus dérivé P' peut être simulé par le processus dérivé Q'.

- Une relation binaire S sur les processus est une *bisimulation* si à la fois S et son inverse S^{-1} sont des simulations.

- Une relation binaire S est une *simulation forte* entre processus si $P\ S\ Q$ implique:

  1. si $P \stackrel{\alpha}{\rightarrow} P'$ et α est une action libre, alors $\exists\ Q'$ tel que $Q \stackrel{\alpha}{\rightarrow} Q'$ et $P'\ S\ Q'$ (i.e, S est une simulation).
  2. si $P \stackrel{x(y)}{\rightarrow} P'$ et $y \notin n(P, Q)$, alors $\exists\ Q'$ tel que $Q \stackrel{x(y)}{\rightarrow} Q'$ et $\forall\ w$, $P'\{w/y\}\ S\ Q'\{w/y\}$.
  3. si $P \stackrel{\overline{x}(y)}{\rightarrow} P'$ et $y \notin n(P, Q)$, alors $\exists\ Q'$ tel que $Q \stackrel{\overline{x}(y)}{\rightarrow} Q'$ et $P'\ S\ Q'$.

- Une relation binaire S est une *bisimulation forte* entre processus si à la fois S et son inverse S^{-1} sont des simulations fortes.

6.2 L'équivalence forte close ($\dot{\sim}$)

La relation $\dot{\sim}$ correspond à une équivalence où le nombre d'actions internes est important. L'action interne τ est donc traitée comme toute autre action. Ainsi, on peut distinguer le processus $\tau.\tau.0$ du processus $\tau.0$.

Deux processus P et Q sont *fortement équivalents* ($P \dot{\sim} Q$) si $\exists$ une bisimulation forte S telle que $(P, Q) \in S$.
$\dot{\sim} = \cup\ \{S\ /\ S$ est une bisimulation forte$\}$. Donc, $\dot{\sim}$ est la plus grande bisimulation forte. $\dot{\sim}$ est aussi une relation d'équivalence.

Les lois algébriques vérifiées par $\dot{\sim}$ sont décrites ci-dessous. Le symbôle $=$ est utilisé au lieu de $\dot{\sim}$ afin de permettre d'autres interprétations possibles de ces règles.

- (A) $P \equiv Q \vdash P = Q$ (α-conversion)

- (C0) $P = Q \vdash$:

  $\tau.P = \tau.Q$

  $P + R = Q + R$

  $(\nu\ x)P = (\nu\ x)Q$

  $\overline{x}y.P = \overline{x}y.Q$

  $P \mid R = Q \mid R$

  $[x = y]P = [x = y]Q$

  (i.e, tous les constructeurs, sauf $x(y)$, préservent $=$)

- (C1) $x(y).P = x(y).Q$ ssi $P\{z/y\} = Q\{z/y\}, \forall\ z$

- (S0) $P + 0 = P$

  (S1) $P + P = P$

  (S2) $P + Q = Q + P$

  (S3) $P + (Q + R) = (P + Q) + R$

- (R0) $(\nu\ x)P = P$ (si $x \notin$ fn(P))

  (R1) $(\nu\ x)(\nu\ y)P = (\nu\ y)(\nu\ x)P$

  (R2) $(\nu\ x)(P + Q) = (\nu\ x)P + (\nu\ x)Q$

  (R3) $(\nu\ x)\alpha.P = \alpha.(\nu\ x)P$ (si $x \notin fn(\alpha)$)

  (R4) $(\nu\ x)\alpha.P = 0$ (si x est le sujet de α)

- (M0) $[x = y]P = 0$ si $x \neq y$

  (M1) $[x = x]P = P$

- (I) $A(\tilde{y}) = P\{\tilde{y}/\tilde{x}\}$ si $A(\tilde{x}) \stackrel{def}{=} P$

- (P0) $P \mid 0 = P$

  (P1) $P \mid Q = Q \mid P$

  (P2) $P \mid (Q \mid R) = (P \mid Q) \mid R$

  (P3) $(\nu\ x)(P \mid Q) = P \mid (\nu\ x)Q$ (si x $\notin$ fn(P))

- (E) Soient $P = \sum_i \alpha_i.P_i$, $Q = \sum_j \beta_j.Q_j$ tels que $bn(\alpha_i) \cap fn(Q) = \emptyset\ \forall\ i$ et $bn(\beta_j) \cap fn(P) = \emptyset\ \forall\ j$, alors

  $P \mid Q = \sum_i \alpha_i.(P_i \mid Q) + \sum_j \beta_j.(P \mid Q_j) + \sum_{\alpha_i comp \beta_j} \tau.R_{ij}$

  où la relation α_i *comp* β_j (i.e, complémente) est vraie dans les cas suivants qui définissent R_{ij}:

  1. si $\alpha_i = \overline{x}u$ et $\beta_j = x(v)$ alors R_{ij} est $P_i \mid Q_j\{u/v\}$

2. si $\alpha_i = \overline{x}(u)$ et $\beta_j = x(v)$ alors R_{ij} est $(\nu\ w)(P_i\{w/u\} \mid Q_j\{w/v\})$ (w est non libre dans $(\nu\ u)P_i$ et dans $(\nu\ v)Q_j$)
3. si $\alpha_i = x(v)$ et $\beta_j = \overline{x}u$ alors R_{ij} est $P_i\{u/v\} \mid Q_j$
4. si $\alpha_i = x(v)$ et $\beta_j = \overline{x}(u)$ alors R_{ij} est $(\nu\ w)(P_i\{w/v\} \mid Q_j\{w/u\})$ (w est non libre dans $(\nu\ v)P_i$ et dans $(\nu\ u)Q_j$)

L'équivalence $\dot\sim$ n'est pas une congruence. En effet, pour prouver que $x(y).P = x(y).Q$, on doit prouver à l'aide de la règle (C1) que P et Q sont équivalents pour toutes les instanciations de y. L'équivalence $\dot\sim$ n'est pas préservée, en général, par les substitutions des noms libres.

Les preuves de ces propriétés, basées sur une sémantique opérationnelle du π-calcul, sont décrites dans [MPW89b].

6.3 L'équivalence faible close ($\dot\simeq$)

Cette équivalence appelée aussi *équivalence observationnelle* ignore les transitions silencieuses τ. Pour prouver que 2 agents sont équivalents, une action τ (non observable par un agent externe) dans l'un des 2 agents peut être simulée par 0 ou plusieurs actions τ dans l'autre agent. Les actions internes ne sont importantes que quand elles empêchent d'autres actions de se produire. Ainsi, $\tau.\tau.0 \dot\simeq \tau.0$.

Comme $\dot\sim$, l'équivalence $\dot\simeq$ n'est pas préservée par le préfixe positif $x(y)$ puisqu'elle n'est pas préservée par les substitutions. Et contrairement à $\dot\sim$, elle n'est pas préservée par la somme de processus.

Deux processus P et Q sont *observationnellement équivalents* (ou faiblement équivalents) $P \dot\simeq Q$ si $(P, Q) \in S$ pour une bisimulation faible S.

$\dot\simeq = \cup\ \{S\ /\ S$ est une bisimulation faible$\}$. Donc, $\dot\simeq$ est la plus grande bisimulation faible. $\dot\simeq$ est aussi une relation d'équivalence.

$\dot\simeq$ est strictement faible que $\dot\sim$ et satisfait les mêmes lois algébriques. De plus, elle satisfait les τ-lois [Mil89] suivantes:

- (T0) $\alpha.\tau.P \dot\simeq \alpha.P$
- (T1) $P + \tau.P \dot\simeq \tau.P$
- (T2) $\alpha.(P + \tau.Q) + \alpha.Q \dot\simeq \alpha.(P + \tau.Q)$.

Afin d'éliminer les boucles τ dans les processus définis récursivement, on introduit la loi suivante [Mil89]:

- (L) si $A = P + \tau.A$ et $B = \tau.P$ alors $A \dot\simeq B$

Ci-dessous un exemple qui illustre que $\dot{\sim}$ et $\dot{\simeq}$ ne sont pas préservées, en général, par les substitutions des noms libres:
$\overline{x}u.0 \mid y(w).0 \mathrel{\dot{\sim}} \overline{x}u.y(w).0 + y(w).\overline{x}u.0$

Mais, si on substitue x par y, l'équivalence n'est plus vérifiée car dans ce cas, une communication devient possible le long de y. Par contre,
$\overline{y}u.0 \mid y(w).0 \mathrel{\dot{\sim}} \overline{y}u.y(w).0 + y(w).\overline{y}u.0 + \tau.0$, ce qui explique la condition d'application de la règle (C1).

6.4 L'équivalence forte non close ($\sim$)

C'est l'extension de l'équivalence forte close à toutes les substitutions de noms.
$P \sim Q$ ssi $P\sigma \mathrel{\dot{\sim}} Q\sigma$, $\forall\ \sigma$ une substitution de noms à des noms.

Si on reprend l'exemple précédent, $\overline{x}u.0 \mid y(w).0 \sim \overline{x}u.y(w).0 + y(w).\overline{x}u.0 + [x = y]\tau.0$ où $[x = y]\tau.0$ permet de réaliser la communication le long du canal y dans le cas où on substitue x par y et de préserver donc l'équivalence.

Les définitions récursives de processus préservent $\sim$ (ceci n'est pas valable pour $\dot{\sim}$). La bisimulation forte non close $\sim$ est ainsi préservée par tous les contextes. Donc, $\sim$ est une congruence.

6.5 L'équivalence faible non close ($\simeq$)

De façon analogue, on définit la relation $\simeq$.
$P \simeq Q$ ssi $P\sigma \mathrel{\dot{\simeq}} Q\sigma$, $\forall\ \sigma$ une substitution de noms à des noms.

Comme $\dot{\simeq}$, $\simeq$ n'est pas préservée par la somme de processus (voir l'exemple qui suit). La bisimulation faible non close $\simeq$ n'est donc pas une congruence.

On a $x(y) \simeq \tau.x(y)$. Soient $P = x(y) + u(v)$ et $Q = \tau.x(y) + u(v)$. Alors, $P \not\simeq Q$. En effet, l'action non observable τ n'est pas contrôlable par l'environnement. Le processus Q peut exécuter l'action τ de manière autonome et perdre la possibilité d'exécuter l'action $u(v)$; tandis que le processus P préserve cette possibilité. Donc, dans un environnement qui demande l'action $u(v)$, P n'entrera pas en situation de deadlock alors que Q peut se retrouver en situation de deallock. Ainsi, l'action non observable τ peut affecter l'observation des actions visibles.

6.6 Les lois algébriques pour les définitions récursives

Les équivalences non closes sont utilisées dans les lois manipulant les processus définis récursivement [Mil89].

Soient E, F, ... des expressions de processus contenant des variables qui peuvent être remplacées par des processus. $E(P_1, ..., P_n)$ est le processus résultant du remplacement dans E des variables $X_1, ..., X_n$ par les processus $P_1, ..., P_n$.

Deux agents E et F sont (fortement/faiblement) équivalents si $E(\widetilde{P})$ et $F(\widetilde{P})$ sont (fortement/faiblement) équivalents, $\forall\ \widetilde{P} = P_1, ..., P_n$.

Les lois suivantes sont applicables à des processus définis récursivement:

- première loi pour la récursion:

  Si la partie droite des définitions d'un processus est transformée en respectant l'équivalence (i.e, en remplaçant des processus par d'autres processus équivalents), alors la transformation obtenue est équivalente au processus défini.

  (U0) Soient $E_1, ..., E_n$ et $F_1, ..., F_n$ des expressions de processus. Soient $A_1, ..., A_n$ et $B_1, ..., B_n$ des identificateurs de processus tels que $\forall\ i$:

  $E_i = F_i$ et $A_i(\widetilde{x_i}) = E_i(A_1, ..., A_n)$ et $B_i(\widetilde{x_i}) = F_i(B_1, ..., B_n)$

  alors $A_i(\widetilde{x_i}) = B_i(\widetilde{x_i})$, $\forall\ i$.

  Cette loi est valable pour les deux équivalences non closes (forte et faible). Cependant, elle n'est pas valable pour les équivalences closes.

- deuxième loi pour la récursion:

  Si deux processus satisfont un système d'équations récursives vérifiant les conditions ci-dessous, alors ces deux processus sont équivalents.

  (U1) Soient $E_1, ..., E_n$ des expressions de processus. Soient $P_1, ..., P_n$ et $Q_1, ..., Q_n$ des processus tels que $\forall\ i$:

  $P_i = E_i(P_1, ..., P_n)$ et $Q_i = E_i(Q_1, ..., Q_n)$

  alors $P_i = Q_i$, $\forall\ i$.

  Cette loi est valable pour l'équivalence forte non close $\sim$ sachant que les expressions $E_1, ..., E_n$ sont faiblement gardées (i.e, toutes les occurrences de P_j dans $E_i(P_1, ..., P_n)$ sont préfixées par un opérateur input, output ou τ).

  Elle est valable aussi pour l'équivalence faible non close $\simeq$ sachant que les expressions $E_1, ..., E_n$ sont gardées (i.e, toutes les occurrences de P_j dans $E_i(P_1, ..., P_n)$ sont préfixées par un opérateur input ou output) et séquentielles (i.e, aucune des expressions E_i ne contient une composition parallèle).

  Autrement-dit, **un système d'équations récursives gardées et séquentielles (respectivement faiblement gardées) possède une solution unique modulo l'équivalence faible (respectivement forte) non close**. Donc, si deux processus satisfont les équations du système (i.e, sont solutions) alors ils sont forcément équivalents.

7 Preuve formelle du protocole

Le but est de prouver formellement que l'implémentation du protocole ($System$) et sa vue abstraite (S_0) ont des comportements équivalents. Ce résultat est très important. En effet, un concepteur qui veut utiliser le protocole pour construire un système plus complexe n'aura pas besoin de considérer toute l'implémentation du protocole (i.e, $System$ et ses composantes P, Q, $perte\_msg$ et $perte\_ack$). Il lui suffit de comprendre et d'utiliser la spécification S_0 qui est simple mais qui fournit exactement le même comportement observable que l'implémentation. Ce point est très important dans le cadre d'un développement modulaire où plusieurs personnes travaillent sur un même projet alors que seules certaines d'entre elles connaissent les particularités du projet.

La preuve de l'équivalence ne signifie pas que le protocole est totalement correct. Mais, elle permet de déduire certaines propriétés désirables du protocole telles que **l'absence de deadlock**.

7.1 La méthode de preuve

On veut démontrer que le protocole est faiblement (ou observationnellement) équivalent à la spécification abstraite S_0. La procédure à suivre est la suivante:

1. appliquer au protocole, répétitivement, la loi d'expansion (E) ce qui permet d'établir toutes les exécutions possibles du protocole,
2. définir la spécification abstraite et le protocole par un système d'équations récursives,
3. simplifier les équations récursives définissant le protocole en utilisant les τ-lois, en identifiant et substituant les expressions équivalentes et en éliminant les boucles τ,
4. prouver que le protocole satisfait les équations définissant la spécification abstraite
5. et enfin, appliquer la loi (U1) qui dit que si 2 processus satisfont les mêmes équations récursives alors ils sont équivalents.

La preuve suivra cette méthode mais elle est compliquée par la présence de boucles τ dans le protocole. Or, la règle (U1) ne peut être appliquée si les équations ne sont pas gardées.

Dans le protocole, les boucles τ ne sont pas explicites, elles résultent des communications entre les composantes du système. Le premier travail consiste à préparer les équations avant d'éliminer ces boucles. La seule loi qui permet de les éliminer est la loi (L) qui est applicable uniquement à des processus ayant des équations de définition particulières (voir la section 6.3). On va donc utiliser l'équivalence forte jusqu'à ce que le système soit réécrit en une forme qui permet d'éliminer les boucles τ.

Le point de départ est la description formelle du protocole contenant les équations de définition de $System$ et de ses composantes P, Q, $perte\_msg$ et $perte\_ack$. On va appliquer de manière itérative la loi d'expansion (E) aux composantes du système pour générer de nouvelles équations récursives à l'aide de l'équivalence forte close.

L'avantage de cette technique est qu'elle est **modulaire**. En effet, on n'aura jamais à considérer toutes les composantes du système en même temps.

7.2 Traces d'exécution du protocole

Dans ce quit suit, τ^n désigne la séquence $\tau.\tau...\tau$ (n fois).

7.2.1 Expansion de la composante P

Commençons par l'expansion de la composante $P(K, L, abort, restart)$ afin d'établir toutes les étapes qu'elle peut exécuter.

$$P(K, L, abort, restart) \stackrel{def}{=} (\nu\ time1\ \ timeout1)(Sender(K, L, abort, restart) \mid Timer1)$$

On remplace $Sender(K, L, abort, restart)$ et $Timer1$ par leur définition.

$$\begin{aligned}\equiv\ (\nu\ time1\ \ timeout1)(&[\![True]\!]first.[\![True]\!]tag.[\![0]\!]rn.[\![False]\!]last.\\ &\quad Attente\_req(K, L, abort, restart, first, last, tag, rn)\\ &\mid time1.\overline{timeout1}.Timer1)\end{aligned}$$

L'action $[\![True]\!]first$ du $Sender$ n'est pas liée dans la composition parallèle. Le $Sender$ peut alors exécuter cette action de manière indépendante. Pour cela, on applique la loi d'expansion (E).

$$\begin{aligned}\dot\sim\ [\![True]\!]first.(\nu\ time1\ \ timeout1)(&[\![True]\!]tag.[\![0]\!]rn.[\![False]\!]last.\\ &\quad Attente\_req(K, L, abort, restart, first, last, tag, rn)\\ &\mid time1.\overline{timeout1}.Timer1)\end{aligned}$$

$$\begin{aligned}\dot\sim\ [\![True]\!]first.[\![True]\!]tag.&(\nu\ time1\ \ timeout1)\\ &([\![0]\!]rn.[\![False]\!]last.Attente\_req(K, L, abort, restart, first, last, tag, rn)\\ &\mid time1.\overline{timeout1}.Timer1)\end{aligned}$$

$$\begin{aligned}\dot\sim\ [\![True]\!]first.[\![True]\!]tag.[\![0]\!]rn.&(\nu\ time1\ \ timeout1)\\ &([\![False]\!]last.Attente\_req(K, L, abort, restart, first, last, tag, rn)\\ &\mid time1.\overline{timeout1}.Timer1)\end{aligned}$$

$$\begin{aligned}\dot\sim\ [\![True]\!]first.[\![True]\!]tag.[\![0]\!]rn.[\![False]\!]last.&(\nu\ time1\ \ timeout1)\\ &(Attente\_req(K, L, abort, restart, first, last, tag, rn)\\ &\mid time1.\overline{timeout1}.Timer1)\end{aligned}$$

Chaque action indépendante est réduite ici à une seule transition silencieuse τ. Normalement, plusieurs transitions silencieuses résultent de la réduction de chaque action. Un exemple de réduction peut être consulté dans la section 2.2.2.

$$\begin{aligned}\dot\sim\ \tau^4.(\nu\ time1\ \ timeout1)(&Attente\_req(K, L, abort, restart, first, last, tag, rn)\\ &\mid time1.\overline{timeout1}.Timer1)\end{aligned}$$

$$\dot{\sim}\ \tau^4.(\nu\ time1\ \ timeout1)(Req(f).Transfert(K,L,abort,restart,f,first,last,tag,rn) \\ \mid time1.\overline{timeout1}.Timer1)$$

$$\dot{\sim}\ \tau^4.Req(f).(\nu\ time1\ \ timeout1)(Transfert(K,L,abort,restart,f,first,last,tag,rn) \\ \mid time1.\overline{timeout1}.Timer1)$$

$$P(K,L,abort,restart)\ \dot{\sim}\ \tau^4.Req(f).PP(K,L,abort,restart,f,first,last,tag,rn)$$

Considérons la partie $PP(K,L,abort,restart,f,first,last,tag,rn)$:

$$PP(K,L,abort,restart,f,first,last,tag,rn) \stackrel{def}{=} (\nu\ time1\ \ timeout1) \\ (Transfert(K,L,abort,restart,f,first,last,tag,rn) \\ \mid time1.\overline{timeout1}.Timer1)$$

$$PP(K,L,abort,restart,f,first,last,tag,rn)\ \dot{\sim}\ (\nu\ time1\ \ timeout1) \\ (([f=Nil]\ \overline{Conf}\ \ OK.\overline{restart}.Sender(K,L,abort,restart) \\ +\ [f=cons(head,tail)]\ dernier(tail,last).\overline{K\ \ first\ \ last\ \ tag\ \ head}.\overline{time1}. \\ add(rn,1,rn).Attente\_ack(K,L,abort,restart,head,tail,first,last,tag,rn)) \\ \mid time1.\overline{timeout1}.Timer1)$$

Deux cas sont possibles:

1. Soit le fichier est vide, l'émetteur peut confirmer (OK) tout de suite, émet le signal $restart$ (i.e, demande au récepteur de se réinitialiser) et se remet en attente d'une nouvelle requête ($P_1(K,L,abort,restart)$).

2. Soit le fichier est non vide, il peut transférer le message de tête du fichier ($P_2(K,L,abort,restart,cons(head,tail),first,last,tag,rn)$).

$$\dot{\sim}\ (\tau.(\nu\ time1\ \ timeout1) \\ (\overline{Conf}\ \ OK.\overline{restart}.Sender(K,L,abort,restart) \\ \mid time1.\overline{timeout1}.Timer1) \\ +\ \tau.(\nu\ time1\ \ timeout1) \\ (dernier(tail,last).\overline{K\ \ first\ \ last\ \ tag\ \ head}.\overline{time1}. \\ add(rn,1,rn).Attente\_ack(K,L,abort,restart,head,tail,first,last,tag,rn) \\ \mid time1.\overline{timeout1}.Timer1))$$

L'action interne τ correspond à l'évaluation de l'expression $[f = Nil]$ ou à l'évaluation de l'expression $[f = cons(head,tail)]$.

$$PP(K,L,abort,restart,f,first,last,tag,rn)\ \dot{\sim}\ (P_1(K,L,abort,restart) \\ +\ P_2(K,L,abort,restart,cons(head,tail),first,last,tag,rn))$$

Considérons la partie $P_1(K,L,abort,restart)$ (fichier vide):

$$P_1(K,L,abort,restart) \stackrel{def}{=} \tau.(\nu\ time1\ \ timeout1) \\ (\overline{Conf}\ \ OK.\overline{restart}.Sender(K,L,abort,restart) \\ \mid time1.\overline{timeout1}.Timer1)$$

$$\dot\sim\ \tau.\overline{Conf\ \ OK}.(\nu\ \ time1\ \ timeout1)$$
$$(\overline{restart}.Sender(K, L, abort, restart)$$
$$\mid time1.\overline{timeout1}.Timer1)$$

$$\dot\sim\ \tau.\overline{Conf\ \ OK}.\overline{restart}.(\nu\ \ time1\ \ timeout1)$$
$$(Sender(K, L, abort, restart)$$
$$\mid Timer1)$$

$$P_1(K, L, abort, restart)\ \dot\sim\ \tau.\overline{Conf\ \ OK}.\overline{restart}.P(K, L, abort, restart)$$

Considérons la partie $P_2(K, L, abort, restart, cons(head, tail), first, last, tag, rn)$ (fichier non vide):

$$P_2(K, L, abort, restart, cons(head, tail), first, last, tag, rn) \stackrel{def}{=} \tau.(\nu\ \ time1\ \ timeout1)$$
$$(dernier(tail, last).\overline{K}\ \ first\ \ last\ \ tag\ \ head.\overline{time1}.$$
$$add(rn, 1, rn).Attente\_ack(K, L, abort, restart, head, tail, first, last, tag, rn)$$
$$\mid time1.\overline{timeout1}.Timer1)$$

Suite à l'exécution de la fonction $dernier(tail, last)$ qui produit deux transitions internes τ, $last$ vaut la valeur $True$ ou la valeur $False$.

$$\dot\sim\ \tau^3.(\nu\ \ time1\ \ timeout1)$$
$$(\overline{K}\ \ first\ \ last\ \ tag\ \ head.\overline{time1}.$$
$$add(rn, 1, rn).Attente\_ack(K, L, abort, restart, head, tail, first, last, tag, rn)$$
$$\mid time1.\overline{timeout1}.Timer1)$$

$$\dot\sim\ \tau^3.\overline{K}\ \ first\ \ last\ \ tag\ \ head.(\nu\ \ time1\ \ timeout1)$$
$$(\overline{time1}.add(rn, 1, rn).Attente\_ack(K, L, abort, restart, head, tail, first, last, tag, rn)$$
$$\mid time1.\overline{timeout1}.Timer1)$$

Ici, une communication est possible le long du canal $time1$. Elle résulte en une action τ.

$$\dot\sim\ \tau^3.\overline{K}\ \ first\ \ last\ \ tag\ \ head.\tau.(\nu\ \ time1\ \ timeout1)$$
$$(add(rn, 1, rn).Attente\_ack(K, L, abort, restart, head, tail, first, last, tag, rn)$$
$$\mid \overline{timeout1}.Timer1)$$

$$\dot\sim\ \tau^3.\overline{K}\ \ first\ \ last\ \ tag\ \ head.\tau.add(rn, 1, rn).(\nu\ \ time1\ \ timeout1)$$
$$(Attente\_ack(K, L, abort, restart, head, tail, first, last, tag, rn)$$
$$\mid \overline{timeout1}.Timer1)$$

$$\dot\sim\ \tau^3.\overline{K}\ \ first\ \ last\ \ tag\ \ head.\tau.add(rn, 1, rn).(\nu\ \ time1\ \ timeout1)$$
$$(L.timeout1.[\![0]\!]rn.Not(tag).[\![False]\!]first.$$
$$Transfert(K, L, abort, restart, tail, first, last, tag, rn)$$
$$+\ timeout1.Retransmission(K, L, abort, restart, cons(head, tail), first, last, tag, rn)$$
$$\mid \overline{timeout1}.Timer1)$$

Deux situations sont possibles:

1. Le récepteur reçoit un acquittement ($P_{21}(K, L, abort, restart, tail, first, last, tag, rn)$).
2. Le récepteur reçoit le signal $timeout1$, une communication est donc possible le long du canal $timeout1$ ($P_{22}(K, L, abort, restart, cons(head, tail), first, last, tag, rn)$).

$$\begin{aligned}
\dot\sim\ & \tau^3.\overline{K}\ first\ last\ tag\ head.\tau.add(rn, 1, rn). \\
& (L.(\nu\ time1\ timeout1) \\
& \quad (timeout1.[\![0]\!]rn.Not(tag).[\![False]\!]first. \\
& \qquad Transfert(K, L, abort, restart, tail, first, last, tag, rn) \\
& \quad \mid \overline{timeout1}.Timer1) \\
& + \tau.(\nu\ time1\ timeout1) \\
& \quad (Retransmission(K, L, abort, restart, cons(head, tail), first, last, tag, rn) \\
& \quad \mid Timer1))
\end{aligned}$$

$$\begin{aligned}
& P_2(K, L, abort, restart, cons(head, tail), first, last, tag, rn)\ \dot\sim\ \tau^3.\overline{K}\ first\ last\ tag\ head. \\
& \quad \tau.add(rn, 1, rn).(P_{21}(K, L, abort, restart, tail, first, last, tag, rn) \\
& \qquad + P_{22}(K, L, abort, restart, cons(head, tail), first, last, tag, rn))
\end{aligned}$$

Considérons la partie $P_{21}(K, L, abort, restart, tail, first, last, tag, rn)$:

$$\begin{aligned}
& P_{21}(K, L, abort, restart, tail, first, last, tag, rn) \stackrel{def}{=} L.(\nu\ time1\ timeout1) \\
& \quad (timeout1.[\![0]\!]rn.Not(tag).[\![False]\!]first. \\
& \qquad Transfert(K, L, abort, restart, tail, first, last, tag, rn) \\
& \quad \mid \overline{timeout1}.Timer1)
\end{aligned}$$

Une communication est possible le long du canal $timeout1$. Elle résulte en une action τ.

$$\begin{aligned}
& P_{21}(K, L, abort, restart, tail, first, last, tag, rn)\ \dot\sim\ L.\tau.(\nu\ time1\ timeout1) \\
& \quad ([\![0]\!]rn.Not(tag).[\![False]\!]first.Transfert(K, L, abort, restart, tail, first, last, tag, rn) \\
& \quad \mid Timer1)
\end{aligned}$$

$$\begin{aligned}
\dot\sim\ & L.\tau.[\![0]\!]rn.(\nu\ time1\ timeout1) \\
& \quad (Not(tag).[\![False]\!]first.Transfert(K, L, abort, restart, tail, first, last, tag, rn) \\
& \quad \mid Timer1)
\end{aligned}$$

$$\begin{aligned}
\dot\sim\ & L.\tau.[\![0]\!]rn.Not(tag).(\nu\ time1\ timeout1) \\
& \quad ([\![False]\!]first.Transfert(K, L, abort, restart, tail, first, last, tag, rn) \\
& \quad \mid Timer1)
\end{aligned}$$

$$\begin{aligned}
\dot\sim\ & L.\tau.[\![0]\!]rn.Not(tag).[\![False]\!]first.(\nu\ time1\ timeout1) \\
& \quad (Transfert(K, L, abort, restart, tail, first, last, tag, rn) \\
& \quad \mid time1.\overline{timeout1}.Timer1)
\end{aligned}$$

$$\begin{aligned}
& P_{21}(K, L, abort, restart, tail, first, last, tag, rn)\ \dot\sim\ L.\tau.[\![0]\!]rn.Not(tag).[\![False]\!]first. \\
& \quad PP(K, L, abort, restart, tail, first, last, tag, rn)
\end{aligned}$$

Considérons la partie $P_{22}(K, L, abort, restart, cons(head, tail), first, last, tag, rn)$:

$$
\begin{aligned}
&P_{22}(K, L, abort, restart, cons(head, tail), first, last, tag, rn) \stackrel{def}{=} \tau.(\nu\ time1\ \ timeout1) \\
&\qquad (Retransmission(K, L, abort, restart, cons(head, tail), first, last, tag, rn) \\
&\qquad \mid Timer1)
\end{aligned}
$$

$$
\begin{aligned}
&P_{22}(K, L, abort, restart, cons(head, tail), first, last, tag, rn) \stackrel{.}{\sim} \tau.(\nu\ time1\ \ timeout1) \\
&\qquad (If\ \ equal(rn, MAX)\ \ then \\
&\qquad\quad ([last = True]\ \ \overline{Conf}\ \ DONTKNOW.\overline{abort}.Sender(K, L, abort, restart) \\
&\qquad\quad +\ [last = False]\ \ \overline{Conf}\ \ NOTOK.\overline{abort}.Sender(K, L, abort, restart)) \\
&\qquad else\ \ Transfert(K, L, abort, restart, cons(head, tail), first, last, tag, rn) \\
&\qquad \mid Timer1)
\end{aligned}
$$

Deux situations sont possibles:

1. Soit rn a atteint le MAX (i.e, le nombre restant de tentatives de retransmission, appelons le $reste$, est nul), selon la valeur de $last$, l'émetteur envoie une confirmation $NOTOK$ ou $DONTKNOW$ ($P_{221}(K, L, abort, restart, last)$).
2. Soit rn est inférieur à MAX (i.e, $reste > 0$), l'émetteur retransmet le message ($P_{222}(K, L, abort, restart, cons(head, tail), first, last, tag, rn)$).

$$
\begin{aligned}
&\stackrel{.}{\sim}\ \tau^2.((\nu\ time1\ \ timeout1) \\
&\qquad\quad (([last = True]\ \ \overline{Conf}\ \ DONTKNOW.\overline{abort}.Sender(K, L, abort, restart) \\
&\qquad\quad +\ [last = False]\ \ \overline{Conf}\ \ NOTOK.\overline{abort}.Sender(K, L, abort, restart)) \\
&\qquad\quad \mid Timer1) \\
&\quad +\ (\nu\ time1\ \ timeout1) \\
&\qquad\quad (Transfert(K, L, abort, restart, cons(head, tail), first, last, tag, rn) \\
&\qquad\quad \mid Timer1))
\end{aligned}
$$

La transition τ correspond à l'évaluation de $equal(rn, MAX)$.

$$
\begin{aligned}
&P_{22}(K, L, abort, restart, cons(head, tail), first, last, tag, rn) \stackrel{.}{\sim} \tau^2. \\
&\qquad (P_{221}(K, L, abort, restart, last) \\
&\qquad +\ P_{222}(K, L, abort, restart, cons(head, tail), first, last, tag, rn))
\end{aligned}
$$

Considérons la partie $P_{221}(K, L, abort, restart, last)$ ($reste = 0$, i.e, $rn = MAX$):

$$
\begin{aligned}
&P_{221}(K, L, abort, restart, last) \stackrel{def}{=} (\nu\ time1\ \ timeout1) \\
&\qquad (([last = True]\ \ \overline{Conf}\ \ DONTKNOW.\overline{abort}.Sender(K, L, abort, restart) \\
&\qquad +\ [last = False]\ \ \overline{Conf}\ \ NOTOK.\overline{abort}.Sender(K, L, abort, restart)) \\
&\qquad \mid Timer1)
\end{aligned}
$$

À nouveau, deux situations sont possibles:

1. la variable $last$ vaut la valeur $True$ ($P_{2211}(K, L, abort, restart, last)$).

2. la variable $last$ vaut la valeur $False$ ($P_{2212}(K, L, abort, restart, last)$).

$$P_{221}(K, L, abort, restart, last) \ \dot{\sim}\ (\tau.(\nu\ time1\ timeout1) \\ (\overline{Conf}\ \ DONTKNOW.\overline{abort}.Sender(K, L, abort, restart) \\ \mid Timer1) \\ +\ \tau.(\nu\ time1\ timeout1) \\ (\overline{Conf}\ \ NOTOK.\overline{abort}.Sender(K, L, abort, restart) \\ \mid Timer1))$$

$$P_{221}(K, L, abort, restart, last) \ \dot{\sim}\ P_{2211}(K, L, abort, restart, last) \\ +\ P_{2212}(K, L, abort, restart, last)$$

Considérons la partie $P_{2211}(K, L, abort, restart, last)$ ($rn = MAX$ et $last = True$):

$$P_{2211}(K, L, abort, restart, last) \stackrel{def}{=} \tau.(\nu\ time1\ timeout1) \\ (\overline{Conf}\ \ DONTKNOW.\overline{abort}.Sender(K, L, abort, restart) \\ \mid Timer1)$$

$$\dot{\sim}\ \tau.\overline{Conf}\ \ DONTKNOW.(\nu\ time1\ timeout1) \\ (\overline{abort}.Sender(K, L, abort, restart) \\ \mid Timer1)$$

$$\dot{\sim}\ \tau.\overline{Conf}\ \ DONTKNOW.\overline{abort}.(\nu\ time1\ timeout1) \\ (Sender(K, L, abort, restart) \\ \mid Timer1)$$

$$P_{2211}(K, L, abort, restart, last) \ \dot{\sim}\ \tau.\overline{Conf}\ \ DONTKNOW.\overline{abort}.P(K, L, abort, restart)$$

Considérons la partie $P_{2212}(K, L, abort, restart, last)$ ($rn = MAX$ et $last = False$):

$$P_{2212}(K, L, abort, restart, last) \stackrel{def}{=} \tau.(\nu\ time1\ timeout1) \\ (\overline{Conf}\ \ NOTOK.\overline{abort}.Sender(K, L, abort, restart) \\ \mid Timer1)$$

$$\dot{\sim}\ \tau.\overline{Conf}NOTOK.(\nu\ time1\ timeout1) \\ (\overline{abort}.Sender(K, L, abort, restart) \\ \mid Timer1)$$

$$\dot{\sim}\ \tau.\overline{Conf}NOTOK.\overline{abort}.(\nu\ time1\ timeout1) \\ (Sender(K, L, abort, restart) \\ \mid Timer1)$$

$$P_{2212}(K, L, abort, restart, last) \ \dot{\sim}\ \tau.\overline{Conf}\ \ NOTOK.\overline{abort}.P(K, L, abort, restart)$$

Considérons la partie $P_{222}(K, L, abort, restart, cons(head, tail), first, last, tag, rn)$ (reste > 0, on retransmet le message):

$$P_{222}(K, L, abort, restart, cons(head, tail), first, last, tag, rn) \stackrel{def}{=} (\nu\ time1\ timeout1)$$
$$(Transfert(K, L, abort, restart, cons(head, tail), first, last, tag, rn)$$
$$|\ Timer1)$$

$$P_{222}(K, L, abort, restart, cons(head, tail), first, last, tag, rn) \dot{\sim} (\nu\ time1\ timeout1)$$
$$(([cons(head, tail) = Nil]\ \overline{Conf}\ OK.\overline{restart}.Sender(K, L, abort, restart)$$
$$+\ [cons(head, tail) = cons(head, tail)]\ dernier(tail, last).$$
$$\overline{K}\ first\ last\ tag\ head.\overline{time1}.add(rn, 1, rn).$$
$$Attente\_ack(K, L, abort, restart, head, tail, first, last, tag, rn))$$
$$|\ Timer1)$$

Le fichier est naturellement non vide puisqu'il s'agit ici d'une retransmission de message.

$$\dot{\sim}\ \tau.(\nu\ time1\ timeout1)$$
$$(dernier(tail, last).\overline{K}\ first\ last\ tag\ head.\overline{time1}.$$
$$add(rn, 1, rn).Attente\_ack(K, L, abort, restart, head, tail, first, last, tag, rn)$$
$$|\ time1.\overline{timeout1}.Timer1)$$

$$P_{222}(K, L, abort, restart, cons(head, tail), first, last, tag, rn) \dot{\sim}$$
$$P_2(K, L, abort, restart, cons(head, tail), first, last, tag, rn)$$

Si on récapitule, on obtient les 10 équations suivantes:

$$P(K, L, abort, restart) \dot{\sim} \tau^4.Req(f).PP(K, L, abort, restart, f, first, last, tag, rn)$$

$$PP(K, L, abort, restart, f, first, last, tag, rn) \dot{\sim} (P_1(K, L, abort, restart)$$
$$+\ P_2(K, L, abort, restart, cons(head, tail), first, last, tag, rn))$$

$$P_1(K, L, abort, restart) \dot{\sim} \tau.\overline{Conf}\ OK.\overline{restart}.P(K, L, abort, restart)$$

$$P_2(K, L, abort, restart, cons(head, tail), first, last, tag, rn) \dot{\sim} \tau^3.\overline{K}\ first\ last\ tag\ head.$$
$$\tau.add(rn, 1, rn).(P_{21}(K, L, abort, restart, tail, first, last, tag, rn)$$
$$+\ P_{22}(K, L, abort, restart, cons(head, tail), first, last, tag, rn))$$

$$P_{21}(K, L, abort, restart, tail, first, last, tag, rn) \dot{\sim} L.\tau.[\![0]\!]rn.Not(tag).[\![False]\!]first.$$
$$PP(K, L, abort, restart, tail, first, last, tag, rn)$$

$$P_{22}(K, L, abort, restart, cons(head, tail), first, last, tag, rn) \dot{\sim} \tau^2.$$
$$(P_{221}(K, L, abort, restart, last)$$
$$+\ P_{222}(K, L, abort, restart, cons(head, tail), first, last, tag, rn))$$

$$P_{221}(K, L, abort, restart, last) \dot{\sim} P_{2211}(K, L, abort, restart, last)$$
$$+\ P_{2212}(K, L, abort, restart, last)$$

$$P_{2211}(K, L, abort, restart, last) \dot{\sim} \tau.\overline{Conf}\ DONTKNOW.\overline{abort}.P(K, L, abort, restart)$$

$$P_{2212}(K, L, abort, restart, last) \dot{\sim} \tau.\overline{Conf}\ NOTOK.\overline{abort}.P(K, L, abort, restart)$$

$$P_{222}(K, L, abort, restart, cons(head, tail), first, last, tag, rn) \dot{\sim}$$
$$P_2(K, L, abort, restart, cons(head, tail), first, last, tag, rn)$$

7.2.2 Expansion de la composante Q

De la même manière, appliquons la loi d'expansion à la composante $Q(K, L, abort, restart)$ afin d'établir toutes les étapes qu'elle peut exécuter.

$$Q(K, L, abort, restart) \stackrel{def}{=} (\nu\ \ time2\ \ timeout2)(Receiver(K, L, abort, restart) \mid Timer2)$$

$$\begin{aligned}
\equiv\ & (\nu\ \ time2\ \ timeout2)\\
& \quad ([\![False]\!]rtag.[\![False]\!]end.[\![False]\!]t2enabled.Reception(K, L, abort, restart, rtag, end, t2enabled)\\
& \quad \mid Timer2)
\end{aligned}$$

$$\begin{aligned}
\dot{\sim}\ & [\![False]\!]\ rtag.(\nu\ \ time2\ \ timeout2)\\
& \qquad ([\![False]\!]end.[\![False]\!]t2enabled.Reception(K, L, abort, restart, rtag, end, t2enabled)\\
& \qquad \mid Timer2)
\end{aligned}$$

$$\begin{aligned}
\dot{\sim}\ & [\![False]\!]\ rtag.[\![False]\!]end.(\nu\ \ time2\ \ timeout2)\\
& \qquad ([\![False]\!]t2enabled.Reception(K, L, abort, restart, rtag, end, t2enabled)\\
& \qquad \mid Timer2)
\end{aligned}$$

$$\begin{aligned}
\dot{\sim}\ & [\![False]\!]\ rtag.[\![False]\!]end.[\![False]\!]t2enabled.(\nu\ \ time2\ \ timeout2)\\
& \qquad (Reception(K, L, abort, restart, rtag, end, t2enabled)\\
& \qquad \mid Timer2)
\end{aligned}$$

$$\begin{aligned}
\dot{\sim}\ & \tau^3.(\nu\ \ time2\ \ timeout2)\\
& \qquad (Reception(K, L, abort, restart, rtag, end, t2enabled)\\
& \qquad \mid Timer2)
\end{aligned}$$

$$\begin{aligned}
\dot{\sim}\ & \tau^3.(\nu\ \ time2\ \ timeout2)\\
& \quad K(first\ \ last\ \ tag\ \ m).Traitement(K, L, abort, restart, first, last, tag, m, rtag, end, t2enabled)\\
& \quad +\ If\ \ equal_bool(t2enabled, True)\ \ then\\
& \qquad timeout2.abort.Abort(K, L, abort, restart, end)\\
& \quad +\ restart.\\
& \qquad If\ \ equal_bool(t2enabled, True)\ \ then\\
& \qquad\quad timeout2.Receiver(K, L, abort, restart)\\
& \qquad else\ \ Receiver(K, L, abort, restart)\\
& \quad \mid Timer2)
\end{aligned}$$

Deux situations sont possibles:

1. Soit le récepteur reçoit un message ($Q_1(K, L, abort, restart, rtag, end, t2enabled)$).

2. Soit il reçoit le signal $restart$ ($Q_2(K, L, abort, restart, t2enabled)$).

Le récepteur ne peut recevoir le signal $timeout2$ au démarrage car la variable $t2enabled$ vaut $False$ (i.e, $timer2$ n'a pas encore été déclenché).

$$\dot{\sim}\ \tau^3.(K(first\ \ last\ \ tag\ \ m).(\nu\ \ time2\ \ timeout2)$$

$$
\begin{aligned}
&\quad\quad (Traitement(K,L,abort,restart,first,last,tag,m,rtag,end,t2enabled)\\
&\quad\quad \mid Timer2)\\
&\quad + \; restart.(\nu \; time2 \; timeout2)\\
&\quad\quad (If \;\; equal\_bool(t2enabled,True) then\\
&\quad\quad\quad timeout2.Receiver(K,L,abort,restart)\\
&\quad\quad else \;\; Receiver(K,L,abort,restart)\\
&\quad\quad \mid Timer2))
\end{aligned}
$$

$$
\begin{aligned}
Q(K,L,abort,restart) \;\dot{\sim}\; & \tau^3.(Q_1(K,L,abort,restart,rtag,end,t2enabled)\\
& \quad + \; Q_2(K,L,abort,restart,t2enabled))
\end{aligned}
$$

Considérons la partie $Q_2(K,L,abort,restart,t2enabled)$ (i.e, réception du signal $restart$):

$$
\begin{aligned}
& Q_2(K,L,abort,restart,t2enabled) \stackrel{def}{=} restart.(\nu \; time2 \; timeout2)\\
&\quad\quad (If \;\; equal\_bool(t2enabled,True) then\\
&\quad\quad\quad timeout2.Receiver(K,L,abort,restart)\\
&\quad\quad else \;\; Receiver(K,L,abort,restart)\\
&\quad\quad \mid Timer2))
\end{aligned}
$$

Au démarrage $t2enabled$ vaut $False$.

$$
\begin{aligned}
& Q_2(K,L,abort,restart,t2enabled) \;\dot{\sim}\; restart.\tau.(\nu \; time2 \; timeout2)\\
&\quad\quad (Receiver(K,L,abort,restart)\\
&\quad\quad \mid Timer2)
\end{aligned}
$$

$$
Q_2(K,L,abort,restart,t2enabled) \;\dot{\sim}\; restart.\tau.Q(K,L,abort,restart)
$$

Considérons maintenant la partie $Q_1(K,L,abort,restart,rtag,end,t2enabled)$ (i.e, réception d'un message):

$$
\begin{aligned}
& Q_1(K,L,abort,restart,rtag,end,t2enabled) \stackrel{def}{=} K(first \;\; last \;\; tag \;\; m).(\nu \; time2 \; timeout2)\\
&\quad\quad (Traitement(K,L,abort,restart,first,last,tag,m,rtag,end,t2enabled)\\
&\quad\quad \mid Timer2)
\end{aligned}
$$

$$
\begin{aligned}
& Q_1(K,L,abort,restart,rtag,end,t2enabled) \;\dot{\sim}\; K(first \;\; last \;\; tag \;\; m).(\nu \; time2 \; timeout2)\\
&\quad (If \;\; equal\_bool(tag,rtag) \;\; then\\
&\quad\quad \overline{L}.Reception(K,L,abort,restart,rtag,end,t2enabled)\\
&\quad else\\
&\quad\quad If \;\; equal\_bool(first,True) \;\; then\\
&\quad\quad\quad copy\_bool(tag,rtag).Indication(K,L,abort,restart,first,last,m,rtag,end,t2enabled)\\
&\quad\quad else \;\; copy\_bool(tag,rtag).timeout2.\\
&\quad\quad\quad Indication(K,L,abort,restart,first,last,m,rtag,end,t2enabled)\\
&\quad \mid Timer2)
\end{aligned}
$$

Comme il s'agit du premier message du fichier, grâçe à l'initialisation, $rtag$ est différent de tag et $first$ vaut la valeur $True$.

$$\begin{array}{l}
\dot{\sim}\ K(first\ \ last\ \ tag\ \ m).\tau^2.(\nu\ \ time2\ \ timeout2)\\
\qquad\qquad (copy_bool(tag,rtag).Indication(K,L,abort,restart,first,last,m,rtag,end,t2enabled)\\
\qquad\qquad \mid Timer2)
\end{array}$$

$$\begin{array}{l}
\dot{\sim}\ K(first\ \ last\ \ tag\ \ m).\tau^2.copy_bool(tag,rtag).(\nu\ \ time2\ \ timeout2)\\
\qquad\qquad (Indication(K,L,abort,restart,first,last,m,rtag,end,t2enabled)\\
\qquad\qquad \mid Timer2)
\end{array}$$

$$\begin{array}{l}
\dot{\sim}\ K(first\ \ last\ \ tag\ \ m).\tau^2.copy_bool(tag,rtag).(\nu\ \ time2\ \ timeout2)\\
\quad (If\ \ equal_bool(last,True)\ \ then\\
\quad\quad [\![True]\!]end.\overline{Ind}\ \ m\ \ LAST.\overline{L}.\overline{time2}.[\![True]\!]t2enabled.\\
\quad\quad\quad Reception(K,L,abort,restart,rtag,end,t2enabled)\\
\quad else\\
\quad\quad If\ \ equal_bool(first,True)\ \ then\\
\quad\quad\quad \overline{Ind}\ \ m\ \ FIRST.\overline{L}.\overline{time2}.[\![True]\!]t2enabled.\\
\quad\quad\quad\quad Reception(K,L,abort,restart,rtag,end,t2enabled)\\
\quad\quad else\ \ \overline{Ind}\ \ m\ \ INCOMPLETE.\overline{L}.\overline{time2}.[\![True]\!]t2enabled.\\
\quad\quad\quad Reception(K,L,abort,restart,rtag,end,t2enabled)\\
\quad \mid Timer2)
\end{array}$$

Il s'agit du premier message du fichier. Cependant, deux cas sont possibles:

1. ce message peut être aussi le dernier message du fichier

   $(Q_{11}(K,L,abort,restart,m,rtag,end,t2enabled))$.

2. ce message n'est pas le dernier message du fichier

   $(Q_{12}(K,L,abort,restart,m,rtag,end,t2enabled))$.

$$\begin{array}{l}
\dot{\sim}\ K(first\ \ last\ \ tag\ \ m).\tau^2.copy_bool(tag,rtag).\tau.((\nu\ \ time2\ \ timeout2)\\
\quad\quad ([\![True]\!]end.\overline{Ind}\ \ m\ \ LAST.\overline{L}.\overline{time2}.[\![True]\!]t2enabled.\\
\quad\quad\quad Reception(K,L,abort,restart,rtag,end,t2enabled)\\
\quad\quad \mid Timer2)\\
\quad +\ \ \tau.(\nu\ \ time2\ \ timeout2)\\
\quad\quad (\overline{Ind}\ \ m\ \ FIRST.\overline{L}.\overline{time2}.[\![True]\!]t2enabled.\\
\quad\quad\quad Reception(K,L,abort,restart,rtag,end,t2enabled)\\
\quad\quad \mid Timer2))
\end{array}$$

$$\begin{array}{l}
Q_1(K,L,abort,restart,rtag,end,t2enabled)\ \dot{\sim}\ K(first\ \ last\ \ tag\ \ m).\tau^2.copy_bool(tag,rtag).\tau.\\
\qquad\qquad\qquad (Q_{11}(K,L,abort,restart,m,rtag,end,t2enabled)\\
\qquad\qquad\qquad +\ \ Q_{12}(K,L,abort,restart,m,rtag,end,t2enabled))
\end{array}$$

Considérons la partie $Q_{11}(K, L, abort, restart, m, rtag, end, t2enabled)$:

$$
\begin{aligned}
&Q_{11}(K, L, abort, restart, m, rtag, end, t2enabled) \stackrel{def}{=} (\nu\ time2\ timeout2)\\
&\quad ([\![True]\!]end.\overline{Ind}\ m\ LAST.\overline{L}.\overline{time2}.[\![True]\!]t2enabled.\\
&\qquad Reception(K, L, abort, restart, rtag, end, t2enabled)\\
&\quad \mid Timer2)
\end{aligned}
$$

$$
\begin{aligned}
&Q_{11}(K, L, abort, restart, m, rtag, end, t2enabled) \stackrel{\cdot}{\sim} [\![True]\!]end.(\nu\ time2\ timeout2)\\
&\quad (\overline{Ind}\ m\ LAST.\overline{L}.\overline{time2}.[\![True]\!]t2enabled.\\
&\qquad Reception(K, L, abort, restart, rtag, end, t2enabled)\\
&\quad \mid time2.\overline{timeout2}.Timer2)
\end{aligned}
$$

$$
\begin{aligned}
&\stackrel{\cdot}{\sim} \tau.\overline{Ind}\ m\ LAST.(\nu\ time2\ timeout2)\\
&\quad (\overline{L}.\overline{time2}.[\![True]\!]t2enabled.Reception(K, L, abort, restart, rtag, end, t2enabled)\\
&\quad \mid time2.\overline{timeout2}.Timer2)
\end{aligned}
$$

$$
\begin{aligned}
&\stackrel{\cdot}{\sim} \tau.\overline{Ind}\ m\ LAST.\overline{L}.(\nu\ time2\ timeout2)\\
&\quad (\overline{time2}.[\![True]\!]t2enabled.Reception(K, L, abort, restart, rtag, end, t2enabled)\\
&\quad \mid time2.\overline{timeout2}.Timer2)
\end{aligned}
$$

$$
\begin{aligned}
&\stackrel{\cdot}{\sim} \tau.\overline{Ind}\ m\ LAST.\overline{L}.\tau.(\nu\ time2\ timeout2)\\
&\quad ([\![True]\!]t2enabled.Reception(K, L, abort, restart, rtag, end, t2enabled)\\
&\quad \mid \overline{timeout2}.Timer2)
\end{aligned}
$$

$$
\begin{aligned}
&\stackrel{\cdot}{\sim} \tau.\overline{Ind}\ m\ LAST.\overline{L}.\tau.[\![True]\!]t2enabled.(\nu\ time2\ timeout2)\\
&\quad (Reception(K, L, abort, restart, rtag, end, t2enabled)\\
&\quad \mid \overline{timeout2}.Timer2)
\end{aligned}
$$

$$
\begin{aligned}
&Q_{11}(K, L, abort, restart, m, rtag, end, t2enabled) \stackrel{\cdot}{\sim} \tau.\overline{Ind}\ m\ LAST.\overline{L}.\tau.\\
&\quad [\![True]\!]t2enabled.QQ(K, L, abort, restart, rtag, end, t2enabled)
\end{aligned}
$$

Considérons la partie $Q_{12}(K, L, abort, restart, m, rtag, end, t2enabled)$:

$$
\begin{aligned}
&Q_{12}(K, L, abort, restart, m, rtag, end, t2enabled) \stackrel{def}{=} \tau.(\nu\ time2\ timeout2)\\
&\quad (\overline{Ind}\ m\ FIRST.\overline{L}.\overline{time2}.[\![True]\!]t2enabled.\\
&\qquad Reception(K, L, abort, restart, rtag, end, t2enabled)\\
&\quad \mid Timer2)
\end{aligned}
$$

$$
\begin{aligned}
&Q_{12}(K, L, abort, restart, m, rtag, end, t2enabled) \stackrel{\cdot}{\sim} \tau.\overline{Ind}\ m\ FIRST.(\nu\ time2\ timeout2)\\
&\quad (\overline{L}.\overline{time2}.[\![True]\!]t2enabled.Reception(K, L, abort, restart, rtag, end, t2enabled)\\
&\quad \mid time2.\overline{timeout2}.Timer2)
\end{aligned}
$$

$$
\begin{aligned}
&\stackrel{\cdot}{\sim} \tau.\overline{Ind}\ m\ FIRST.\overline{L}.(\nu\ time2\ timeout2)\\
&\quad (\overline{time2}.[\![True]\!]t2enabled.Reception(K, L, abort, restart, rtag, end, t2enabled)\\
&\quad \mid time2.\overline{timeout2}.Timer2)
\end{aligned}
$$

$\dot{\sim}\ \tau.\overline{Ind}\ \ m\ \ FIRST.\overline{L}.\tau.(\nu\ \ time2\ \ timeout2)$
$\quad([\![True]\!]t2enabled.Reception(K, L, abort, restart, rtag, end, t2enabled)$
$\quad\mid \overline{timeout2}.Timer2)$

$\dot{\sim}\ \tau.\overline{Ind}\ \ m\ \ FIRST.\overline{L}.\tau.[\![True]\!]t2enabled.(\nu\ \ time2\ \ timeout2)$
$\quad(Reception(K, L, abort, restart, rtag, end, t2enabled)$
$\quad\mid \overline{timeout2}.Timer2)$

$Q_{12}(K, L, abort, restart, m, rtag, end, t2enabled)\ \dot{\sim}\ \tau.\overline{Ind}\ \ m\ \ FIRST.\overline{L}.\tau.$
$\quad[\![True]\!]t2enabled.QQ(K, L, abort, restart, rtag, end, t2enabled)$

Considérons la partie $QQ(K, L, abort, restart, rtag, end, t2enabled)$ (au moins un message a été reçu):

$QQ(K, L, abort, restart, rtag, end, t2enabled) \stackrel{def}{=} (\nu\ \ time2\ \ timeout2)$
$\quad(Reception(K, L, abort, restart, rtag, end, t2enabled)$
$\quad\mid \overline{timeout2}.Timer2)$

$QQ(K, L, abort, restart, rtag, end, t2enabled)\ \dot{\sim}\ (\nu\ \ time2\ \ timeout2)$
$\quad(K(first\ \ last\ \ tag\ \ m).Traitement(K, L, abort, restart, first, last, tag, m, rtag, end, t2enabled)$
$\quad+\ If\ \ equal\_bool(t2enabled, True) then$
$\qquad timeout2.abort.Abort(K, L, abort, restart, end)$
$\quad+\ restart.$
$\qquad If\ \ equal\_bool(t2enabled, True) then$
$\qquad\quad timeout2.Receiver(K, L, abort, restart)$
$\qquad else\ \ Receiver(K, L, abort, restart)$
$\quad\mid \overline{timeout2}.Timer2)$

À ce niveau, $t2enabled$ vaut la valeur $True$ (i.e, $timer2$ est déclenché, voir Q_{11} et Q_{12}). Trois situations sont possibles:

1. Soit le récepteur reçoit un message ($QQ_1(K, L, abort, restart, rtag, end, t2enabled)$).
2. Soit il reçoit le signal $timeout2$ ($QQ_2(K, L, abort, restart, end)$).
3. Soit il reçoit le signal $restart$ ($QQ_3(K, L, abort, restart, t2enabled)$).

$\dot{\sim}\ K(first\ \ last\ \ tag\ \ m).(\nu\ \ time2\ \ timeout2)$
$\quad(Traitement(K, L, abort, restart, first, last, tag, m, rtag, end, t2enabled)$
$\quad\mid \overline{timeout2}.Timer2)$
$\quad+\ \tau.(\nu\ \ time2\ \ timeout2)$
$\quad(timeout2.abort.Abort(K, L, abort, restart, end) \mid \overline{timeout2}.Timer2)$
$\quad+\ restart.(\nu\ \ time2\ \ timeout2)$
$\quad(If\ \ equal\_bool(t2enabled, True) then$
$\qquad timeout2.Receiver(K, L, abort, restart)$
$\quad else\ \ Receiver(K, L, abort, restart)$
$\quad\mid \overline{timeout2}.Timer2)$

$$QQ(K,L,abort,restart,rtag,end,t2enabled) \stackrel{\cdot}{\sim} QQ_1(K,L,abort,restart,rtag,end,t2enabled) + \\ QQ_2(K,L,abort,restart,end) + QQ_3(K,L,abort,restart,t2enabled)$$

Considérons la partie $QQ_1(K,L,abort,restart,rtag,end,t2enabled)$ (réception d'un message):

$$QQ_1(K,L,abort,restart,rtag,end,t2enabled) \stackrel{def}{=} K(first\ \ last\ \ tag\ \ m).(\nu\ \ time2\ \ timeout2) \\ (Traitement(K,L,abort,restart,first,last,tag,m,rtag,end,t2enabled) \\ \mid \overline{timeout2}.Timer2)$$

$$QQ_1(K,L,abort,restart,rtag,end,t2enabled) \stackrel{\cdot}{\sim} K(first\ \ last\ \ tag\ \ m).(\nu\ \ time2\ \ timeout2) \\ (If\ \ equal\_bool(tag,rtag)\ \ then \\ \overline{L}.Reception(K,L,abort,restart,rtag,end,t2enabled) \\ else \\ If\ \ equal\_bool(first,True)\ \ then \\ copy\_bool(tag,rtag).Indication(K,L,abort,restart,first,last,m,rtag,end,t2enabled) \\ else\ \ copy\_bool(tag,rtag).timeout2. \\ Indication(K,L,abort,restart,first,last,m,rtag,end,t2enabled) \\ \mid \overline{timeout2}.Timer2)$$

Deux cas sont possibles:

1. Soit il s'agit d'une retransmission de message ($QQ_{11}(K,L,abort,restart,rtag,end,t2enabled)$).

2. Soit il s'agit d'un nouveau message et ce n'est pas le premier message du fichier ($QQ_{12}(K,L,abort,restart,first,last,tag,m,rtag,end,t2enabled)$).

S'il s'agit d'un nouveau message, il ne peut être le premier message du fichier car maintenant $first$ vaut la valeur $False$.

$$\stackrel{\cdot}{\sim} K(first\ \ last\ \ tag\ \ m).\tau. \\ ((\nu\ \ time2\ \ timeout2) \\ (\overline{L}.Reception(K,L,abort,restart,rtag,end,t2enabled) \\ \mid \overline{timeout2}.Timer2) \\ + \tau.(\nu\ \ time2\ \ timeout2) \\ (copy\_bool(tag,rtag).timeout2. \\ Indication(K,L,abort,restart,first,last,m,rtag,end,t2enabled) \\ \mid \overline{timeout2}.Timer2))$$

$$QQ_1(K,L,abort,restart,rtag,end,t2enabled) \stackrel{\cdot}{\sim} K(first\ \ last\ \ tag\ \ m).\tau. \\ (QQ_{11}(K,L,abort,restart,rtag,end,t2enabled) \\ + QQ_{12}(K,L,abort,restart,first,last,tag,m,rtag,end,t2enabled))$$

Considérons la partie $QQ_{11}(K,L,abort,restart,rtag,end,t2enabled)$ (réception d'un message dupliqué):

$$QQ_{11}(K,L,abort,restart,rtag,end,t2enabled) \stackrel{def}{=} (\nu\ \ time2\ \ timeout2) \\ (\overline{L}.Reception(K,L,abort,restart,rtag,end,t2enabled) \\ \mid \overline{timeout2}.Timer2)$$

$$QQ_{11}(K,L,abort,restart,rtag,end,t2enabled) \;\dot{\sim}\; \overline{L}.(\nu\ \ time2\ \ timeout2)$$
$$(Reception(K,L,abort,restart,rtag,end,t2enabled)$$
$$\mid \overline{timeout2}.Timer2)$$

$$QQ_{11}(K,L,abort,restart,rtag,end,t2enabled) \;\dot{\sim}\; \overline{L}.QQ(K,L,abort,restart,rtag,end,t2enabled)$$

Considérons la partie $QQ_{12}(K,L,abort,restart,first,last,tag,m,rtag,end,t2enabled)$ (réception d'un nouveau message):

$$QQ_{12}(K,L,abort,restart,first,last,tag,m,rtag,end,t2enabled) \stackrel{def}{=} \tau.(\nu\ \ time2\ \ timeout2)$$
$$(copy\_bool(tag,rtag).timeout2.Indication(K,L,abort,restart,first,last,m,rtag,end,t2enabled)$$
$$\mid \overline{timeout2}.Timer2)$$

$$QQ_{12}(K,L,abort,restart,first,last,tag,m,rtag,end,t2enabled) \;\dot{\sim}\; \tau.copy\_bool(tag,rtag).$$
$$(\nu\ \ time2\ \ timeout2)$$
$$(timeout2.Indication(K,L,abort,restart,first,last,m,rtag,end,t2enabled)$$
$$\mid \overline{timeout2}.Timer2)$$

$$\dot{\sim}\; \tau^2.(\nu\ \ time2\ \ timeout2)$$
$$(timeout2.Indication(K,L,abort,restart,first,last,m,rtag,end,t2enabled)$$
$$\mid \overline{timeout2}.Timer2)$$

$$\dot{\sim}\; \tau^3.(\nu\ \ time2\ \ timeout2)$$
$$(Indication(K,L,abort,restart,first,last,m,rtag,end,t2enabled)$$
$$\mid Timer2)$$

$$\dot{\sim}\; \tau^3.(\nu\ \ time2\ \ timeout2)$$
$$(If\ \ equal\_bool(last,True)\ \ then$$
$$[\![True]\!]end.\overline{Ind}\ \ m\ \ LAST.\overline{L}.\overline{time2}.[\![True]\!]t2enabled.$$
$$Reception(K,L,abort,restart,rtag,end,t2enabled)$$
$$else$$
$$If\ \ equal\_bool(first,True)\ \ then$$
$$\overline{Ind}\ \ m\ \ FIRST.\overline{L}.\overline{time2}.[\![True]\!]t2enabled.$$
$$Reception(K,L,abort,restart,rtag,end,t2enabled)$$
$$else\ \ \overline{Ind}\ \ m\ \ INCOMPLETE.\overline{L}.\overline{time2}.[\![True]\!]t2enabled.$$
$$Reception(K,L,abort,restart,rtag,end,t2enabled)$$
$$\mid Timer2)$$

Deux cas sont possibles:

1. Soit il s'agit du dernier message du fichier ($last$ vaut la valeur $True$), le récepteur envoie une indication $LAST$ ($Q_{11}(K,L,abort,restart,m,rtag,end,t2enabled)$).

2. Soit il s'agit d'un message intermédiaire, le récepteur envoie une indication $INCOMPLETE$ ($QQ_{122}(K,L,abort,restart,m,rtag,end,t2enabled)$).

Il ne peut s'agir du premier message du fichier ($first$ vaut $False$, voir précédement).

$$\dot{\sim}\ \tau^4.((\nu\ time2\ timeout2)$$
$$([\![True]\!]end.\overline{Ind}\ m\ LAST.\overline{L}.\overline{time2}.[\![True]\!]t2enabled.$$
$$Reception(K,L,abort,restart,rtag,end,t2enabled)$$
$$\mid Timer2)$$
$$+\ \tau.(\nu\ time2\ timeout2)$$
$$(\overline{Ind}\ m\ INCOMPLETE.\overline{L}.\overline{time2}.[\![True]\!]t2enabled.$$
$$Reception(K,L,abort,restart,rtag,end,t2enabled)$$
$$\mid Timer2))$$

$$QQ_{12}(K,L,abort,restart,first,last,tag,m,rtag,end,t2enabled)\ \dot{\sim}\ \tau^4.$$
$$(Q_{11}(K,L,abort,restart,m,rtag,end,t2enabled)$$
$$+\ \tau.QQ_{122}(K,L,abort,restart,m,rtag,end,t2enabled))$$

Considérons la partie $QQ_{122}(K,L,abort,restart,m,rtag,end,t2enabled)$ (livraison d'un message intermédiaire du fichier):

$$QQ_{122}(K,L,abort,restart,m,rtag,end,t2enabled)\ \stackrel{def}{=}\ (\nu\ time2\ timeout2)$$
$$(\overline{Ind}\ m\ INCOMPLETE.\overline{L}.\overline{time2}.[\![True]\!]t2enabled.$$
$$Reception(K,L,abort,restart,rtag,end,t2enabled)$$
$$\mid Timer2)$$

$$QQ_{122}(K,L,abort,restart,m,rtag,end,t2enabled)\ \dot{\sim}\ \overline{Ind}\ m\ INCOMPLETE.$$
$$(\nu\ time2\ timeout2)$$
$$(\overline{L}.\overline{time2}.[\![True]\!]t2enabled.Reception(K,L,abort,restart,rtag,end,t2enabled)$$
$$\mid time2.\overline{timeout2}.Timer2)$$

$$\dot{\sim}\ \overline{Ind}\ m\ INCOMPLETE.\overline{L}.(\nu\ time2\ timeout2)$$
$$(\overline{time2}.[\![True]\!]t2enabled.Reception(K,L,abort,restart,rtag,end,t2enabled)$$
$$\mid time2.\overline{timeout2}.Timer2)$$

$$\dot{\sim}\ \overline{Ind}\ m\ INCOMPLETE.\overline{L}.\tau.(\nu\ time2\ timeout2)$$
$$([\![True]\!]t2enabled.Reception(K,L,abort,restart,rtag,end,t2enabled)$$
$$\mid \overline{timeout2}.Timer2)$$

$$\dot{\sim}\ \overline{Ind}\ m\ INCOMPLETE.\overline{L}.\tau.[\![True]\!]t2enabled.(\nu\ time2\ timeout2)$$
$$(Reception(K,L,abort,restart,rtag,end,t2enabled)$$
$$\mid \overline{timeout2}.Timer2)$$

$$QQ_{122}(K,L,abort,restart,m,rtag,end,t2enabled)\ \dot{\sim}\ \overline{Ind}\ m\ INCOMPLETE.\overline{L}.\tau.$$
$$[\![True]\!]t2enabled.QQ(K,L,abort,restart,rtag,end,t2enabled)$$

Considérons la partie $QQ_2(K,L,abort,restart,end)$ (expiration de $timer2$ car aucun nouveau message n'est arrivé, le récepteur interrompt le transfert):

$$QQ_2(K,L,abort,restart,end)\ \stackrel{def}{=}\ \tau.(\nu\ time2\ timeout2)$$
$$(timeout2.abort.Abort(K,L,abort,restart,end)$$
$$\mid \overline{timeout2}.Timer2)$$

$$QQ_2(K, L, abort, restart, end) \stackrel{\cdot}{\sim} \tau^2.(\nu\ time2\ timeout2)$$
$$(abort.Abort(K, L, abort, restart, end)$$
$$\mid Timer2)$$

$$\stackrel{\cdot}{\sim} \tau^2.abort.(\nu\ time2\ timeout2)$$
$$(Abort(K, L, abort, restart, end)$$
$$\mid Timer2)$$

$$\stackrel{\cdot}{\sim} \tau^2.abort.(\nu\ time2\ timeout2)$$
$$(If\ \ equal\_bool(end, True)\ \ then$$
$$Receiver(K, L, abort, restart)$$
$$else$$
$$\overline{Ind\_err}.Receiver(K, L, abort, restart)$$
$$\mid Timer2)$$

Deux situations sont possibles:

1. L'abort se produit sur le dernier message du fichier, le récepteur se réinitialise et se remet en attente de message ($Q(K, L, abort, restart)$).
2. L'abort se produit sur un message intermédiaire du fichier, le récepteur envoie une indication d'erreur, ensuite se réinitialise et se remet en attente de message

   ($QQ_{22}(K, L, abort, restart)$).

$$\stackrel{\cdot}{\sim} \tau^2.abort.\tau.((\nu\ time2\ timeout2)$$
$$(Receiver(K, L, abort, restart) \mid Timer2)$$
$$+\ (\nu\ time2\ timeout2)$$
$$(\overline{Ind\_err}.Receiver(K, L, abort, restart) \mid Timer2))$$

$$QQ_2(K, L, abort, restart, end) \stackrel{\cdot}{\sim} \tau^2.abort.\tau.(Q(K, L, abort, restart)$$
$$+\ QQ_{22}(K, L, abort, restart))$$

Considérons la partie $QQ_{22}(K, L, abort, restart)$ (abort lors du transfert d'un message intermédiaire du fichier, le récepteur envoie une indication d'erreur au consommateur):

$$QQ_{22}(K, L, abort, restart) \stackrel{def}{=} (\nu\ time2\ timeout2)$$
$$(\overline{Ind\_err}.Receiver(K, L, abort, restart) \mid Timer2)$$

$$QQ_{22}(K, L, abort, restart) \stackrel{\cdot}{\sim} \overline{Ind\_err}.(\nu\ time2\ timeout2)$$
$$(Receiver(K, L, abort, restart) \mid Timer2)$$

$$QQ_{22}(K, L, abort, restart) \stackrel{\cdot}{\sim} \overline{Ind\_err}.Q(K, L, abort, restart)$$

Considérons la partie $QQ_3(K, L, abort, restart, t2enabled)$ (réception du signal $restart$ par le récepteur après la fin du transfert d'un fichier, il doit se réinitialiser):

$$QQ_3(K, L, abort, restart, t2enabled) \stackrel{def}{=} restart.(\nu\ time2\ timeout2)$$
$$(If\ \ equal\_bool(t2enabled, True) then$$
$$timeout2.Receiver(K, L, abort, restart)$$
$$else\ \ Receiver(K, L, abort, restart)$$
$$\mid \overline{timeout2}.Timer2)$$

Or, $t2enabled$ vaut la valeur $True$ (i.e, $timer2$ est déclenché après l'acquittement du dernier message du fichier).

$$QQ_3(K, L, abort, restart, t2enabled) \dot{\sim} restart.\tau.(\nu\ time2\ timeout2)$$
$$(timeout2.Receiver(K, L, abort, restart) \mid \overline{timeout2}.Timer2)$$

$$\dot{\sim} restart.\tau^2.(\nu\ time2\ timeout2)$$
$$(Receiver(K, L, abort, restart) \mid Timer2)$$

$$QQ_3(K, L, abort, restart, t2enabled) \dot{\sim} restart.\tau^2.Q(K, L, abort, restart)$$

Si on récapitule, on obtient les 13 équations suivantes:

$$Q(K,L, abort,restart) \dot{\sim} \tau^3.(Q_1(K, L, abort, restart, rtag, end, t2enabled)$$
$$+ Q_2(K, L, abort, restart, t2enabled))$$

$$Q_2(K, L, abort, restart, t2enabled) \dot{\sim} restart.\tau.Q(K, L, abort, restart)$$

$$Q_1(K, L, abort, restart, rtag, end, t2enabled) \dot{\sim} K(first\ \ last\ \ tag\ \ m).\tau^2.copy\_bool(tag, rtag).\tau.$$
$$(Q_{11}(K, L, abort, restart, m, rtag, end, t2enabled)$$
$$+ Q_{12}(K, L, abort, restart, m, rtag, end, t2enabled))$$

$$Q_{11}(K, L, abort, restart, m, rtag, end, t2enabled) \dot{\sim} \tau.\overline{Ind}\ \ m\ \ LAST.\overline{L}.\tau.$$
$$[\![True]\!]t2enabled.QQ(K, L, abort, restart, rtag, end, t2enabled)$$

$$Q_{12}(K, L, abort, restart, m, rtag, end, t2enabled) \dot{\sim} \tau.\overline{Ind}\ \ m\ \ FIRST.\overline{L}.\tau.$$
$$[\![True]\!]t2enabled.QQ(K, L, abort, restart, rtag, end, t2enabled)$$

$$QQ(K, L, abort, restart, rtag, end, t2enabled) \dot{\sim} QQ_1(K, L, abort, restart, rtag, end, t2enabled) +$$
$$QQ_2(K, L, abort, restart, end) + QQ_3(K, L, abort, restart, t2enabled)$$

$$QQ_1(K, L, abort, restart, rtag, end, t2enabled) \dot{\sim} K(first\ \ last\ \ tag\ \ m).\tau.$$
$$(QQ_{11}(K, L, abort, restart, rtag, end, t2enabled)$$
$$+ QQ_{12}(K, L, abort, restart, first, last, tag, m, rtag, end, t2enabled))$$

$$QQ_{11}(K, L, abort, restart, rtag, end, t2enabled) \dot{\sim} \overline{L}.QQ(K, L, abort, restart, rtag, end, t2enabled)$$

$$QQ_{12}(K, L, abort, restart, first, last, tag, m, rtag, end, t2enabled) \;\dot{\sim}\; \tau^4.$$
$$(Q_{11}(K, L, abort, restart, m, rtag, end, t2enabled)$$
$$+\; \tau.QQ_{122}(K, L, abort, restart, m, rtag, end, t2enabled))$$

$$QQ_{122}(K,L, abort, restart, m, rtag, end, t2enabled) \;\dot{\sim}\; \overline{Ind}\;\; m\;\; INCOMPLETE.\overline{L}.\tau.$$
$$[\![True]\!]t2enabled.QQ(K, L, abort, restart, rtag, end, t2enabled)$$

$$QQ_2(K, L, abort, restart, end) \;\dot{\sim}\; \tau^2.abort.\tau.(Q(K, L, abort, restart)$$
$$+\; QQ_{22}(K, L, abort, restart))$$

$$QQ_{22}(K, L, abort, restart) \;\dot{\sim}\; \overline{Ind\_err}.Q(K, L, abort, restart)$$

$$QQ_3(K, L, abort, restart, t2enabled) \;\dot{\sim}\; restart.\tau^2.Q(K, L, abort, restart)$$

7.2.3 Expansion du système

Maintenant, on peut appliquer la loi d'expansion au système (i.e, à la composition de P, Q, *perte_msg* et *perte_ack*) afin d'établir toutes les étapes qu'il peut exécuter.

$$System \equiv (\nu\;\; K\;\; L\;\; abort\;\; restart)(P(K, L, abort, restart) \mid Q(K, L, abort, restart))$$

Rappelons que les processus *perte_msg* et *perte_ack* s'exécutent en parallèle avec les composantes P et Q et qu'ils peuvent se produire à tout moment. Dans un souci d'allégement des expressions dans ce qui suit, ils n'ont pas été représentés dans la composition parallèle. Toutefois, ils sont utilisés quand c'est nécessaire.

$$\dot{\sim}\; (\nu\;\; K\;\; L\;\; abort\;\; Restart)$$
$$(\tau^4.Req(f).PP(K, L, abort, restart, f, first, last, tag, rn)$$
$$\mid \tau^3.(Q_1(K, L, abort, restart, rtag, end, t2enabled) \;+\; Q_2(K, L, abort, restart, t2enabled)))$$

$$\dot{\sim}\; (\nu\;\; K\;\; L\;\; abort\;\; restart)$$
$$(\tau^4.(Req(Nil).P_1(K, L, abort, restart)$$
$$+\; Req(cons(head, tail)).P_2(K, L, abort, restart, cons(head, tail), first, last, tag, rn))$$
$$\mid \tau^3.(Q_1(K, L, abort, restart, rtag, end, t2enabled) \;+\; Q_2(K, L, abort, restart, t2enabled)))$$

Les composantes P et Q peuvent exécuter indépendamment leurs actions internes.

$$\dot{\sim}\; \tau^7.(\nu\;\; K\;\; L\;\; abort\;\; restart)$$
$$((Req(Nil).P_1(K, L, abort, restart)$$
$$+\; Req(cons(head, tail)).P_2(K, L, abort, restart, cons(head, tail), first, last, tag, rn))$$
$$\mid (Q_1(K, L, abort, restart, rtag, end, t2enabled) \;+\; Q_2(K, L, abort, restart, t2enabled)))$$

$$\dot{\sim}\; \tau^7.(\nu\;\; K\;\; L\;\; abort\;\; restart)$$
$$((Req(Nil).\tau.\overline{Conf}\;\; OK.\overline{restart}.P(K, L, abort, restart)$$
$$+\; Req(cons(head, tail)).\tau^3.\overline{K}\;\; first\;\; last\;\; tag\;\; head.\tau.add(rn, 1, rn).$$
$$(P_{21}(K, L, abort, restart, tail, first, last, tag, rn)$$

$$
\begin{aligned}
&\quad + \; P_{22}(K, L, abort, restart, cons(head, tail), first, last, tag, rn)))\\
&\mid (K(first\ \ last\ \ tag\ \ m).\tau^2.copy\_bool(tag, rtag).\tau.\\
&\quad (Q_{11}(K, L, abort, restart, m, rtag, end, t2enabled)\\
&\quad + \; Q_{12}(K, L, abort, restart, m, rtag, end, t2enabled))\\
&+ \; restart.\tau.Q(K, L, abort, restart)))
\end{aligned}
$$

Trois situations sont possibles:

- L'émetteur reçoit un fichier vide, il émet le signal *restart* au récepteur pour que celui-ci se réinitialise (A_2).
- Le fichier reçu est non vide, l'émetteur émet un message au récepteur et ce message est bien reçu par le récepteur ($A_3(head, tail)$).
- Le fichier reçu est non vide, l'émetteur émet un message au récepteur mais ce message est perdu car l'évévement *perte_msg* (i.e, $K(first\ last\ tag\ m).perte\_msg$) se produit ($A_5(head, tail)$).

$$
\begin{aligned}
\dot{\sim}\ & \tau^7.(Req(Nil).\tau.\overline{Conf}\ \ OK.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\qquad (\overline{restart}.P(K, L, abort, restart) \mid restart.\tau.Q(K, L, abort, restart))\\
&\ + \; Req(cons(head, tail)).\tau^3.\tau.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\qquad (\tau.add(rn, 1, rn).(P_{21}(K, L, abort, restart, tail, first, last, tag, rn)\\
&\qquad\qquad + \; P_{22}(K, L, abort, restart, cons(head, tail), first, last, tag, rn))\\
&\qquad \mid \tau^2.copy\_bool(tag, rtag).\tau.\\
&\qquad\quad (Q_{11}(K, L, abort, restart, head, rtag, end, t2enabled)\\
&\qquad\quad + \; Q_{12}(K, L, abort, restart, head, rtag, end, t2enabled)))\\
&\ + \; Req(cons(head, tail)).\tau^3.\tau.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\qquad (\tau.add(rn, 1, rn).(P_{21}(K, L, abort, restart, tail, first, last, tag, rn)\\
&\qquad\qquad + \; P_{22}(K, L, abort, restart, cons(head, tail), first, last, tag, rn)))\\
&\qquad \mid (K(first\ \ last\ \ tag\ \ m).\tau^2.copy\_bool(tag, rtag).\tau.\\
&\qquad\quad (Q_{11}(K, L, abort, restart, m, rtag, end, t2enabled)\\
&\qquad\quad + \; Q_{12}(K, L, abort, restart, m, rtag, end, t2enabled))\\
&\qquad + \; restart.\tau.Q(K, L, abort, restart)))
\end{aligned}
$$

$$
\begin{aligned}
System\ \dot{\sim}\ & \tau^7.(Req(Nil).A_2 \; + \; Req(cons(head, tail)).A_3(head, tail)\\
& + \; Req(cons(head, tail)).\tau^4.A_5(head, tail))
\end{aligned}
$$

Considérons la partie A_2 (le fichier reçu par l'émetteur est vide):

$$
\begin{aligned}
A_2 \stackrel{def}{=}\ & \tau.\overline{Conf}\ \ OK.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
& (\overline{restart}.P(K, L, abort, restart) \mid restart.\tau.Q(K, L, abort, restart))
\end{aligned}
$$

$$
\begin{aligned}
A_2\ \dot{\sim}\ & \tau.\overline{Conf}\ \ OK.\tau.\tau.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
& (P(K, L, abort, restart) \mid Q(K, L, abort, restart))
\end{aligned}
$$

$$
A_2\ \dot{\sim}\ \tau.\overline{Conf}\ \ OK.\tau^2.System
$$

Considérons la partie $A_5(head, tail)$ (le premier message du fichier émis par l'émetteur est perdu):

$$
\begin{aligned}
&A_5(head{,}tail) \stackrel{def}{=} (\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\quad (\tau.add(rn,1,rn).(P_{21}(K,L,abort,restart,tail,first,last,tag,rn)\\
&\qquad\qquad + P_{22}(K,L,abort,restart,cons(head,tail),first,last,tag,rn))\\
&\quad \mid (K(first\ \ last\ \ tag\ \ m).\tau^2.copy\_bool(tag,rtag).\tau.\\
&\qquad\quad (Q_{11}(K,L,abort,restart,m,rtag,end,t2enabled)\\
&\qquad\quad + Q_{12}(K,L,abort,restart,m,rtag,end,t2enabled))\\
&\qquad + restart.\tau.Q(K,L,abort,restart)))
\end{aligned}
$$

$$
\begin{aligned}
&A_5(head{,}tail) \stackrel{\cdot}{\sim} \tau.add(rn,1,rn).(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\quad ((P_{21}(K,L,abort,restart,tail,first,last,tag,rn)\\
&\qquad + P_{22}(K,L,abort,restart,cons(head,tail),first,last,tag,rn))\\
&\quad \mid (K(first\ \ last\ \ tag\ \ m).\tau^2.copy\_bool(tag,rtag).\tau.\\
&\qquad\quad (Q_{11}(K,L,abort,restart,m,rtag,end,t2enabled)\\
&\qquad\quad + Q_{12}(K,L,abort,restart,m,rtag,end,t2enabled))\\
&\qquad + restart.\tau.Q(K,L,abort,restart)))
\end{aligned}
$$

$$
\begin{aligned}
&\stackrel{\cdot}{\sim} \tau^2.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\qquad (L.\tau.[\![0]\!]rn.Not(tag).[\![False]\!]first.PP(K,L,abort,restart,tail,first,last,tag,rn)\\
&\qquad + \tau^2.(P_{221}(K,L,abort,restart,last)\\
&\qquad\qquad + P_{222}(K,L,abort,restart,cons(head,tail),first,last,tag,rn))\\
&\qquad \mid (K(first\ \ last\ \ tag\ \ m).\tau^2.copy\_bool(tag,rtag).\tau.\\
&\qquad\qquad (Q_{11}(K,L,abort,restart,m,rtag,end,t2enabled)\\
&\qquad\qquad + Q_{12}(K,L,abort,restart,m,rtag,end,t2enabled))\\
&\qquad + restart.\tau.Q(K,L,abort,restart)))
\end{aligned}
$$

Sachant que le message est perdu, l'émetteur ne peut recevoir d'acquittement.

$$
\begin{aligned}
&\stackrel{\cdot}{\sim} \tau^2.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\qquad (\tau^2.(P_{221}(K,L,abort,restart,last)\\
&\qquad\qquad + P_{222}(K,L,abort,restart,cons(head,tail),first,last,tag,rn))\\
&\qquad \mid (K(first\ \ last\ \ tag\ \ m).\tau^2.copy\_bool(tag,rtag).\tau.\\
&\qquad\qquad (Q_{11}(K,L,abort,restart,m,rtag,end,t2enabled)\\
&\qquad\qquad + Q_{12}(K,L,abort,restart,m,rtag,end,t2enabled))\\
&\qquad + restart.\tau.Q(K,L,abort,restart)))
\end{aligned}
$$

$$
\begin{aligned}
&\stackrel{\cdot}{\sim} \tau^4.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\qquad ((P_{2211}(K,L,abort,restart,last) + P_{2212}(K,L,abort,restart,last)\\
&\qquad + P_2(K,L,abort,restart,cons(head,tail),first,last,tag,rn))\\
&\qquad \mid (K(first\ \ last\ \ tag\ \ m).\tau^2.copy\_bool(tag,rtag).\tau.\\
&\qquad\qquad (Q_{11}(K,L,abort,restart,m,rtag,end,t2enabled)\\
&\qquad\qquad + Q_{12}(K,L,abort,restart,m,rtag,end,t2enabled))\\
&\qquad + restart.\tau.Q(K,L,abort,restart)))
\end{aligned}
$$

$$
\begin{array}{l}
\dot{\sim}\ \tau^4.(\nu\ \ K\ \ L\ \ abort\ \ restart) \\
\qquad\qquad ((\tau.\overline{Conf}\ \ DONTKNOW.\overline{abort}.P(K, L, abort, restart) \\
\qquad\qquad +\ \tau.\overline{Conf}\ \ NOTOK.\overline{abort}.P(K, L, abort, restart) \\
\qquad\qquad +\ \tau^3.\overline{K}\ \ first\ \ last\ \ tag\ \ head.\tau.add(rn, 1, rn). \\
\qquad\qquad\qquad (P_{21}(K, L, abort, restart, tail, first, last, tag, rn) \\
\qquad\qquad\qquad +\ P_{22}(K, L, abort, restart, cons(head, tail), first, last, tag, rn))) \\
\qquad\qquad \mid (K(first\ \ last\ \ tag\ \ m).\tau^2.copy\_bool(tag, rtag).\tau. \\
\qquad\qquad\qquad (Q_{11}(K, L, abort, restart, m, rtag, end, t2enabled) \\
\qquad\qquad\qquad +\ Q_{12}(K, L, abort, restart, m, rtag, end, t2enabled)) \\
\qquad\qquad +\ restart.\tau.Q(K, L, abort, restart)))
\end{array}
$$

Quatre situations sont possibles:

- Le premier message du fichier est retransmis mais il est encore perdu car l'événement *petre_msg* se reproduit. L'émetteur interrompt le transfert et envoie une confirmation $DONTKNOW$ au producteur car le fichier contient un seul message (A_{51}).
- Le premier message du fichier est retransmis mais il est encore perdu car l'événement *petre_msg* se reproduit. L'émetteur interrompt le transfert et envoie une confirmation $NOTOK$ au producteur car le fichier contient d'autres messages (A_{51}).
- Le premier message du fichier est retransmis mais il est encore perdu car l'événement *petre_msg* se reproduit. L'émetteur retransmet ce message ($A_5(head, tail)$).
- Le premier message du fichier est retransmis et il est reçu par le récepteur ($A_3(head, tail)$).

$$
\begin{array}{l}
\dot{\sim}\ \tau^4.(\tau.\overline{Conf}\ \ DONTKNOW.(\nu\ \ K\ \ L\ \ abort\ \ restart) \\
\qquad\qquad (\overline{abort}.P(K, L, abort, restart) \\
\qquad\qquad \mid (K(first\ \ last\ \ tag\ \ m).\tau^2.copy\_bool(tag, rtag).\tau. \\
\qquad\qquad\qquad (Q_{11}(K, L, abort, restart, m, rtag, end, t2enabled) \\
\qquad\qquad\qquad +\ Q_{12}(K, L, abort, restart, m, rtag, end, t2enabled)) \\
\qquad\qquad +\ restart.\tau.Q(K, L, abort, restart))) \\
\qquad +\ \tau.\overline{Conf}\ \ NOTOK.(\nu\ \ K\ \ L\ \ abort\ \ restart) \\
\qquad\qquad (\overline{abort}.P(K, L, abort, restart) \\
\qquad\qquad \mid (K(first\ \ last\ \ tag\ \ m).\tau^2.copy\_bool(tag, rtag).\tau. \\
\qquad\qquad\qquad (Q_{11}(K, L, abort, restart, m, rtag, end, t2enabled) \\
\qquad\qquad\qquad +\ Q_{12}(K, L, abort, restart, m, rtag, end, t2enabled)) \\
\qquad\qquad +\ restart.\tau.Q(K, L, abort, restart))) \\
\qquad +\ \tau^3.\tau.(\nu\ \ K\ \ L\ \ abort\ \ restart) \\
\qquad\qquad (\tau.add(rn, 1, rn).(P_{21}(K, L, abort, restart, tail, first, last, tag, rn) \\
\qquad\qquad\qquad\qquad +\ P_{22}(K, L, abort, restart, cons(head, tail), first, last, tag, rn)) \\
\qquad\qquad \mid \tau^2.copy\_bool(tag, rtag).\tau.(Q_{11}(K, L, abort, restart, head, rtag, end, t2enabled) \\
\qquad\qquad\qquad +\ Q_{12}(K, L, abort, restart, head, rtag, end, t2enabled))) \\
\qquad +\ \tau^3.\tau.(\nu\ \ K\ \ L\ \ abort\ \ restart) \\
\qquad\qquad (\tau.add(rn, 1, rn).(P_{21}(K, L, abort, restart, tail, first, last, tag, rn) \\
\qquad\qquad\qquad\qquad +\ P_{22}(K, L, abort, restart, cons(head, tail), first, last, tag, rn))
\end{array}
$$

$$
\begin{aligned}
&\mid (K(first\ \ last\ \ tag\ \ m).\tau^2.copy\_bool(tag, rtag).\tau.\\
&\qquad (Q_{11}(K, L, abort, restart, m, rtag, end, t2enabled)\\
&\qquad +\ Q_{12}(K, L, abort, restart, m, rtag, end, t2enabled))\\
&+\ restart.\tau.Q(K, L, abort, restart))))
\end{aligned}
$$

$$
\begin{aligned}
A_5(head,tail)\ \dot{\sim}\ \tau^4.(&\tau.\overline{Conf}\ \ DONTKNOW.A_{51}\\
+\ &\tau.\overline{Conf}\ \ NOTOK.A_{51}\\
+\ &A_3(head, tail)\\
+\ &\tau^4.A_5(head, tail))
\end{aligned}
$$

Comme la situation A_{51} n'est pas envisagée (voir ci-dessous), alors:

$$A_5(head, tail)\ \dot{\sim}\ \tau^4.(A_3(head, tail)\ +\ \tau^4.A_5(head, tail))$$

Considérons la partie A_{51}:

$$
\begin{aligned}
A_{51}\ \stackrel{def}{=}\ &(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&(\overline{abort}.P(K, L, abort, restart)\\
&\mid (K(first\ \ last\ \ tag\ \ m).\tau^2.copy\_bool(tag, rtag).\tau.\\
&\qquad (Q_{11}(K, L, abort, restart, m, rtag, end, t2enabled)\\
&\qquad +\ Q_{12}(K, L, abort, restart, m, rtag, end, t2enabled))\\
&+\ restart.\tau.Q(K, L, abort, restart)))
\end{aligned}
$$

Si le premier message du fichier n'arrive jamais chez le récepteur, l'émetteur décide d'interrompre le transfert et envoie le signal *abort* au récepteur. Comme *timer*2 n'est pas déclenché au démarrage, le récepteur ne peut recevoir le signal *timeout*2 et par conséquence il ne peut réaliser un rendez-vouz avec l'émetteur sur le signal *abort*. Ceci nous amène donc à une situation de **deadlock**.

Il n'exite pas de solution à ce problème en milieu distribué car d'une part l'émetteur n'a aucun moyen pour détecter le fait que le premier message n'est jamais reçu par le récepteur. D'autre part, le récepteur n'a aucun moyen pour être au courant de ce début de transfert et pour réagir à l'abort.

On va donc supposer que ce cas extrême ne se produira jamais et que le premier message transmis par l'émetteur finit par arriver chez le récepteur.

Considérons la partie $A_3(head, tail)$ (le premier message du fichier est reçu par le récepteur):

$$
\begin{aligned}
A_3(head,tail)\ \stackrel{def}{=}\ &\tau^3.\tau.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&(\tau.add(rn, 1, rn).(P_{21}(K, L, abort, restart, tail, first, last, tag, rn)\\
&\qquad +\ P_{22}(K, L, abort, restart, cons(head, tail), first, last, tag, rn))\\
&\mid \tau^2.copy\_bool(tag, rtag).\tau.(Q_{11}(K, L, abort, restart, head, rtag, end, t2enabled)\\
&\qquad +\ Q_{12}(K, L, abort, restart, head, rtag, end, t2enabled)))
\end{aligned}
$$

$$A_3(head,tail) \stackrel{\cdot}{\sim} \tau^4.\tau.\tau^2.add(rn,1,rn).copy\_bool(tag,rtag).\tau.(\nu\ K\ L\ abort\ restart)$$
$$((P_{21}(K,L,abort,restart,tail,first,last,tag,rn)$$
$$+\ P_{22}(K,L,abort,restart,cons(head,tail),first,last,tag,rn))$$
$$\mid (Q_{11}(K,L,abort,restart,head,rtag,end,t2enabled)$$
$$+\ Q_{12}(K,L,abort,restart,head,rtag,end,t2enabled)))$$

$$\stackrel{\cdot}{\sim} \tau^{10}.(\nu\ K\ L\ abort\ restart)$$
$$((P_{21}(K,L,abort,restart,tail,first,last,tag,rn)$$
$$+\ P_{22}(K,L,abort,restart,cons(head,tail),first,last,tag,rn))$$
$$\mid (Q_{11}(K,L,abort,restart,head,rtag,end,t2enabled)$$
$$+\ Q_{12}(K,L,abort,restart,head,rtag,end,t2enabled)))$$

$$A_3(head,tail) \stackrel{\cdot}{\sim} \tau^{10}.AA_3(head,tail)$$

Considérons la partie $AA_3(head,tail)$ (le premier message du fichier est reçu par le récepteur):

$$AA_3(head,tail) \stackrel{def}{=} (\nu\ K\ L\ abort\ restart)$$
$$((P_{21}(K,L,abort,restart,tail,first,last,tag,rn)$$
$$+\ P_{22}(K,L,abort,restart,cons(head,tail),first,last,tag,rn))$$
$$\mid (Q_{11}(K,L,abort,restart,head,rtag,end,t2enabled)$$
$$+\ Q_{12}(K,L,abort,restart,head,rtag,end,t2enabled)))$$

$$AA_3(head,tail) \stackrel{\cdot}{\sim} (\nu\ K\ L\ abort\ restart)$$
$$((L.\tau.[\![0]\!]rn.Not(tag).[\![False]\!]first.PP(K,L,abort,restart,tail,first,last,tag,rn)$$
$$+\ \tau^2.(P_{221}(K,L,abort,restart,last)$$
$$+\ P_{222}(K,L,abort,restart,cons(head,tail),first,last,tag,rn)))$$
$$\mid$$
$$(\tau.\overline{Ind}\ \ head\ \ LAST.\overline{L}.\tau.[\![True]\!]t2enabled.QQ(K,L,abort,restart,rtag,end,t2enabled)$$
$$+\ \tau.\overline{Ind}\ \ head\ \ FIRST.\overline{L}.\tau.[\![True]\!]t2enabled.QQ(K,L,abort,restart,rtag,end,t2enabled)))$$

Quatre situations sont possibles:

- Le récepteur délivre le premier message avec une indication $FIRST$ et envoie un acquittement qui est reçu par l'émetteur ($A_{31}(tail)$).
- Le récepteur délivre le premier message avec une indication $LAST$ et envoie un acquittement qui est reçu par l'émetteur ($A_{31}(tail)$).
- Le récepteur délivre le premier message avec une indication $FIRST$ et envoie un acquittement qui est perdu car l'événement *perte_ack* (i.e, *L.perte_ack*) se produit ($A_{32}(head,tail)$).
- Le récepteur délivre le premier message avec une indication $LAST$ et envoie un acquittement qui est perdu car l'événement *perte_ack* se produit ($A_{32}(head,tail)$).

$$
\begin{aligned}
\dot\sim\ & (\tau.\overline{Ind}\ \ head\ \ LAST.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\qquad (L.\tau.[\![0]\!]rn.Not(tag).[\![False]\!]first.PP(K,L,abort,restart,tail,first,last,tag,rn)\\
&\qquad \mid \overline{L}.\tau.[\![True]\!]t2enabled.QQ(K,L,abort,restart,rtag,end,t2enabled))\\
&\quad + \tau.\overline{Ind}\ \ head\ \ FIRST.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\qquad (L.\tau.[\![0]\!]rn.Not(tag).[\![False]\!]first.PP(K,L,abort,restart,tail,first,last,tag,rn)\\
&\qquad \mid \overline{L}.\tau.[\![True]\!]t2enabled.QQ(K,L,abort,restart,rtag,end,t2enabled))\\
&\quad + \tau^2.\tau.\overline{Ind}\ \ head\ \ LAST.\tau.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\qquad ((P_{221}(K,L,abort,restart,last)\\
&\qquad + P_{222}(K,L,abort,restart,cons(head,tail),first,last,tag,rn))\\
&\qquad \mid \tau.[\![True]\!]t2enabled.QQ(K,L,abort,restart,rtag,end,t2enabled))\\
&\quad + \tau^2.\tau.\overline{Ind}\ \ head\ \ FIRST.\tau.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\qquad ((P_{221}(K,L,abort,restart,last)\\
&\qquad + P_{222}(K,L,abort,restart,cons(head,tail),first,last,tag,rn))\\
&\qquad \mid \tau.[\![True]\!]t2enabled.QQ(K,L,abort,restart,rtag,end,t2enabled)))
\end{aligned}
$$

$$
\begin{aligned}
AA_3(head,tail)\ \dot\sim\ & (\tau.\overline{Ind}\ \ head\ \ LAST.A_{31}(tail)\\
& + \tau.\overline{Ind}\ \ head\ \ FIRST.A_{31}(tail)\\
& + \tau^3.\overline{Ind}\ \ head\ \ LAST.A_{32}(head,tail)\\
& + \tau^3.\overline{Ind}\ \ head\ \ FIRST.A_{32}(head,tail))
\end{aligned}
$$

Considérons la partie $A_{31}(tail)$ (l'acquittement du message est reçu par l'émetteur):

$$
\begin{aligned}
A_{31}(tail)\ \stackrel{def}{=}\ & (\nu\ \ K\ \ L\ \ abort\ \ restart)\\
& (L.\tau.[\![0]\!]rn.Not(tag).[\![False]\!]first.PP(K,L,abort,restart,tail,first,last,tag,rn)\\
& \mid \overline{L}.\tau.[\![True]\!]t2enabled.QQ(K,L,abort,restart,rtag,end,t2enabled))
\end{aligned}
$$

$$
\begin{aligned}
A_{31}(tail)\ \dot\sim\ & \tau.\tau.\tau.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
& ([\![0]\!]rn.Not(tag).[\![False]\!]first.PP(K,L,abort,restart,tail,first,last,tag,rn)\\
& \mid [\![True]\!]t2enabled.QQ(K,L,abort,restart,rtag,end,t2enabled))
\end{aligned}
$$

$$
\begin{aligned}
\dot\sim\ & \tau^3.[\![0]\!]rn.Not(tag).[\![False]\!]first.[\![True]\!]t2enabled.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
& (PP(K,L,abort,restart,tail,first,last,tag,rn)\\
& \mid QQ(K,L,abort,restart,rtag,end,t2enabled))
\end{aligned}
$$

$$
\begin{aligned}
\dot\sim\ & \tau^7.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\quad ((P_1(K,L,abort,restart) + P_2(K,L,abort,restart,cons(hd,tl),first,last,tag,rn))\\
&\quad \mid (QQ_1(K,L,abort,restart,rtag,end,t2enabled) + QQ_2(K,L,abort,restart,end)\\
&\qquad + QQ_3(K,L,abort,restart,t2enabled)))
\end{aligned}
$$

Ici, hd et tl représentent respectivement la tête et le reste du fichier $tail$ (i.e, $hd = head\_list(tail)$ et $tl = tail\_list(tail)$).

$$
\begin{aligned}
\dot\sim\ & \tau^7.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\quad ((\tau.\overline{Conf}\ \ OK.\overline{restart}.P(K,L,abort,restart)\\
&\quad + \tau^3.\overline{K}\ \ first\ \ last\ \ tag\ \ hd.\tau.add(rn,1,rn).
\end{aligned}
$$

$$
\begin{aligned}
&\qquad (P_{21}(K, L, abort, restart, tl, first, last, tag, rn) \\
&\qquad\quad + \ P_{22}(K, L, abort, restart, cons(hd, tl), first, last, tag, rn))) \\
&\quad \mid (K(first\ \ last\ \ tag\ \ m).\tau.(QQ_{11}(K, L, abort, restart, rtag, end, t2enabled) \\
&\qquad\quad + \ QQ_{12}(K, L, abort, restart, first, last, tag, m, rtag, end, t2enabled)) \\
&\quad + \ \tau^2.abort.\tau.(Q(K, L, abort, restart) \ + \ QQ_{22}(K, L, abort, restart)) \\
&\quad + \ restart.\tau^2.Q(K, L, abort, restart)))
\end{aligned}
$$

Trois situations sont possibles:

- Le transfert du fichier est terminé, l'émetteur émet le signal *restart* afin que le récepteur se réinitialise (A_{314}).
- L'émetteur transmet le message suivant du fichier, le récepteur le reçoit ($A_{315}(hd, tl)$).
- L'émetteur transmet le message suivant du fichier, mais ce message est perdu car l'événement *perte_msg* se produit ($A_{319}(hd, tl)$).

$$
\begin{aligned}
\dot{\sim}\ & \tau^7.(\tau.\overline{Conf}\ \ OK.(\nu\ \ K\ \ L\ \ abort\ \ restart) \\
&\quad (\overline{restart}.P(K, L, abort, restart) \mid restart.\tau^2.Q(K, L, abort, restart)) \\
&\ + \ \tau^3.\tau.(\nu\ \ K\ \ L\ \ abort\ \ restart) \\
&\quad (\tau.add(rn, 1, rn).(P_{21}(K, L, abort, restart, tl, first, last, tag, rn) \\
&\qquad + \ P_{22}(K, L, abort, restart, cons(hd, tl), first, last, tag, rn)) \\
&\quad \mid \tau.(QQ_{11}(K, L, abort, restart, rtag, end, t2enabled) \\
&\qquad + \ QQ_{12}(K, L, abort, restart, first, last, tag, hd, rtag, end, t2enabled))) \\
&\ + \ \tau^3.\tau.(\nu\ \ K\ \ L\ \ abort\ \ restart) \\
&\quad (\tau.add(rn, 1, rn).(P_{21}(K, L, abort, restart, tl, first, last, tag, rn) \\
&\qquad + \ P_{22}(K, L, abort, restart, cons(hd, tl), first, last, tag, rn)) \\
&\quad \mid (K(first\ \ last\ \ tag\ \ m).\tau.(QQ_{11}(K, L, abort, restart, rtag, end, t2enabled) \\
&\qquad + \ QQ_{12}(K, L, abort, restart, first, last, tag, m, rtag, end, t2enabled)) \\
&\quad + \ \tau^2.abort.\tau.(Q(K, L, abort, restart) \ + \ QQ_{22}(K, L, abort, restart)) \\
&\quad + \ restart.\tau^2.Q(K, L, abort, restart))))
\end{aligned}
$$

$$
A_{31}(tail) \ \dot{\sim}\ \tau^7.(A_{314} \ + \ A_{315}(hd, tl) \ + \ A_{319}(hd, tl))
$$

Considérons la partie A_{314} (le transfert du fichier est terminé, l'émetteur et le récepteur se synchronisent à l'aide du signal *restart* pour commencer un nouveau transfert de fichier):

$$
\begin{aligned}
A_{314} \stackrel{def}{=}\ & \tau.\overline{Conf}\ \ OK.(\nu\ \ K\ \ L\ \ abort\ \ restart) \\
& (\overline{restart}.P(K, L, abort, restart) \mid restart.\tau^2.Q(K, L, abort, restart))
\end{aligned}
$$

$$
\begin{aligned}
A_{314} \ \dot{\sim}\ & \tau.\overline{Conf}\ \ OK.\tau.\tau^2.(\nu\ \ K\ \ L\ \ abort\ \ restart) \\
& (P(K, L, abort, restart) \mid Q(K, L, abort, restart))
\end{aligned}
$$

$$
A_{314} \ \dot{\sim}\ \tau.\overline{Conf}\ \ OK.\tau^3.System
$$

Considérons la partie $A_{319}(hd, tl)$ (le message pas encore délivré est transmis, mais il est perdu car l'événement *perte_msg* se produit):

$$
\begin{aligned}
A_{319}(hd,tl) \stackrel{def}{=}\ & \tau^3.\tau.(\nu\ \ K\ \ L\ \ abort\ \ restart) \\
& (\tau.add(rn,1,rn).(P_{21}(K,L,abort,restart,tl,first,last,tag,rn) \\
& \quad +\ P_{22}(K,L,abort,restart,cons(hd,tl),first,last,tag,rn)) \\
& \mid (K(first\ \ last\ \ tag\ \ m).\tau.(QQ_{11}(K,L,abort,restart,rtag,end,t2enabled) \\
& \quad +\ QQ_{12}(K,L,abort,restart,first,last,tag,m,rtag,end,t2enabled)) \\
& +\ \tau^2.abort.\tau.(Q(K,L,abort,restart)\ +\ QQ_{22}(K,L,abort,restart)) \\
& +\ restart.\tau^2.Q(K,L,abort,restart)))
\end{aligned}
$$

$$
\begin{aligned}
A_{319}(hd,tl)\ \dot{\sim}\ & \tau^4.\tau.add(rn,1,rn).(\nu\ \ K\ \ L\ \ abort\ \ restart) \\
& ((P_{21}(K,L,abort,restart,tl,first,last,tag,rn) \\
& \quad +\ P_{22}(K,L,abort,restart,cons(hd,tl),first,last,tag,rn)) \\
& \mid (K(first\ \ last\ \ tag\ \ m).\tau.(QQ_{11}(K,L,abort,restart,rtag,end,t2enabled) \\
& \quad +\ QQ_{12}(K,L,abort,restart,first,last,tag,m,rtag,end,t2enabled)) \\
& +\ \tau^2.abort.\tau.(Q(K,L,abort,restart)\ +\ QQ_{22}(K,L,abort,restart)) \\
& +\ restart.\tau^2.Q(K,L,abort,restart)))
\end{aligned}
$$

$$
\begin{aligned}
\dot{\sim}\ & \tau^6.(\nu\ \ K\ \ L\ \ abort\ \ restart) \\
& ((L.\tau.[\![0]\!]rn.Not(tag).[\![False]\!]first.PP(K,L,abort,restart,tl,first,last,tag,rn) \\
& +\ \tau^2.(P_{221}(K,L,abort,restart,last) \\
& \qquad +\ P_{222}(K,L,abort,restart,cons(hd,tl),first,last,tag,rn))) \\
& \mid (K(first\ \ last\ \ tag\ \ m).\tau.(QQ_{11}(K,L,abort,restart,rtag,end,t2enabled) \\
& \qquad +\ QQ_{12}(K,L,abort,restart,first,last,tag,m,rtag,end,t2enabled)) \\
& +\ \tau^2.abort.\tau.(Q(K,L,abort,restart)\ +\ QQ_{22}(K,L,abort,restart)) \\
& +\ restart.\tau^2.Q(K,L,abort,restart)))
\end{aligned}
$$

Sachant que le message est perdu, l'émetteur ne peut recevoir d'acquittement.

$$
\begin{aligned}
\dot{\sim}\ & \tau^6.(\nu\ \ K\ \ L\ \ abort\ \ restart) \\
& (\tau^2.(P_{221}(K,L,abort,restart,last) \\
& \qquad +\ P_{222}(K,L,abort,restart,cons(hd,tl),first,last,tag,rn)) \\
& \mid (K(first\ \ last\ \ tag\ \ m).\tau.(QQ_{11}(K,L,abort,restart,rtag,end,t2enabled) \\
& \qquad +\ QQ_{12}(K,L,abort,restart,first,last,tag,m,rtag,end,t2enabled)) \\
& +\ \tau^2.abort.\tau.(Q(K,L,abort,restart)\ +\ QQ_{22}(K,L,abort,restart)) \\
& +\ restart.\tau^2.Q(K,L,abort,restart)))
\end{aligned}
$$

$$
\begin{aligned}
\dot{\sim}\ & \tau^8.(\nu\ \ K\ \ L\ \ abort\ \ restart) \\
& ((P_{2211}(K,L,abort,restart,last)\ +\ P_{2212}(K,L,abort,restart,last) \\
& \qquad +\ P_2(K,L,abort,restart,cons(hd,tl),first,last,tag,rn)) \\
& \mid (K(first\ \ last\ \ tag\ \ m).\tau.(QQ_{11}(K,L,abort,restart,rtag,end,t2enabled) \\
& \qquad +\ QQ_{12}(K,L,abort,restart,first,last,tag,m,rtag,end,t2enabled)) \\
& +\ \tau^2.abort.\tau.(Q(K,L,abort,restart)\ +\ QQ_{22}(K,L,abort,restart)) \\
& +\ restart.\tau^2.Q(K,L,abort,restart)))
\end{aligned}
$$

$$
\begin{aligned}
\stackrel{\cdot}{\sim}\ & \tau^8.(\nu\ K\ L\ abort\ restart)\\
&\quad ((\tau.\overline{Conf}\ DONTKNOW.\overline{abort}.P(K,L,abort,restart)\\
&\quad +\ \tau.\overline{Conf}\ NOTOK.\overline{abort}.P(K,L,abort,restart)\\
&\quad +\ \tau^3.\overline{K}\ first\ last\ tag\ hd.\tau.add(rn,1,rn).\\
&\qquad (P_{21}(K,L,abort,restart,tl,first,last,tag,rn)\\
&\qquad +\ P_{22}(K,L,abort,restart,cons(hd,tl),first,last,tag,rn)))\\
&\quad \mid (K(first\ last\ tag\ m).\tau.(QQ_{11}(K,L,abort,restart,rtag,end,t2enabled)\\
&\qquad +\ QQ_{12}(K,L,abort,restart,first,last,tag,m,rtag,end,t2enabled))\\
&\quad +\ \tau^2.abort.\tau.(Q(K,L,abort,restart)\ +\ QQ_{22}(K,L,abort,restart))\\
&\quad +\ restart.\tau^2.Q(K,L,abort,restart)))
\end{aligned}
$$

Quatre situations sont possibles:

- L'émetteur interrompt le transfert et envoie une confirmation $DONTKNOW$ au producteur car il s'agit du dernier message du fichier; le récepteur reçoit le signal *abort* et envoie une indication d'erreur au consommateur car ce message n'a jamais été délivré au consommateur (i.e, la variable *end* vaut $False$) (A_{3192}).

- L'émetteur interrompt le transfert et envoie une confirmation $NOTOK$ au producteur car il s'agit d'un message intermédiaire du fichier; le récepteur reçoit le signal *abort* et envoie une indication d'erreur au consommateur (A_{3192}).

- L'émetteur retransmet le message, le récepteur le reçoit ($A_{315}(hd,tl)$).

- L'émetteur retransmet le message mais ce message est encore perdu car l'événement *perte_msg* se produit ($A_{319}(hd,tl)$).

$$
\begin{aligned}
\stackrel{\cdot}{\sim}\ & \tau^8.(\tau^2.\tau.\overline{Conf}\ DONTKNOW.(\nu\ K\ L\ abort\ restart)\\
&\quad (\overline{abort}.P(K,L,abort,restart) \mid abort.\tau.(Q(K,L,abort,restart)\ +\ QQ_{22}(K,L,abort,restart)))\\
&\ +\ \tau^2.\tau.\overline{Conf}\ NOTOK.(\nu\ K\ L\ abort\ restart)\\
&\quad (\overline{abort}.P(K,L,abort,restart) \mid abort.\tau.(Q(K,L,abort,restart)\ +\ QQ_{22}(K,L,abort,restart)))\\
&\ +\ \tau^3.\tau.(\nu\ K\ L\ abort\ restart)\\
&\quad (\tau.add(rn,1,rn).(P_{21}(K,L,abort,restart,tl,first,last,tag,rn)\\
&\qquad +\ P_{22}(K,L,abort,restart,cons(hd,tl),first,last,tag,rn))\\
&\quad \mid \tau.(QQ_{11}(K,L,abort,restart,rtag,end,t2enabled)\\
&\qquad +\ QQ_{12}(K,L,abort,restart,first,last,tag,hd,rtag,end,t2enabled)))\\
&\ +\ \tau^3.\tau.(\nu\ K\ L\ abort\ restart)\\
&\quad (\tau.add(rn,1,rn).(P_{21}(K,L,abort,restart,tl,first,last,tag,rn)\\
&\qquad +\ P_{22}(K,L,abort,restart,cons(hd,tl),first,last,tag,rn))\\
&\quad \mid (K(first\ last\ tag\ m).\tau.(QQ_{11}(K,L,abort,restart,rtag,end,t2enabled)\\
&\qquad +\ QQ_{12}(K,L,abort,restart,first,last,tag,m,rtag,end,t2enabled))\\
&\quad +\ \tau^2.abort.\tau.(Q(K,L,abort,restart)\ +\ QQ_{22}(K,L,abort,restart))\\
&\quad +\ restart.\tau^2.Q(K,L,abort,restart))))
\end{aligned}
$$

$$
\begin{aligned}
A_{319}(hd,tl)\ \stackrel{\cdot}{\sim}\ & \tau^8.(\tau^3.\overline{Conf}\ DONTKNOW.A_{3192}\\
&\ +\ \tau^3.\overline{Conf}\ NOTOK.A_{3192}
\end{aligned}
$$

$$+ \ A_{315}(hd, tl)$$
$$+ \ A_{319}(hd, tl))$$

Considérons la partie A_{3192} (abort du transfert par l'émetteur avec une confirmation $NOTOK$ ou $DONTKNOW$ et une indication d'erreur côté récepteur, i.e, l'abort a eu lieu lors du transfert d'un message intermédiaire du fichier ou lors du transfert du dernier message du fichier, ce message n'étant jamais arrivé côté récepteur):

$$A_{3192} \stackrel{def}{=} (\nu \ K \ L \ abort \ restart)$$
$$(\overline{abort}.P(K, L, abort, restart) \mid abort.\tau.(Q(K, L, abort, restart) \ + \ QQ_{22}(K, L, abort, restart)))$$

Que le message soit le dernier message du fichier ou pas, comme ce message n'est jamais arrivé chez le récepteur, le récepteur doit envoyer une indication d'erreur au consommateur.

$$A_{3192} \stackrel{\cdot}{\sim} (\nu \ K \ L \ abort \ restart)$$
$$(\overline{abort}.P(K, L, abort, restart) \mid abort.\tau.QQ_{22}(K, L, abort, restart))$$

$$\stackrel{\cdot}{\sim} (\nu \ K \ L \ abort \ restart)$$
$$(\overline{abort}.P(K, L, abort, restart) \mid abort.\tau.\overline{Ind\_err}.Q(K, L, abort, restart))$$

$$\stackrel{\cdot}{\sim} \tau.\tau.\overline{Ind\_err}.(\nu \ K \ L \ abort \ restart)$$
$$(P(K, L, abort, restart) \mid Q(K, L, abort, restart))$$

$$A_{3192} \stackrel{\cdot}{\sim} \tau^2.\overline{Ind\_err}.System$$

Considérons la partie $A_{315}(hd, tl)$ (le message suivant du fichier a été retransmis par l'émetteur et reçu pour la première fois par le récepteur):

$$A_{315}(hd,tl) \stackrel{def}{=} \tau^3.\tau.(\nu \ K \ L \ abort \ restart)$$
$$(\tau.add(rn, 1, rn).(P_{21}(K, L, abort, restart, tl, first, last, tag, rn)$$
$$+ \ P_{22}(K, L, abort, restart, cons(hd, tl), first, last, tag, rn))$$
$$\mid \tau.(QQ_{11}(K, L, abort, restart, rtag, end, t2enabled)$$
$$+ \ QQ_{12}(K, L, abort, restart, first, last, tag, hd, rtag, end, t2enabled)))$$

$$A_{315}(hd,tl) \stackrel{\cdot}{\sim} \tau^4.\tau.\tau.add(rn, 1, rn).(\nu \ K \ L \ abort \ restart)$$
$$((P_{21}(K, L, abort, restart, tl, first, last, tag, rn)$$
$$+ \ P_{22}(K, L, abort, restart, cons(hd, tl), first, last, tag, rn))$$
$$\mid (QQ_{11}(K, L, abort, restart, rtag, end, t2enabled)$$
$$+ \ QQ_{12}(K, L, abort, restart, first, last, tag, hd, rtag, end, t2enabled)))$$

$$\stackrel{\cdot}{\sim} \tau^5.\tau^2.(\nu \ K \ L \ abort \ restart)$$
$$((P_{21}(K, L, abort, restart, tl, first, last, tag, rn)$$
$$+ \ P_{22}(K, L, abort, restart, cons(hd, tl), first, last, tag, rn))$$
$$\mid (QQ_{11}(K, L, abort, restart, rtag, end, t2enabled)$$
$$+ \ QQ_{12}(K, L, abort, restart, first, last, tag, hd, rtag, end, t2enabled)))$$

$A_{315}(hd,tl) \;\dot{\sim}\; \tau^5.B(hd,tl)$

Considérons la partie $B(hd,tl)$ (le message suivant du fichier a été retransmis par l'émetteur et reçu pour la première fois par le récepteur):

$$
\begin{aligned}
B(hd,tl) \stackrel{def}{=} \; & \tau^2.(\nu\;\; K\;\; L\;\; abort\;\; restart)\\
& ((P_{21}(K,L,abort,restart,tl,first,last,tag,rn)\\
& +\; P_{22}(K,L,abort,restart,cons(hd,tl),first,last,tag,rn))\\
& \mid (QQ_{11}(K,L,abort,restart,rtag,end,t2enabled)\\
& +\; QQ_{12}(K,L,abort,restart,first,last,tag,hd,rtag,end,t2enabled)))
\end{aligned}
$$

$$
\begin{aligned}
B(hd,tl) \;\dot{\sim}\; & \tau^2.(\nu\;\; K\;\; L\;\; abort\;\; restart)\\
& ((L.\tau.[\![0]\!]rn.Not(tag).[\![False]\!]first.PP(K,L,abort,restart,tl,first,last,tag,rn)\\
& +\; \tau^2.(P_{221}(K,L,abort,restart,last)\\
& \qquad +\; P_{222}(K,L,abort,restart,cons(hd,tl),first,last,tag,rn)))\\
& \mid (\overline{L}.QQ(K,L,abort,restart,rtag,end,t2enabled)\\
& +\; \tau^4.(Q_{11}(K,L,abort,restart,hd,rtag,end,t2enabled)\\
& \qquad +\; \tau.QQ_{122}(K,L,abort,restart,hd,rtag,end,t2enabled))))
\end{aligned}
$$

Le message arrive pour la première fois chez le récepteur. Celui-ci doit envoyer une indication de message au consommateur (i.e, il ne peut s'agir d'une duplication de message).

$$
\begin{aligned}
\dot{\sim}\; & \tau^2.(\nu\;\; K\;\; L\;\; abort\;\; restart)\\
& ((L.\tau.[\![0]\!]rn.Not(tag).[\![False]\!]first.PP(K,L,abort,restart,tl,first,last,tag,rn)\\
& +\; \tau^2.(P_{2211}(K,L,abort,restart,last) \;+\; P_{2212}(K,L,abort,restart,last))\\
& +\; \tau^2.P_2(K,L,abort,restart,cons(hd,tl),first,last,tag,rn))\\
& \mid (\tau^4.\tau.\overline{Ind}\;\; hd\;\; LAST.\overline{L}.\tau.[\![True]\!]t2enabled.QQ(K,L,abort,restart,rtag,end,t2enabled)\\
& +\; \tau^4.\tau.\overline{Ind}\;\; hd\;\; INCOMPLETE.\overline{L}.\tau.[\![True]\!]t2enabled.\\
& \quad QQ(K,L,abort,restart,rtag,end,t2enabled)))
\end{aligned}
$$

$$
\begin{aligned}
\dot{\sim}\; & \tau^2.(\nu\;\; K\;\; L\;\; abort\;\; restart)\\
& ((L.\tau.[\![0]\!]rn.Not(tag).[\![False]\!]first.PP(K,L,abort,restart,tl,first,last,tag,rn)\\
& +\; \tau^2.\tau.\overline{Conf}\;\; DONTKNOW.\overline{abort}.P(K,L,abort,restart)\\
& +\; \tau^2.\tau.\overline{Conf}\;\; NOTOK.\overline{abort}.P(K,L,abort,restart)\\
& +\; \tau^2.P_2(K,L,abort,restart,cons(hd,tl),first,last,tag,rn))\\
& \mid (\tau^5.\overline{Ind}\;\; hd\;\; LAST.\overline{L}.\tau.[\![True]\!]t2enabled.QQ(K,L,abort,restart,rtag,end,t2enabled)\\
& +\; \tau^5.\overline{Ind}\;\; hd\;\; INCOMPLETE.\overline{L}.\tau.[\![True]\!]t2enabled.\\
& \quad QQ(K,L,abort,restart,rtag,end,t2enabled)))
\end{aligned}
$$

Six situations sont possibles:

- L'acquittement émis par le récepteur est reçu par l'émetteur, le récepteur ayant délivré le message avec une indication $LAST$ ($A_{31}(tl)$).

- L'acquittement émis par le récepteur est reçu par l'émetteur, le récepteur ayant délivré le message avec une indication $INCOMPLETE$ ($A_{31}(tl)$).

– L'acquittement est perdu, le récepteur venant de délivrer le message avec une indication $LAST$, l'émetteur interrompt le transfert avec une confirmation $DONTKNOW$; le récepteur n'envoie pas d'indication d'erreur au consommateur car le dernier message a été délivré (B_3).

– L'acquittement émis par le récepteur est perdu, le récepteur venant de délivrer le message avec une indication $INCOMPLETE$, l'émetteur interrompt le transfert avec une confirmation $NOTOK$; le récepteur envoie une indication d'erreur au consommateur (BB_3).

– L'acquittement émis par le récepteur est perdu, l'émetteur retransmet le message qui vient d'être délivré avec une indication $LAST$ par le récepteur ($B_4(hd, tl)$).

– L'acquittement émis par le récepteur est perdu, l'émetteur retransmet le message qui vient d'être délivré avec une indication $INCOMPLETE$ par le récepteur ($B_4(hd, tl)$).

$$
\begin{array}{l}
\dot{\sim}\ (\tau^7.\overline{Ind}\ hd\ LAST.(\nu\ K\ L\ abort\ restart) \\
\qquad (L.\tau.[\![0]\!]rn.Not(tag).[\![False]\!]first.PP(K, L, abort, restart, tl, first, last, tag, rn) \\
\qquad \mid \overline{L}.\tau.[\![True]\!]t2enabled.QQ(K, L, abort, restart, rtag, end, t2enabled)) \\
\quad + \ \tau^7.\overline{Ind}\ hd\ INCOMPLETE.(\nu\ K\ L\ abort\ restart) \\
\qquad (L.\tau.[\![0]\!]rn.Not(tag).[\![False]\!]first.PP(K, L, abort, restart, tl, first, last, tag, rn) \\
\qquad \mid \overline{L}.\tau.[\![True]\!]t2enabled.QQ(K, L, abort, restart, rtag, end, t2enabled)) \\
\quad + \ \tau^5.\tau^5.\overline{Ind}\ hd\ LAST.\tau.\overline{Conf}\ DONTKNOW.(\nu\ K\ L\ abort\ restart) \\
\qquad (\overline{abort}.P(K, L, abort, restart) \\
\qquad \mid \tau.[\![True]\!]t2enabled.QQ(K, L, abort, restart, rtag, end, t2enabled)) \\
\quad + \ \tau^5.\tau^5.\overline{Ind}\ hd\ INCOMPLETE.\tau.\overline{Conf}\ NOTOK.(\nu\ K\ L\ abort\ restart) \\
\qquad (\overline{abort}.P(K, L, abort, restart) \\
\qquad \mid \tau.[\![True]\!]t2enabled.QQ(K, L, abort, restart, rtag, end, t2enabled)) \\
\quad + \ \tau^4.\tau^5.\overline{Ind}\ hd\ LAST.\tau.(\nu\ K\ L\ abort\ restart) \\
\qquad (P_2(K, L, restart, cons(hd, tl), first, last, tag, rn) \\
\qquad \mid \tau.[\![True]\!]t2enabled.QQ(K, L, abort, restart, rtag, end, t2enabled)) \\
\quad + \ \tau^4.\tau^5.\overline{Ind}\ hd\ INCOMPLETE.\tau.(\nu\ K\ L\ abort\ restart) \\
\qquad (P_2(K, L, restart, cons(hd, tl), first, last, tag, rn) \\
\qquad \mid \tau.[\![True]\!]t2enabled.QQ(K, L, abort, restart, rtag, end, t2enabled)))
\end{array}
$$

$$
\begin{array}{rl}
B(hd, tl)\ \dot{\sim} & \tau^7.\overline{Ind}\ hd\ LAST.A_{31}(tl) \\
& + \ \tau^7.\overline{Ind}\ hd\ INCOMPLETE.A_{31}(tl) \\
& + \ \tau^{10}.\overline{Ind}\ hd\ LAST.\tau.\overline{Conf}\ DONTKNOW.B_3 \\
& + \ \tau^{10}.\overline{Ind}\ hd\ INCOMPLETE.\tau.\overline{Conf}\ NOTOK.BB_3 \\
& + \ \tau^9.\overline{Ind}\ hd\ LAST.B_4(hd, tl) \\
& + \ \tau^9.\overline{Ind}\ hd\ INCOMPLETE.B_4(hd, tl)
\end{array}
$$

Considérons la partie B_3 (le dernier message vient d'être délivré car jamais parvenu auparavant chez le récepteur mais l'acquittement correspondant est perdu, l'émetteur interrompt le transfert):

$B_3 \stackrel{def}{=} (\nu\ K\ L\ abort\ restart)$

$$(\overline{abort}.P(K,L,abort,restart)$$
$$\mid \tau.[\![True]\!]t2enabled.QQ(K,L,abort,restart,rtag,end,t2enabled))$$

$$B_3 \stackrel{\cdot}{\sim} \tau.(\nu\ K\ L\ abort\ restart)$$
$$(\overline{abort}.P(K,L,abort,restart) \mid [\![True]\!]t2enabled.QQ(K,L,abort,restart,rtag,end,t2enabled))$$

$$\stackrel{\cdot}{\sim} \tau.[\![True]\!]t2enabled.(\nu\ K\ L\ abort\ restart)$$
$$(\overline{abort}.P(K,L,abort,restart) \mid QQ(K,L,abort,restart,rtag,end,t2enabled))$$

$$\stackrel{\cdot}{\sim} \tau^2.(\nu\ K\ L\ abort\ restart)$$
$$(\overline{abort}.P(K,L,abort,restart)$$
$$\mid (QQ_1(K,L,abort,restart,rtag,end,t2enabled) + QQ_2(K,L,abort,restart,end)$$
$$+ QQ_3(K,L,abort,restart,t2enabled)))$$

$$\stackrel{\cdot}{\sim} \tau^2.(\nu\ K\ L\ abort\ restart)$$
$$(\overline{abort}.P(K,L,abort,restart)$$
$$\mid (K(first\ last\ tag\ m).\tau.(QQ_{11}(K,L,abort,restart,rtag,end,t2enabled)$$
$$+ QQ_{12}(K,L,abort,restart,first,last,tag,m,rtag,end,t2enabled))$$
$$+ \tau^2.abort.\tau.(Q(K,L,abort,restart) + QQ_{22}(K,L,abort,restart))$$
$$+ restart.\tau^2.Q(K,L,abort,restart)))$$

$$\stackrel{\cdot}{\sim} \tau^2.\tau^2.\tau.(\nu\ K\ L\ abort\ restart)$$
$$(P(K,L,abort,restart) \mid \tau.(Q(K,L,abort,restart) + QQ_{22}(K,L,abort,restart)))$$

$$\stackrel{\cdot}{\sim} \tau^5.(\nu\ K\ L\ abort\ restart)$$
$$(P(K,L,abort,restart)$$
$$\mid (\tau.Q(K,L,abort,restart)$$
$$+ \tau.\overline{Ind\_err}.Q(K,L,abort,restart)))$$

Il s'agit du dernier message du fichier. Mais comme ce message a déjà été délivré, le récepteur n'envoie pas d'indication d'erreur au consommateur.

$$\stackrel{\cdot}{\sim} \tau^5.\tau.(\nu\ K\ L\ abort\ restart)$$
$$(P(K,L,abort,restart) \mid Q(K,L,abort,restart))$$

$$B_3 \stackrel{\cdot}{\sim} \tau^6.System$$

Considérons la partie BB_3 (un message intermédiaire vient d'être délivré car jamais parvenu auparavant chez le récepteur mais l'acquittement correspondant est perdu, l'émetteur interrompt le transfert; le récepteur envoie une indication d'erreur au consommateur):

$$BB_3 \stackrel{def}{=} (\nu\ K\ L\ abort\ restart)$$
$$(\overline{abort}.P(K,L,abort,restart)$$
$$\mid \tau.[\![True]\!]t2enabled.QQ(K,L,abort,restart,rtag,end,t2enabled))$$

$$BB_3 \stackrel{\cdot}{\sim} \tau.(\nu\ K\ L\ abort\ restart)$$
$$(\overline{abort}.P(K,L,abort,restart) \mid [\![True]\!]t2enabled.QQ(K,L,abort,restart,rtag,end,t2enabled))$$

$$\stackrel{\cdot}{\sim}\ \tau.[\![True]\!]t2enabled.(\nu\ \ K\ \ L\ \ abort\ \ restart)$$
$$\qquad(\overline{abort}.P(K,L,abort,restart)\mid QQ(K,L,abort,restart,rtag,end,t2enabled))$$

$$\stackrel{\cdot}{\sim}\ \tau^2.(\nu\ \ K\ \ L\ \ abort\ \ restart)$$
$$\qquad(\overline{abort}.P(K,L,abort,restart)$$
$$\qquad\mid(QQ_1(K,L,abort,restart,rtag,end,t2enabled)\ +\ QQ_2(K,L,abort,restart,end)$$
$$\qquad\qquad+\ QQ_3(K,L,abort,restart,t2enabled)))$$

$$\stackrel{\cdot}{\sim}\ \tau^2.(\nu\ \ K\ \ L\ \ abort\ \ restart)$$
$$\qquad(\overline{abort}.P(K,L,abort,restart)$$
$$\qquad\mid(K(first\ \ last\ \ tag\ \ m).\tau.(QQ_{11}(K,L,abort,restart,rtag,end,t2enabled)$$
$$\qquad\qquad+\ QQ_{12}(K,L,abort,restart,first,last,tag,m,rtag,end,t2enabled))$$
$$\qquad+\ \tau^2.abort.\tau.(Q(K,L,abort,restart)\ +\ QQ_{22}(K,L,abort,restart))$$
$$\qquad+\ restart.\tau^2.Q(K,L,abort,restart)))$$

$$\stackrel{\cdot}{\sim}\ \tau^2.\tau^2.\tau.(\nu\ \ K\ \ L\ \ abort\ \ restart)$$
$$\qquad(P(K,L,abort,restart)\mid\tau.(Q(K,L,abort,restart)\ +\ QQ_{22}(K,L,abort,restart)))$$

$$\stackrel{\cdot}{\sim}\ \tau^5.(\nu\ \ K\ \ L\ \ abort\ \ restart)$$
$$\qquad(P(K,L,abort,restart)$$
$$\qquad\mid(\tau.Q(K,L,abort,restart)$$
$$\qquad+\ \tau.\overline{Ind\_err}.Q(K,L,abort,restart)))$$

Comme il s'agit d'un message intermédiaire du fichier, le récepteur envoie une indication d'erreur au consommateur.

$$\stackrel{\cdot}{\sim}\ \tau^5.\tau.\overline{Ind\_err}.(\nu\ \ K\ \ L\ \ abort\ \ restart)$$
$$\qquad(P(K,L,abort,restart)\mid Q(K,L,abort,restart))$$

$$BB_3\ \stackrel{\cdot}{\sim}\ \tau^6.\overline{Ind\_err}.System$$

Considérons la partie $B_4(hd,tl)$ (le message vient d'être délivré avec une indication $LAST$ ou $INCOMPLETE$ car jamais parvenu auparavant chez le récepteur mais l'acquittement correspondant est perdu, l'émetteur retransmet le message):

$$B_4(hd,tl)\ \stackrel{def}{=}\ \tau.(\nu\ \ K\ \ L\ \ abort\ \ restart)$$
$$\qquad(P_2(K,L,restart,cons(hd,tl),first,last,tag,rn)$$
$$\qquad\mid\tau.[\![True]\!]t2enabled.QQ(K,L,abort,restart,rtag,end,t2enabled))$$

$$B_4(hd,tl)\ \stackrel{\cdot}{\sim}\ \tau.\tau.(\nu\ \ K\ \ L\ \ abort\ \ restart)$$
$$\qquad(P_2(K,L,restart,cons(hd,tl),first,last,tag,rn)$$
$$\qquad\mid[\![True]\!]t2enabled.QQ(K,L,abort,restart,rtag,end,t2enabled))$$

$$\stackrel{\cdot}{\sim}\ \tau^2.[\![True]\!]t2enabled.(\nu\ \ K\ \ L\ \ abort\ \ restart)$$
$$\qquad(P_2(K,L,restart,cons(hd,tl),first,last,tag,rn)$$
$$\qquad\mid QQ(K,L,abort,restart,rtag,end,t2enabled))$$

$$\dot{\sim}\ \tau^3.(\nu\ \ K\ \ L\ \ abort\ \ restart)$$
$$(\tau^3.\overline{K}\ \ first\ \ last\ \ tag\ \ hd.\tau.add(rn,1,rn).$$
$$(P_{21}(K,L,abort,restart,tl,first,last,tag,rn)$$
$$+\ \ P_{22}(K,L,abort,restart,cons(hd,tl),first,last,tag,rn))$$
$$\mid (QQ_1(K,L,abort,restart,rtag,end,t2enabled)\ +\ QQ_2(K,L,abort,restart,end)$$
$$+\ \ QQ_3(K,L,abort,restart,t2enabled)))$$

$$\dot{\sim}\ \tau^3.\tau^3.(\nu\ \ K\ \ L\ \ abort\ \ restart)$$
$$(\overline{K}\ \ first\ \ last\ \ tag\ \ hd.\tau.add(rn,1,rn).$$
$$(P_{21}(K,L,abort,restart,tl,first,last,tag,rn)$$
$$+\ \ P_{22}(K,L,abort,restart,cons(hd,tl),first,last,tag,rn))$$
$$\mid (K(first\ \ last\ \ tag\ \ m).\tau.(QQ_{11}(K,L,abort,restart,rtag,end,t2enabled)$$
$$+\ \ QQ_{12}(K,L,abort,restart,first,last,tag,m,rtag,end,t2enabled))$$
$$+\ \ \tau^2.abort.\tau.(Q(K,L,abort,restart)\ +\ QQ_{22}(K,L,abort,restart))$$
$$+\ \ restart.\tau^2.Q(K,L,abort,restart)))$$

Deux situations sont possibles:

- Le message déjà délivré et retransmis est reçu par le récepteur, le récepteur envoie un autre acquittement à l'émetteur ($B_{21}(hd,tl)$).
- Le message déjà délivré et retransmis est perdu car l'événement *perte_msg* se produit ($A_{3191}(hd,tl)$).

$$\dot{\sim}\ \tau^6.\tau.(\nu\ \ K\ \ L\ \ abort\ \ restart)$$
$$(\tau.add(rn,1,rn).(P_{21}(K,L,abort,restart,tl,first,last,tag,rn)$$
$$+\ \ P_{22}(K,L,abort,restart,cons(hd,tl),first,last,tag,rn))$$
$$\mid \tau.(QQ_{11}(K,L,abort,restart,rtag,end,t2enabled)$$
$$+\ \ QQ_{12}(K,L,abort,restart,first,last,tag,hd,rtag,end,t2enabled)))$$
$$+\ \ \tau^6.\tau.(\nu\ \ K\ \ L\ \ abort\ \ restart)$$
$$(\tau.add(rn,1,rn).(P_{21}(K,L,abort,restart,tl,first,last,tag,rn)$$
$$+\ \ P_{22}(K,L,abort,restart,cons(hd,tl),first,last,tag,rn))$$
$$\mid (K(first\ \ last\ \ tag\ \ m).\tau.(QQ_{11}(K,L,abort,restart,rtag,end,t2enabled)$$
$$+\ \ QQ_{12}(K,L,abort,restart,first,last,tag,m,rtag,end,t2enabled))$$
$$+\ \ \tau^2.abort.\tau.(Q(K,L,abort,restart)\ +\ QQ_{22}(K,L,abort,restart))$$
$$+\ \ restart.\tau^2.Q(K,L,abort,restart)))$$

$$\dot{\sim}\ \tau^7.\tau.\tau.add(rn,1,rn).(\nu\ \ K\ \ L\ \ abort\ \ restart)$$
$$((P_{21}(K,L,abort,restart,tl,first,last,tag,rn)$$
$$+\ \ P_{22}(K,L,abort,restart,cons(hd,tl),first,last,tag,rn))$$
$$\mid (QQ_{11}(K,L,abort,restart,rtag,end,t2enabled)$$
$$+\ \ QQ_{12}(K,L,abort,restart,first,last,tag,hd,rtag,end,t2enabled)))$$
$$+\ \ \tau^3.\tau^4.(\nu\ \ K\ \ L\ \ abort\ \ restart)$$
$$(\tau.add(rn,1,rn).(P_{21}(K,L,abort,restart,tl,first,last,tag,rn)$$
$$+\ \ P_{22}(K,L,abort,restart,cons(hd,tl),first,last,tag,rn))$$
$$\mid (K(first\ \ last\ \ tag\ \ m).\tau.(QQ_{11}(K,L,abort,restart,rtag,end,t2enabled)$$

$$
\begin{array}{l}
\quad\quad + \; QQ_{12}(K, L, abort, restart, first, last, tag, m, rtag, end, t2enabled)) \\
\quad + \; \tau^2.abort.\tau.(Q(K, L, abort, restart) \; + \; QQ_{22}(K, L, abort, restart)) \\
\quad + \; restart.\tau^2.Q(K, L, abort, restart)))
\end{array}
$$

$$
\begin{array}{l}
\dot{\sim} \; \tau^8.\tau^2.(\nu \;\; K \;\; L \;\; abort \;\; restart) \\
\quad ((P_{21}(K, L, abort, restart, tl, first, last, tag, rn) \\
\quad + \; P_{22}(K, L, abort, restart, cons(hd, tl), first, last, tag, rn)) \\
\quad \mid (QQ_{11}(K, L, abort, restart, rtag, end, t2enabled) \\
\quad + \; QQ_{12}(K, L, abort, restart, first, last, tag, hd, rtag, end, t2enabled))) \\
\; + \; \tau^3.\tau^4.(\nu \;\; K \;\; L \;\; abort \;\; restart) \\
\quad (\tau.add(rn, 1, rn).(P_{21}(K, L, abort, restart, tl, first, last, tag, rn) \\
\quad\quad\quad + \; P_{22}(K, L, abort, restart, cons(hd, tl), first, last, tag, rn)) \\
\quad \mid (K(first \;\; last \;\; tag \;\; m).\tau.(QQ_{11}(K, L, abort, restart, rtag, end, t2enabled) \\
\quad\quad + \; QQ_{12}(K, L, abort, restart, first, last, tag, m, rtag, end, t2enabled)) \\
\quad + \; \tau^2.abort.\tau.(Q(K, L, abort, restart) \; + \; QQ_{22}(K, L, abort, restart)) \\
\quad + \; restart.\tau^2.Q(K, L, abort, restart)))
\end{array}
$$

$$
B_4(hd, tl) \; \dot{\sim} \; \tau^8.B_{21}(hd, tl) \; + \; \tau^3.A_{3191}(hd, tl)
$$

Considérons la partie $A_{3191}(hd, tl)$ (le message déjà délivré est retransmis, mais il est perdu car l'événement *perte_msg* se produit):

$$
\begin{array}{l}
A_{3191}(hd,tl) \stackrel{def}{=} \tau^4.(\nu \;\; K \;\; L \;\; abort \;\; restart) \\
\quad (\tau.add(rn, 1, rn).(P_{21}(K, L, abort, restart, tl, first, last, tag, rn) \\
\quad\quad + \; P_{22}(K, L, abort, restart, cons(hd, tl), first, last, tag, rn)) \\
\quad \mid (K(first \;\; last \;\; tag \;\; m).\tau.(QQ_{11}(K, L, abort, restart, rtag, end, t2enabled) \\
\quad\quad + \; QQ_{12}(K, L, abort, restart, first, last, tag, m, rtag, end, t2enabled)) \\
\quad + \; \tau^2.abort.\tau.(Q(K, L, abort, restart) \; + \; QQ_{22}(K, L, abort, restart)) \\
\quad + \; restart.\tau^2.Q(K, L, abort, restart)))
\end{array}
$$

$$
\begin{array}{l}
A_{3191}(hd,tl) \; \dot{\sim} \; \tau^4.\tau.add(rn, 1, rn).(\nu \;\; K \;\; L \;\; abort \;\; restart) \\
\quad ((P_{21}(K, L, abort, restart, tl, first, last, tag, rn) \\
\quad\quad + \; P_{22}(K, L, abort, restart, cons(hd, tl), first, last, tag, rn)) \\
\quad \mid (K(first \;\; last \;\; tag \;\; m).\tau.(QQ_{11}(K, L, abort, restart, rtag, end, t2enabled) \\
\quad\quad + \; QQ_{12}(K, L, abort, restart, first, last, tag, m, rtag, end, t2enabled)) \\
\quad + \; \tau^2.abort.\tau.(Q(K, L, abort, restart) \; + \; QQ_{22}(K, L, abort, restart)) \\
\quad + \; restart.\tau^2.Q(K, L, abort, restart)))
\end{array}
$$

$$
\begin{array}{l}
\dot{\sim} \; \tau^6.(\nu \;\; K \;\; L \;\; abort \;\; restart) \\
\quad ((L.\tau.[\![0]\!]rn.Not(tag).[\![False]\!]first.PP(K, L, abort, restart, tl, first, last, tag, rn) \\
\quad + \; \tau^2.(P_{221}(K, L, abort, restart, last) \\
\quad\quad\quad + \; P_{222}(K, L, abort, restart, cons(hd, tl), first, last, tag, rn))) \\
\quad \mid (K(first \;\; last \;\; tag \;\; m).\tau.(QQ_{11}(K, L, abort, restart, rtag, end, t2enabled) \\
\quad\quad + \; QQ_{12}(K, L, abort, restart, first, last, tag, m, rtag, end, t2enabled)) \\
\quad + \; \tau^2.abort.\tau.(Q(K, L, abort, restart) \; + \; QQ_{22}(K, L, abort, restart)) \\
\quad + \; restart.\tau^2.Q(K, L, abort, restart)))
\end{array}
$$

Sachant que le message est perdu, l'émetteur ne peut recevoir d'acquittement.

$$
\begin{aligned}
\dot{\sim}\ & \tau^6.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\quad (\tau^2.(P_{221}(K, L, abort, restart, last)\\
&\qquad\quad +\ P_{222}(K, L, abort, restart, cons(hd, tl), first, last, tag, rn))\\
&\quad \mid (K(first\ \ last\ \ tag\ \ m).\tau.(QQ_{11}(K, L, abort, restart, rtag, end, t2enabled)\\
&\qquad\quad +\ QQ_{12}(K, L, abort, restart, first, last, tag, m, rtag, end, t2enabled))\\
&\quad +\ \tau^2.abort.\tau.(Q(K, L, abort, restart)\ +\ QQ_{22}(K, L, abort, restart))\\
&\quad +\ restart.\tau^2.Q(K, L, abort, restart)))
\end{aligned}
$$

$$
\begin{aligned}
\dot{\sim}\ & \tau^8.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\quad ((P_{2211}(K, L, abort, restart, last)\ +\ P_{2212}(K, L, abort, restart, last)\\
&\qquad\quad +\ P_2(K, L, abort, restart, cons(hd, tl), first, last, tag, rn))\\
&\quad \mid (K(first\ \ last\ \ tag\ \ m).\tau.(QQ_{11}(K, L, abort, restart, rtag, end, t2enabled)\\
&\qquad\quad +\ QQ_{12}(K, L, abort, restart, first, last, tag, m, rtag, end, t2enabled))\\
&\quad +\ \tau^2.abort.\tau.(Q(K, L, abort, restart)\ +\ QQ_{22}(K, L, abort, restart))\\
&\quad +\ restart.\tau^2.Q(K, L, abort, restart)))
\end{aligned}
$$

$$
\begin{aligned}
\dot{\sim}\ & \tau^8.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\quad ((\tau.\overline{Conf}\ \ DONTKNOW.\overline{abort}.P(K, L, abort, restart)\\
&\quad +\ \tau.\overline{Conf}\ \ NOTOK.\overline{abort}.P(K, L, abort, restart)\\
&\quad +\ \tau^3.\overline{K}\ \ first\ \ last\ \ tag\ \ hd.\tau.add(rn, 1, rn).\\
&\qquad\quad (P_{21}(K, L, abort, restart, tl, first, last, tag, rn)\\
&\qquad\quad +\ P_{22}(K, L, abort, restart, cons(hd, tl), first, last, tag, rn)))\\
&\quad \mid (K(first\ \ last\ \ tag\ \ m).\tau.(QQ_{11}(K, L, abort, restart, rtag, end, t2enabled)\\
&\qquad\quad +\ QQ_{12}(K, L, abort, restart, first, last, tag, m, rtag, end, t2enabled))\\
&\quad +\ \tau^2.abort.\tau.(Q(K, L, abort, restart)\ +\ QQ_{22}(K, L, abort, restart))\\
&\quad +\ restart.\tau^2.Q(K, L, abort, restart)))
\end{aligned}
$$

Quatre situations sont possibles:

- L'émetteur interrompt le transfert et envoie une confirmation $DONTKNOW$ au producteur car il s'agit du dernier message du fichier; le récepteur reçoit le signal *abort* et se réinitialise (A_{322}).

- L'émetteur interrompt le transfert et envoie une confirmation $NOTOK$ au producteur car il s'agit d'un message intermédiaire du fichier; le récepteur reçoit le signal *abort* et envoie une indication d'erreur au consommateur (A_{327}).

- L'émetteur retransmet le message, le récepteur le reçoit ($A_{3151}(hd, tl)$).

- L'émetteur retransmet le message mais ce message est encore perdu car l'événement *perte_msg* se produit ($A_{3191}(hd, tl)$).

$$
\begin{aligned}
\dot{\sim}\ & \tau^8.(\tau^2.\tau.\overline{Conf}\ \ DONTKNOW.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\quad (\overline{abort}.P(K, L, abort, restart) \mid abort.\tau.(Q(K, L, abort, restart)\ +\ QQ_{22}(K, L, abort, restart)))
\end{aligned}
$$

$$
\begin{aligned}
&+\ \tau^2.\tau.\overline{Conf}\ \ NOTOK.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\quad(\overline{abort}.P(K,L,abort,restart) \mid abort.\tau.(Q(K,L,abort,restart)\ +\ QQ_{22}(K,L,abort,restart)))\\
&+\ \tau^3.\tau.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\quad(\tau.add(rn,1,rn).(P_{21}(K,L,abort,restart,tl,first,last,tag,rn)\\
&\qquad\qquad+\ P_{22}(K,L,abort,restart,cons(hd,tl),first,last,tag,rn))\\
&\quad\mid \tau.(QQ_{11}(K,L,abort,restart,rtag,end,t2enabled)\\
&\qquad+\ QQ_{12}(K,L,abort,restart,first,last,tag,hd,rtag,end,t2enabled)))\\
&+\ \tau^4.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\quad(\tau.add(rn,1,rn).(P_{21}(K,L,abort,restart,tl,first,last,tag,rn)\\
&\qquad\qquad+\ P_{22}(K,L,abort,restart,cons(hd,tl),first,last,tag,rn))\\
&\quad\mid (K(first\ \ last\ \ tag\ \ m).\tau.(QQ_{11}(K,L,abort,restart,rtag,end,t2enabled)\\
&\qquad+\ QQ_{12}(K,L,abort,restart,first,last,tag,m,rtag,end,t2enabled))\\
&\quad+\ \tau^2.abort.\tau.(Q(K,L,abort,restart)\ +\ QQ_{22}(K,L,abort,restart))\\
&\quad+\ restart.\tau^2.Q(K,L,abort,restart))))
\end{aligned}
$$

$$
\begin{aligned}
A_{3191}(hd,tl)\ \dot{\sim}\ \tau^8.(&\tau^3.\overline{Conf}\ \ DONTKNOW.A_{322}\\
&+\ \tau^3.\overline{Conf}\ \ NOTOK.A_{327}\\
&+\ A_{3151}(hd,tl)\\
&+\ A_{3191}(hd,tl))
\end{aligned}
$$

Considérons la partie $A_{3151}(hd,tl)$ (le message déjà délivré a été retransmis par l'émetteur et reçu par le récepteur):

$$
\begin{aligned}
A_{3151}(hd,tl)\ \stackrel{def}{=}\ &\tau^3.\tau.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&(\tau.add(rn,1,rn).(P_{21}(K,L,abort,restart,tl,first,last,tag,rn)\\
&\qquad+\ P_{22}(K,L,abort,restart,cons(hd,tl),first,last,tag,rn))\\
&\mid \tau.(QQ_{11}(K,L,abort,restart,rtag,end,t2enabled)\\
&\quad+\ QQ_{12}(K,L,abort,restart,first,last,tag,hd,rtag,end,t2enabled)))
\end{aligned}
$$

$$
\begin{aligned}
A_{3151}(hd,tl)\ \dot{\sim}\ &\tau^4.\tau.\tau.add(rn,1,rn).(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&((P_{21}(K,L,abort,restart,tl,first,last,tag,rn)\\
&+\ P_{22}(K,L,abort,restart,cons(hd,tl),first,last,tag,rn))\\
&\mid (QQ_{11}(K,L,abort,restart,rtag,end,t2enabled)\\
&+\ QQ_{12}(K,L,abort,restart,first,last,tag,hd,rtag,end,t2enabled)))
\end{aligned}
$$

$$
\begin{aligned}
\dot{\sim}\ \tau^5.\tau^2.(\nu\ \ K\ \ L\ \ abort\ \ restart)&\\
&((P_{21}(K,L,abort,restart,tl,first,last,tag,rn)\\
&+\ P_{22}(K,L,abort,restart,cons(hd,tl),first,last,tag,rn))\\
&\mid (QQ_{11}(K,L,abort,restart,rtag,end,t2enabled)\\
&+\ QQ_{12}(K,L,abort,restart,first,last,tag,hd,rtag,end,t2enabled)))
\end{aligned}
$$

$$A_{3151}(hd,tl)\ \dot{\sim}\ \tau^5.B_{21}(hd,tl)$$

Considérons la partie $B_{21}(hd,tl)$ (le message déjà délivré et retransmis par l'émetteur est reçu par le récepteur, celui-ci envoie un autre acquittement à l'émetteur):

$$B_{21}(hd,tl)\ \stackrel{def}{=}\ \tau^2.(\nu\ \ K\ \ L\ \ abort\ \ restart)$$

$$((P_{21}(K, L, abort, restart, tl, first, last, tag, rn)$$
$$+\ P_{22}(K, L, abort, restart, cons(hd, tl), first, last, tag, rn))$$
$$\mid (QQ_{11}(K, L, abort, restart, rtag, end, t2enabled)$$
$$+\ QQ_{12}(K, L, abort, restart, first, last, tag, hd, rtag, end, t2enabled)))$$

Sachant que le message est déjà délivré, le récepteur n'envoie pas d'indication pour ce message.

$$B_{21}(hd, tl) \stackrel{\cdot}{\sim} \tau^2.(\nu\ \ K\ \ L\ \ abort\ \ restart)$$
$$((P_{21}(K, L, abort, restart, tl, first, last, tag, rn)$$
$$+\ P_{22}(K, L, abort, restart, cons(hd, tl), first, last, tag, rn))$$
$$\mid QQ_{11}(K, L, abort, restart, rtag, end, t2enabled))$$

$$\stackrel{\cdot}{\sim}\ \tau^2.(\nu\ \ K\ \ L\ \ abort\ \ restart)$$
$$((L.\tau.[\![0]\!]rn.Not(tag).[\![False]\!]first.PP(K, L, abort, restart, tl, first, last, tag, rn)$$
$$+\ \tau^2.(P_{221}(K, L, abort, restart, last)$$
$$+\ P_{222}(K, L, abort, restart, cons(hd, tl), first, last, tag, rn)))$$
$$\mid \overline{L}.QQ(K, L, abort, restart, rtag, end, t2enabled))$$

$$\stackrel{\cdot}{\sim}\ \tau^2.(\nu\ \ K\ \ L\ \ abort\ \ restart)$$
$$((L.\tau.[\![0]\!]rn.Not(tag).[\![False]\!]first.PP(K, L, abort, restart, tl, first, last, tag, rn)$$
$$+\ \tau^2.(P_{2211}(K, L, abort, restart, last)\ +\ P_{2212}(K, L, abort, restart, last))$$
$$+\ \tau^2.P_2(K, L, abort, restart, cons(hd, tl), first, last, tag, rn))$$
$$\mid \overline{L}.QQ(K, L, abort, restart, rtag, end, t2enabled))$$

$$\stackrel{\cdot}{\sim}\ \tau^2.(\nu\ \ K\ \ L\ \ abort\ \ restart)$$
$$((L.\tau.[\![0]\!]rn.Not(tag).[\![False]\!]first.PP(K, L, abort, restart, tl, first, last, tag, rn)$$
$$+\ \tau^2.\tau.\overline{Conf}\ \ DONTKNOW.\overline{abort}.P(K, L, abort, restart)$$
$$+\ \tau^2.\tau.\overline{Conf}\ \ NOTOK.\overline{abort}.P(K, L, abort, restart)$$
$$+\ \tau^2.P_2(K, L, abort, restart, cons(hd, tl), first, last, tag, rn))$$
$$\mid \overline{L}.QQ(K, L, abort, restart, rtag, end, t2enabled))$$

Quatre situations sont possibles:

- L'acquittement est reçu par l'émetteur ($A_{31}(tl)$).
- L'acquittement est perdu, l'émetteur interrompt le transfert avec une confirmation $DONTKNOW$ (B_1).
- L'acquittement est perdu, l'émetteur interrompt le transfert avec une confirmation $NOTOK$; le récepteur envoie une indication d'erreur au consommateur (BB_1).
- L'acquittement est perdu, l'émetteur retransmet à nouveau le message ($B_2(hd, tl)$).

$$
\begin{aligned}
\dot\sim\ & \tau^2.\tau.\tau.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\quad ([\![0]\!]rn.Not(tag).[\![False]\!]first.PP(K,L,abort,restart,tl,first,last,tag,rn)\\
&\quad \mid QQ(K,L,abort,restart,rtag,end,t2enabled))\\
&+\ \tau^2.\tau^2.\tau.\tau.\overline{Conf}\ \ DONTKNOW.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\quad (\overline{abort}.P(K,L,abort,restart) \mid QQ(K,L,abort,restart,rtag,end,t2enabled))\\
&+\ \tau^2.\tau^2.\tau.\tau.\overline{Conf}\ \ NOTOK.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\quad (\overline{abort}.P(K,L,abort,restart) \mid QQ(K,L,abort,restart,rtag,end,t2enabled))\\
&+\ \tau^2.\tau^2.\tau.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\quad (P_2(K,L,abort,restart,cons(hd,tl),first,last,tag,rn)\\
&\quad \mid QQ(K,L,abort,restart,rtag,end,t2enabled))
\end{aligned}
$$

$$
\begin{aligned}
B_{21}(hd,tl)\ \dot\sim\ & A_{31}(tl)\\
&+\ \tau^6.\overline{Conf}\ \ DONTKNOW.B_1\\
&+\ \tau^6.\overline{Conf}\ \ NOTOK.BB_1\\
&+\ B_2(hd,tl)
\end{aligned}
$$

Considérons la partie B_1 (le dernier message du fichier a déjà été délivré mais l'acquittement correspondant est perdu; l'émetteur interrompt le transfert):

$$
\begin{aligned}
B_1 \stackrel{def}{=}\ & (\nu\ \ K\ \ L\ \ abort\ \ restart)\\
& (\overline{abort}.P(K,L,abort,restart) \mid QQ(K,L,abort,restart,rtag,end,t2enabled))
\end{aligned}
$$

$$
\begin{aligned}
B_1\ \dot\sim\ & (\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\quad (\overline{abort}.P(K,L,abort,restart)\\
&\quad \mid (QQ_1(K,L,abort,restart,rtag,end,t2enabled)\ +\ QQ_2(K,L,abort,restart,end)\\
&\qquad +\ QQ_3(K,L,abort,restart,t2enabled)))
\end{aligned}
$$

$$
\begin{aligned}
\dot\sim\ & (\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\quad (\overline{abort}.P(K,L,abort,restart)\\
&\quad \mid (K(first\ \ last\ \ tag\ \ m).\tau.(QQ_{11}(K,L,abort,restart,rtag,end,t2enabled)\\
&\qquad +\ QQ_{12}(K,L,abort,restart,first,last,tag,m,rtag,end,t2enabled))\\
&\quad +\ \tau^2.abort.\tau.(Q(K,L,abort,restart)\ +\ QQ_{22}(K,L,abort,restart))\\
&\quad +\ restart.\tau^2.Q(K,L,abort,restart)))
\end{aligned}
$$

$$
\begin{aligned}
\dot\sim\ & \tau^2.\tau.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\quad (P(K,L,abort,restart) \mid \tau.(Q(K,L,abort,restart)\ +\ QQ_{22}(K,L,abort,restart)))
\end{aligned}
$$

$$
\begin{aligned}
\dot\sim\ & \tau^3.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\quad (P(K,L,abort,restart)\\
&\quad \mid (\tau.Q(K,L,abort,restart)\\
&\quad +\ \tau.\overline{Ind\_err}.Q(K,L,abort,restart)))
\end{aligned}
$$

Or, comme l'abort a eu lieu lors du transfert du dernier message du fichier et que ce message a déjà été délivré, aucune indication d'erreur n'est envoyée par le récepteur au consommateur.

$$
\begin{aligned}
\dot\sim\ & \tau^3.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\quad (P(K,L,abort,restart) \mid \tau.Q(K,L,abort,restart))
\end{aligned}
$$

$$\dot{\sim}\ \tau^3.\tau.(\nu\ \ K\ \ L\ \ abort\ \ restart)$$
$$(P(K, L, abort, restart) \mid Q(K, , abort, restart))$$

$$B_1\ \dot{\sim}\ \tau^4.System$$

Considérons la partie BB_1 (l'acquittement d'un message intermédiaire du fichier, ayant déjà été délivré; est perdu, l'émetteur interrompt le transfert et le récepteur envoie une indication d'erreur au consommateur):

$$BB_1 \stackrel{def}{=} (\nu\ \ K\ \ L\ \ abort\ \ restart)$$
$$(\overline{abort}.P(K, L, abort, restart) \mid QQ(K, L, abort, restart, rtag, end, t2enabled))$$

$$BB_1\ \dot{\sim}\ (\nu\ \ K\ \ L\ \ abort\ \ restart)$$
$$(\overline{abort}.P(K, L, abort, restart)$$
$$\mid (QQ_1(K, L, abort, restart, rtag, end, t2enabled)\ +\ QQ_2(K, L, abort, restart, end)$$
$$+\ QQ_3(K, L, abort, restart, t2enabled)))$$

$$\dot{\sim}\ (\nu\ \ K\ \ L\ \ abort\ \ restart)$$
$$(\overline{abort}.P(K, L, abort, restart)$$
$$\mid (K(first\ \ last\ \ tag\ \ m).\tau.(QQ_{11}(K, L, abort, restart, rtag, end, t2enabled)$$
$$+\ QQ_{12}(K, L, abort, restart, first, last, tag, m, rtag, end, t2enabled))$$
$$+\ \tau^2.abort.\tau.(Q(K, L, abort, restart)\ +\ QQ_{22}(K, L, abort, restart))$$
$$+\ restart.\tau^2.Q(K, L, abort, restart)))$$

$$\dot{\sim}\ \tau^2.\tau.(\nu\ \ K\ \ L\ \ abort\ \ restart)$$
$$(P(K, L, abort, restart) \mid \tau.(Q(K, L, abort, restart)\ +\ QQ_{22}(K, L, abort, restart)))$$

$$\dot{\sim}\ \tau^3.(\nu\ \ K\ \ L\ \ abort\ \ restart)$$
$$(P(K, L, abort, restart)$$
$$\mid (\tau.Q(K, L, abort, restart)$$
$$+\ \tau.\overline{Ind\_err}.Q(K, L, abort, restart)))$$

Comme il s'agit d'un message intermédiaire du fichier, une indication d'erreur est envoyée par le récepteur au consommateur.

$$\dot{\sim}\ \tau^3.(\nu\ \ K\ \ L\ \ abort\ \ restart)$$
$$(P(K, L, abort, restart)$$
$$\mid \tau.\overline{Ind\_err}.Q(K, L, abort, restart))$$

$$\dot{\sim}\ \tau^3.\tau.\overline{Ind\_err}.(\nu\ \ K\ \ L\ \ abort\ \ restart)$$
$$(P(K, L, abort, restart) \mid Q(K, L, abort, restart))$$

$$BB_1\ \dot{\sim}\ \tau^4.\overline{Ind\_err}.System$$

Considérons la partie $B_2(hd,tl)$ (l'acquittement du message déjà délivré est perdu, l'émetteur retransmet le message):

$$\begin{array}{l}
B_2(hd,tl) \stackrel{def}{=} \tau^5.(\nu\ K\ L\ abort\ restart) \\
\qquad (P_2(K,L,restart,cons(hd,tl),first,last,tag,rn) \\
\qquad \mid QQ(K,L,abort,restart,rtag,end,t2enabled))
\end{array}$$

$$\begin{array}{l}
B_2(hd,tl)\ \dot{\sim}\ \tau^5.(\nu\ K\ L\ abort\ restart) \\
\qquad (\tau^3.\overline{K}\ first\ last\ tag\ hd.\tau.add(rn,1,rn). \\
\qquad\quad (P_{21}(K,L,abort,restart,tl,first,last,tag,rn) \\
\qquad\quad +\ P_{22}(K,L,abort,restart,cons(hd,tl),first,last,tag,rn)) \\
\qquad \mid QQ(K,L,abort,restart,rtag,end,t2enabled))
\end{array}$$

$$\begin{array}{l}
\dot{\sim}\ \tau^8.(\nu\ K\ L\ abort\ restart) \\
\qquad (\overline{K}\ first\ last\ tag\ hd.\tau.add(rn,1,rn). \\
\qquad\quad (P_{21}(K,L,abort,restart,tl,first,last,tag,rn) \\
\qquad\quad +\ P_{22}(K,L,abort,restart,cons(hd,tl),first,last,tag,rn)) \\
\qquad \mid QQ(K,L,abort,restart,rtag,end,t2enabled))
\end{array}$$

$$\begin{array}{l}
\dot{\sim}\ \tau^8.(\nu\ K\ L\ abort\ restart) \\
\qquad (\overline{K}\ first\ last\ tag\ hd.\tau.add(rn,1,rn). \\
\qquad\quad (P_{21}(K,L,abort,restart,tl,first,last,tag,rn) \\
\qquad\quad +\ P_{22}(K,L,abort,restart,cons(hd,tl),first,last,tag,rn)) \\
\qquad \mid (QQ_1(K,L,abort,restart,rtag,end,t2enabled)\ +\ QQ_2(K,L,abort,restart,end) \\
\qquad\quad +\ QQ_3(K,L,abort,restart,t2enabled)))
\end{array}$$

$$\begin{array}{l}
\dot{\sim}\ \tau^8.(\nu\ K\ L\ abort\ restart) \\
\qquad (\overline{K}\ first\ last\ tag\ hd.\tau.add(rn,1,rn). \\
\qquad\quad (P_{21}(K,L,abort,restart,tl,first,last,tag,rn) \\
\qquad\quad +\ P_{22}(K,L,abort,restart,cons(hd,tl),first,last,tag,rn)) \\
\qquad \mid (K(first\ last\ tag\ m).\tau.(QQ_{11}(K,L,abort,restart,rtag,end,t2enabled) \\
\qquad\quad +\ QQ_{12}(K,L,abort,restart,first,last,tag,m,rtag,end,t2enabled)) \\
\qquad +\ \tau^2.abort.\tau.(Q(K,L,abort,restart)\ +\ QQ_{22}(K,L,abort,restart)) \\
\qquad +\ restart.\tau^2.Q(K,L,abort,restart)))
\end{array}$$

Deux situations sont possibles:

- le message déjà délivré au consommateur et retransmis par l'émetteur est reçu par le récepteur, ce dernier envoie alors un autre acquittement pour ce message et pas d'indication au consommateur ($B_{21}(hd,tl)$).
- le message déjà délivré au consommateur est perdu lors de la retransmission car l'événement *perte_msg* se produit ($A_{3191}(hd,tl)$).

$$
\begin{aligned}
\dot{\sim}\ & \tau^{8}.\tau.(\nu\ \ K\ \ L\ \ abort\ \ restart) \\
& \quad (\tau.add(rn,1,rn).(P_{21}(K,L,abort,restart,tl,first,last,tag,rn) \\
& \qquad\qquad + \ P_{22}(K,L,abort,restart,cons(hd,tl),first,last,tag,rn)) \\
& \quad \mid \tau.(QQ_{11}(K,L,abort,restart,rtag,end,t2enabled) \\
& \qquad + \ QQ_{12}(K,L,abort,restart,first,last,tag,hd,rtag,end,t2enabled))) \\
& + \ \tau^{8}.\tau.(\nu\ \ K\ \ L\ \ abort\ \ restart) \\
& \quad (\tau.add(rn,1,rn).(P_{21}(K,L,abort,restart,tl,first,last,tag,rn) \\
& \qquad\qquad + \ P_{22}(K,L,abort,restart,cons(hd,tl),first,last,tag,rn)) \\
& \quad \mid (K(first\ \ last\ \ tag\ \ m).\tau.(QQ_{11}(K,L,abort,restart,rtag,end,t2enabled) \\
& \qquad + \ QQ_{12}(K,L,abort,restart,first,last,tag,m,rtag,end,t2enabled)) \\
& \quad + \ \tau^{2}.abort.\tau.(Q(K,L,abort,restart) \ + \ QQ_{22}(K,L,abort,restart)) \\
& \quad + \ restart.\tau^{2}.Q(K,L,abort,restart)))
\end{aligned}
$$

$$
\begin{aligned}
\dot{\sim}\ & \tau^{9}.\tau.\tau.add(rn,1,rn).(\nu\ \ K\ \ L\ \ abort\ \ restart) \\
& \quad ((P_{21}(K,L,abort,restart,tl,first,last,tag,rn) \\
& \quad + \ P_{22}(K,L,abort,restart,cons(hd,tl),first,last,tag,rn)) \\
& \quad \mid (QQ_{11}(K,L,abort,restart,rtag,end,t2enabled) \\
& \quad + \ QQ_{12}(K,L,abort,restart,first,last,tag,hd,rtag,end,t2enabled))) \\
& + \ \tau^{5}.\tau^{4}.(\nu\ \ K\ \ L\ \ abort\ \ restart) \\
& \quad (\tau.add(rn,1,rn).(P_{21}(K,L,abort,restart,tl,first,last,tag,rn) \\
& \qquad\qquad + \ P_{22}(K,L,abort,restart,cons(hd,tl),first,last,tag,rn)) \\
& \quad \mid (K(first\ \ last\ \ tag\ \ m).\tau.(QQ_{11}(K,L,abort,restart,rtag,end,t2enabled) \\
& \qquad + \ QQ_{12}(K,L,abort,restart,first,last,tag,m,rtag,end,t2enabled)) \\
& \quad + \ \tau^{2}.abort.\tau.(Q(K,L,abort,restart) \ + \ QQ_{22}(K,L,abort,restart)) \\
& \quad + \ restart.\tau^{2}.Q(K,L,abort,restart)))
\end{aligned}
$$

$$
\begin{aligned}
\dot{\sim}\ & \tau^{10}.\tau^{2}.(\nu\ \ K\ \ L\ \ abort\ \ restart) \\
& \quad ((P_{21}(K,L,abort,restart,tl,first,last,tag,rn) \\
& \quad + \ P_{22}(K,L,abort,restart,cons(hd,tl),first,last,tag,rn)) \\
& \quad \mid (QQ_{11}(K,L,abort,restart,rtag,end,t2enabled) \\
& \quad + \ QQ_{12}(K,L,abort,restart,first,last,tag,hd,rtag,end,t2enabled))) \\
& + \ \tau^{5}.\tau^{4}.(\nu\ \ K\ \ L\ \ abort\ \ restart) \\
& \quad (\tau.add(rn,1,rn).(P_{21}(K,L,abort,restart,tl,first,last,tag,rn) \\
& \qquad\qquad + \ P_{22}(K,L,abort,restart,cons(hd,tl),first,last,tag,rn)) \\
& \quad \mid (K(first\ \ last\ \ tag\ \ m).\tau.(QQ_{11}(K,L,abort,restart,rtag,end,t2enabled) \\
& \qquad + \ QQ_{12}(K,L,abort,restart,first,last,tag,m,rtag,end,t2enabled)) \\
& \quad + \ \tau^{2}.abort.\tau.(Q(K,L,abort,restart) \ + \ QQ_{22}(K,L,abort,restart)) \\
& \quad + \ restart.\tau^{2}.Q(K,L,abort,restart)))
\end{aligned}
$$

$$B_2(hd,tl) \ \dot{\sim} \ \tau^{10}.B_{21}(hd,tl) \ + \ \tau^{5}.A_{3191}(hd,tl)$$

Considérons la partie $A_{32}(head,tail)$ (le premier message a été délivré par le récepteur, mais l'acquittement correspondant est perdu):

$$A_{32}(head,tail) \stackrel{def}{=} \tau.(\nu\ \ K\ \ L\ \ abort\ \ restart)$$

$$
\begin{aligned}
&((P_{221}(K, L, abort, restart, last)\\
&+\ P_{222}(K, L, abort, restart, cons(head, tail), first, last, tag, rn))\\
&\mid \tau.[\![True]\!]t2enabled.QQ(K, L, abort, restart, rtag, end, t2enabled))
\end{aligned}
$$

$$
\begin{aligned}
A_{32}(head, tail)\ \dot{\sim}\ & \tau.\tau.[\![True]\!]t2enabled.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&((P_{2211}(K, L, abort, restart, last)\ +\ P_{2212}(K, L, abort, restart, last)\\
&+\ P_2(K, L, abort, restart, cons(head, tail), first, last, tag, rn))\\
&\mid QQ(K, L, abort, restart, rtag, end, t2enabled))
\end{aligned}
$$

$$
\begin{aligned}
\dot{\sim}\ & \tau^3.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&((\tau.\overline{Conf}\ \ DONTKNOW.\overline{abort}.P(K, L, abort, restart)\\
&+\ \tau.\overline{Conf}\ \ NOTOK.\overline{abort}.P(K, L, abort, restart)\\
&+\ \tau^3.\overline{K}\ \ first\ \ last\ \ tag\ \ head.\tau.add(rn, 1, rn).\\
&\qquad (P_{21}(K, L, abort, restart, tail, first, last, tag, rn)\\
&\qquad +\ P_{22}(K, L, abort, restart, cons(head, tail), first, last, tag, rn)))\\
&\mid (QQ_1(K, L, abort, restart, rtag, end, t2enabled)\ +\ QQ_2(K, L, abort, restart, end)\\
&\qquad +\ QQ_3(K, L, abort, restart, t2enabled)))
\end{aligned}
$$

$$
\begin{aligned}
\dot{\sim}\ & \tau^3.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&((\tau.\overline{Conf}\ \ DONTKNOW.\overline{abort}.P(K, L, abort, restart)\\
&+\ \tau.\overline{Conf}\ \ NOTOK.\overline{abort}.P(K, L, abort, restart)\\
&+\ \tau^3.\overline{K}\ \ first\ \ last\ \ tag\ \ head.\tau.add(rn, 1, rn).\\
&\qquad (P_{21}(K, L, abort, restart, tail, first, last, tag, rn)\\
&\qquad +\ P_{22}(K, L, abort, restart, cons(head, tail), first, last, tag, rn)))\\
&\mid (K(first\ \ last\ \ tag\ \ m).\tau.(QQ_{11}(K, L, abort, restart, rtag, end, t2enabled)\\
&\qquad +\ QQ_{12}(K, L, abort, restart, first, last, tag, m, rtag, end, t2enabled))\\
&+\ \tau^2.abort.\tau.(Q(K, L, abort, restart)\ +\ QQ_{22}(K, L, abort, restart))\\
&+\ restart.\tau^2.Q(K, L, abort, restart)))
\end{aligned}
$$

Quatre situations sont possibles:

- L'émetteur interrompt le transfert avec une confirmation $DONTKNOW$ car le premier message est aussi le dernier message du fichier (A_{322}).
- L'émetteur interrompt le transfert avec une confirmation $NOTOK$; le récepteur envoie une indication d'erreur au consommateur (A_{327}).
- L'émetteur retransmet le premier message (déjà délivré) et il est reçu par le récepteur ($A_{324}(head, tail)$).
- L'émetteur retransmet le premier message (déjà délivré) mais il est perdu ($A_{3191}(head, tail)$).

$$
\begin{aligned}
\dot{\sim}\ & \tau^3.(\tau^2.\tau.\overline{Conf}\ \ DONTKNOW.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
&\quad (\overline{abort}.P(K, L, abort, restart)\\
&\quad \mid abort.\tau.(Q(K, L, abort, restart)\ +\ QQ_{22}(K, L, abort, restart)))\\
&+\ \tau^2.\tau.\overline{Conf}\ \ NOTOK.(\nu\ \ K\ \ L\ \ abort\ \ restart)
\end{aligned}
$$

$$
\begin{aligned}
&\quad(\overline{abort}.P(K,L,abort,restart)\\
&\quad\mid abort.\tau.(Q(K,L,abort,restart) + QQ_{22}(K,L,abort,restart)))\\
&+ \tau^3.\tau.(\nu\ K\ L\ abort\ restart)\\
&\quad(\tau.add(rn,1,rn).(P_{21}(K,L,abort,restart,tail,first,last,tag,rn)\\
&\qquad\qquad + P_{22}(K,L,abort,restart,cons(head,tail),first,last,tag,rn))\\
&\quad\mid \tau.(QQ_{11}(K,L,abort,restart,rtag,end,t2enabled)\\
&\qquad + QQ_{12}(K,L,abort,restart,first,last,tag,head,rtag,end,t2enabled)))\\
&+ \tau^3.\tau.(\nu\ K\ L\ abort\ restart)\\
&\quad(\tau.add(rn,1,rn).(P_{21}(K,L,abort,restart,tail,first,last,tag,rn)\\
&\qquad\qquad + P_{22}(K,L,abort,restart,cons(head,tail),first,last,tag,rn))\\
&\quad\mid (K(first\ last\ tag\ m).\tau.(QQ_{11}(K,L,abort,restart,rtag,end,t2enabled)\\
&\qquad + QQ_{12}(K,L,abort,restart,first,last,tag,m,rtag,end,t2enabled))\\
&\quad + \tau^2.abort.\tau.(Q(K,L,abort,restart) + QQ_{22}(K,L,abort,restart))\\
&\quad + restart.\tau^2.Q(K,L,abort,restart))))
\end{aligned}
$$

$$
\begin{aligned}
A_{32}(head,tail) \ \dot{\sim}\ \tau^3.(&\tau^3.\overline{Conf}\ \ DONTKNOW.A_{322}\\
&+ \tau^3.\overline{Conf}\ \ NOTOK.A_{327}\\
&+ \tau^4.A_{324}(head,tail)\\
&+ A_{3191}(head,tail))
\end{aligned}
$$

Considérons la partie A_{322} (l'émetteur interrompt le transfert avec une confirmation $DONTKNOW$):

$$
\begin{aligned}
A_{322} \overset{def}{=} &(\nu\ K\ L\ abort\ restart)\\
&(\overline{abort}.P(K,L,abort,restart)\\
&\mid abort.\tau.(Q(K,L,abort,restart) + QQ_{22}(K,L,abort,restart)))
\end{aligned}
$$

$$
\begin{aligned}
A_{322} \ \dot{\sim}\ &\tau.(\nu\ K\ L\ abort\ restart)\\
&(P(K,L,abort,restart)\\
&\mid \tau.(Q(K,L,abort,restart) + QQ_{22}(K,L,abort,restart)))
\end{aligned}
$$

$$
\begin{aligned}
\dot{\sim}\ &\tau.(\nu\ K\ L\ abort\ restart)\\
&\quad(P(K,L,abort,restart)\\
&\quad\mid (\tau.Q(K,L,abort,restart)\\
&\quad + \tau.\overline{Ind\_err}.Q(K,L,abort,restart)))
\end{aligned}
$$

Comme il s'agit du dernier message du fichier et que ce message a déjà été délivré, le récepteur n'envoie pas d'indication d'erreur au consommateur.

$$
\begin{aligned}
\dot{\sim}\ &\tau.(\nu\ K\ L\ abort\ restart)\\
&\quad(P(K,L,abort,restart) \mid \tau.Q(K,L,abort,restart))
\end{aligned}
$$

$$
\begin{aligned}
\dot{\sim}\ &\tau.\tau.(\nu\ K\ L\ abort\ restart)\\
&\quad(P(K,L,abort,restart) \mid Q(K,L,abort,restart))
\end{aligned}
$$

$$A_{322} \ \dot{\sim}\ \tau^2.System$$

Considérons la partie A_{327} (l'émetteur interrompt le transfert avec une confirmation $NOTOK$; le récepteur envoie une indication d'erreur au consommateur):

$$\begin{aligned}
A_{327} \stackrel{def}{=}\ & (\nu\ \ K\ \ L\ \ abort\ \ restart)\\
& (\overline{abort}.P(K, L, abort, restart)\\
& \mid abort.\tau.(Q(K, L, abort, restart)\ +\ QQ_{22}(K, L, abort, restart)))
\end{aligned}$$

$$\begin{aligned}
A_{327}\ \dot{\sim}\ \ & \tau.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
& (P(K, L, abort, restart)\\
& \mid \tau.(Q(K, L, abort, restart)\ +\ QQ_{22}(K, L, abort, restart)))
\end{aligned}$$

$$\begin{aligned}
\dot{\sim}\ \ & \tau.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
& (P(K, L, abort, restart)\\
& \mid (\tau.Q(K, L, abort, restart)\\
& +\ \tau.\overline{Ind_err}.Q(K, L, abort, restart)))
\end{aligned}$$

Comme il s'agit d'un message intermédiaire du fichier, le récepteur envoie une indication d'erreur au consommateur.

$$\begin{aligned}
\dot{\sim}\ \ & \tau.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
& (P(K, L, abort, restart)\\
& \mid \tau.\overline{Ind_err}.Q(K, L, abort, restart))
\end{aligned}$$

$$\begin{aligned}
\dot{\sim}\ \ & \tau.\tau.\overline{Ind_err}.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
& (P(K, L, abort, restart) \mid Q(K, L, abort, restart))
\end{aligned}$$

$$A_{327}\ \dot{\sim}\ \tau^2.\overline{Ind\_err}.System$$

Considérons la partie $A_{324}(head, tail)$ (le premier message déjà délivré par le récepteur a été retransmis par l'émetteur et bien reçu par le récepteur):

$$\begin{aligned}
A_{324}(head, tail) \stackrel{def}{=}\ & (\nu\ \ K\ \ L\ \ abort\ \ restart)\\
& (\tau.add(rn, 1, rn).(P_{21}(K, L, abort, restart, tail, first, last, tag, rn)\\
& \qquad +\ P_{22}(K, L, abort, restart, cons(head, tail), first, last, tag, rn))\\
& \mid \tau.(QQ_{11}(K, L, abort, restart, rtag, end, t2enabled)\\
& \quad +\ QQ_{12}(K, L, abort, restart, first, last, tag, head, rtag, end, t2enabled)))
\end{aligned}$$

$$\begin{aligned}
A_{324}(head, tail)\ \dot{\sim}\ & \tau.\tau.add(rn, 1, rn).(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
& ((P_{21}(K, L, abort, restart, tail, first, last, tag, rn)\\
& +\ P_{22}(K, L, abort, restart, cons(head, tail), first, last, tag, rn))\\
& \mid (QQ_{11}(K, L, abort, restart, rtag, end, t2enabled)\\
& +\ QQ_{12}(K, L, abort, restart, first, last, tag, head, rtag, end, t2enabled)))
\end{aligned}$$

$$\begin{aligned}
\dot{\sim}\ \ & \tau.\tau^2.(\nu\ \ K\ \ L\ \ abort\ \ restart)\\
& ((P_{21}(K, L, abort, restart, tail, first, last, tag, rn)\\
& +\ P_{22}(K, L, abort, restart, cons(head, tail), first, last, tag, rn))\\
& \mid (QQ_{11}(K, L, abort, restart, rtag, end, t2enabled)\\
& +\ QQ_{12}(K, L, abort, restart, first, last, tag, head, rtag, end, t2enabled)))
\end{aligned}$$

$$A_{324}(head, tail) \stackrel{\cdot}{\sim} \tau.B_{21}(head, tail)$$

Si on récapitule, on obtient les 24 équations suivantes:

$$\begin{aligned} System \stackrel{\cdot}{\sim} \tau^7.(&Req(Nil).A_2 + Req(cons(head, tail)).A_3(head, tail) \\ &+ Req(cons(head, tail)).\tau^4.A_5(head, tail)) \end{aligned}$$

$$A_2 \stackrel{\cdot}{\sim} \tau.\overline{Conf}\ OK.\tau^2.System$$

$$A_5(head, tail) \stackrel{\cdot}{\sim} \tau^4.(A_3(head, tail) + \tau^4.A_5(head, tail))$$

$$A_3(head, tail) \stackrel{\cdot}{\sim} \tau^{10}.AA_3(head, tail)$$

$$\begin{aligned} AA_3(head, tail) \stackrel{\cdot}{\sim} (&\tau.\overline{Ind}\ head\ LAST.A_{31}(tail) \\ &+ \tau.\overline{Ind}\ head\ FIRST.A_{31}(tail) \\ &+ \tau^3.\overline{Ind}\ head\ LAST.A_{32}(head, tail) \\ &+ \tau^3.\overline{Ind}\ head\ FIRST.A_{32}(head, tail)) \end{aligned}$$

$$A_{31}(tail) \stackrel{\cdot}{\sim} \tau^7.(A_{314} + A_{315}(hd, tl) + A_{319}(hd, tl))$$

$$A_{314} \stackrel{\cdot}{\sim} \tau.\overline{Conf}\ OK.\tau^3.System$$

$$\begin{aligned} A_{319}(hd,tl) \stackrel{\cdot}{\sim} \tau^8.(&\tau^3.\overline{Conf}\ DONTKNOW.A_{3192} \\ &+ \tau^3.\overline{Conf}\ NOTOK.A_{3192} \\ &+ A_{315}(hd, tl) \\ &+ A_{319}(hd, tl)) \end{aligned}$$

$$A_{3192} \stackrel{\cdot}{\sim} \tau^2.\overline{Ind\_err}.System$$

$$A_{315}(hd, tl) \stackrel{\cdot}{\sim} \tau^5.B(hd, tl)$$

$$\begin{aligned} B(hd, tl) \stackrel{\cdot}{\sim}\ &\tau^7.\overline{Ind}\ hd\ LAST.A_{31}(tl) \\ &+ \tau^7.\overline{Ind}\ hd\ INCOMPLETE.A_{31}(tl) \\ &+ \tau^{10}.\overline{Ind}\ hd\ LAST.\tau.\overline{Conf}\ DONTKNOW.B_3 \\ &+ \tau^{10}.\overline{Ind}\ hd\ INCOMPLETE.\tau.\overline{Conf}\ NOTOK.BB_3 \\ &+ \tau^9.\overline{Ind}\ hd\ LAST.B_4(hd, tl) \\ &+ \tau^9.\overline{Ind}\ hd\ INCOMPLETE.B_4(hd, tl) \end{aligned}$$

$$B_3 \stackrel{\cdot}{\sim} \tau^6.System$$

$$BB_3 \stackrel{\cdot}{\sim} \tau^6.\overline{Ind\_err}.System$$

$$B_4(hd, tl) \stackrel{\cdot}{\sim} \tau^8.B_{21}(hd, tl) + \tau^3.A_{3191}(hd, tl)$$

$$\begin{aligned} A_{3191}(hd,tl) \;\dot{\sim}\; \tau^8.(&\tau^3.\overline{Conf}\ \ DONTKNOW.A_{322} \\ &+\ \tau^3.\overline{Conf}\ \ NOTOK.A_{327} \\ &+\ A_{3151}(hd,tl) \\ &+\ A_{3191}(hd,tl)) \end{aligned}$$

$$A_{3151}(hd,tl) \;\dot{\sim}\; \tau^5.B_{21}(hd,tl)$$

$$\begin{aligned} B_{21}(hd,tl) \;\dot{\sim}\; &A_{31}(tl) \\ &+\ \tau^6.\overline{Conf}\ \ DONTKNOW.B_1 \\ &+\ \tau^6.\overline{Conf}\ \ NOTOK.BB_1 \\ &+\ B_2(hd,tl) \end{aligned}$$

$$B_1 \;\dot{\sim}\; \tau^4.System$$

$$BB_1 \;\dot{\sim}\; \tau^4.\overline{Ind\_err}.System$$

$$B_2(hd,tl) \;\dot{\sim}\; \tau^{10}.B_{21}(hd,tl)\ +\ \tau^5.A_{3191}(hd,tl)$$

$$\begin{aligned} A_{32}(head,tail) \;\dot{\sim}\; \tau^3.(&\tau^3.\overline{Conf}\ \ DONTKNOW.A_{322} \\ &+\ \tau^3.\overline{Conf}\ \ NOTOK.A_{327} \\ &+\ \tau^4.A_{324}(head,tail) \\ &+\ A_{3191}(head,tail)) \end{aligned}$$

$$A_{322} \;\dot{\sim}\; \tau^2.System$$

$$A_{327} \;\dot{\sim}\; \tau^2.\overline{Ind\_err}.System$$

$$A_{324}(head,tail) \;\dot{\sim}\; \tau.B_{21}(head,tail)$$

Les équations ci-dessus sont valides quel que soit le fichier f à transférer (i.e, quelle que soit sa taille et quel que soit son contenu). Elles sont donc valides pour l'équivalence forte non close ($\sim$).

7.3 Équivalence de la spécification et de l'implémentation du protocole

7.3.1 Définition de la spécification par un système d'équations récursives

Rappelons la spécification abstraite du protocole:

$$S_0 \stackrel{def}{=} Req(Nil).\overline{Conf}\ OK.S_0\ +\ Req(cons(head,tail)).S_1(head,tail)$$

$$\begin{aligned} S_1(head,tail) \stackrel{def}{=}\ & \tau.\overline{Ind}\ head\ LAST.\overline{Conf}\ OK.S_0 \\ & +\ \tau.\overline{Ind}\ head\ FIRST.S_2(hd,tl) \\ & +\ \tau.\overline{Ind}\ head\ LAST.\overline{Conf}\ DONTKNOW.S_0 \\ & +\ \tau.\overline{Ind}\ head\ FIRST.\overline{Conf}\ NOTOK.\overline{Inderr}.S_0 \end{aligned}$$

$$\begin{aligned} S_2(hd,tl) \stackrel{def}{=}\ & \tau.\overline{Conf}\ DONTKNOW.\overline{Ind\_err}.S_0 \\ & +\ \tau.\overline{Conf}\ NOTOK.\overline{Ind\_err}.S_0 \\ & +\ \tau.\overline{Ind}\ hd\ LAST.\overline{Conf}\ DONTKNOW.S_0 \\ & +\ \tau.\overline{Ind}\ hd\ INCOMPLETE.\overline{Conf}\ NOTOK.\overline{Inderr}.S_0 \\ & +\ \tau.\overline{Ind}\ hd\ LAST.\overline{Conf}\ OK.S_0 \\ & +\ \tau.\overline{Ind}\ hd\ INCOMPLETE.S_2(hdtl,tltl) \end{aligned}$$

Soit $\widetilde{S} = < S_0, S_1(head,tail), S_2(hd,tl) >$ un ensemble de constantes d'agents.

On définit la spécification abstraite par un système d'équations récursives contenant les expressions d'agents $F_0(\widetilde{S})$, ..., $F_2(\widetilde{S})$ définies par:

$$F_0(\widetilde{S})\ =\ Req(Nil).\overline{Conf}\ OK.S_0\ +\ Req(cons(head,tail)).S_1(head,tail)$$

$$\begin{aligned} F_1(\widetilde{S})\ =\ & \tau.\overline{Ind}\ head\ LAST.\overline{Conf}\ OK.S_0 \\ & +\ \tau.\overline{Ind}\ head\ FIRST.S_2(hd,tl) \\ & +\ \tau.\overline{Ind}\ head\ LAST.\overline{Conf}\ DONTKNOW.S_0 \\ & +\ \tau.\overline{Ind}\ head\ FIRST.\overline{Conf}\ NOTOK.\overline{Inderr}.S_0 \end{aligned}$$

$$\begin{aligned} F_2(\widetilde{S})\ =\ & \tau.\overline{Conf}\ DONTKNOW.\overline{Ind\_err}.S_0 \\ & +\ \tau.\overline{Conf}\ NOTOK.\overline{Ind\_err}.S_0 \\ & +\ \tau.\overline{Ind}\ hd\ LAST.\overline{Conf}\ DONTKNOW.S_0 \\ & +\ \tau.\overline{Ind}\ hd\ INCOMPLETE.\overline{Conf}\ NOTOK.\overline{Inderr}.S_0 \\ & +\ \tau.\overline{Ind}\ hd\ LAST.\overline{Conf}\ OK.S_0 \\ & +\ \tau.\overline{Ind}\ hd\ INCOMPLETE.S_2(hdtl,tltl) \end{aligned}$$

Ainsi, la spécification abstraite du protocole peut s'écrire $S_i \stackrel{def}{=} F_i(\widetilde{S})$, $i = 0\ ..\ 2$ (i.e, $\widetilde{S} \stackrel{def}{=} \widetilde{F}(\widetilde{S})$).

On peut remarquer que les équations $\widetilde{F}$ sont gardées et séquentielles. Donc, le système d'équations $\widetilde{S} \stackrel{def}{=} \widetilde{F}(\widetilde{S})$, possède une **solution unique** modulo l'équivalence faible.

7.3.2 Définition du protocole par un système d'équations récursives

Soit l'ensemble des constantes d'agents

$$\begin{aligned}\widetilde{System} \stackrel{def}{=} \; < & System, A_2, A_3(head,tail), AA_3(head,tail), A_{31}(tail),\\ & A_{314}, A_{315}(hd,tl), B_1, B_2(hd,tl), B_3, B_4(hd,tl),\\ & A_{32}(head,tail), A_{322}, A_{324}(head,tail), B(hd,tl), A_5(head,tail),\\ & A_{319}(hd,tl), A_{3191}(hd,tl), A_{3192}, B_{21}(hd,tl), BB_1, BB_3, A_{327}, A_{3151}(hd,tl) >\end{aligned}$$

Et soient les constantes d'agents:

$$\begin{aligned}\widetilde{D} \stackrel{def}{=} \; < & D_0, D_1, D_2(head,tail), D_3(head,tail), D_4(tail),\\ & D_5, D_6(hd,tl), D_7, D_8(hd,tl), D_9, D_{10}(hd,tl),\\ & D_{11}(head,tail), D_{12}, D_{13}(head,tail), D_{14}(hd,tl), D_{15}(head,tail),\\ & D_{16}(hd,tl), D_{17}(hd,tl), D_{18}, D_{19}(hd,tl), D_{20}, D_{21}, D_{22}, D_{23}(hd,tl) >\end{aligned}$$

On définit le protocole par un système d'équations récursives contenant les expressions d'agents $E_0(\widetilde{D})$, $E_1(\widetilde{D})$, ..., $E_{23}(\widetilde{D})$ définies par:

$$\begin{aligned}E_0(\widetilde{D}) \stackrel{def}{=} \; & \tau^7.(Req(Nil).D_1 \; + \; Req(cons(head,tail)).D_2(head,tail)\\ & + \; Req(cons(head,tail)).\tau^4.D_{15}(head,tail))\end{aligned}$$

$$E_1(\widetilde{D}) \stackrel{def}{=} \tau.\overline{Conf} \; OK.\tau^2.D_0$$

$$E_2(\widetilde{D}) \stackrel{def}{=} \tau^{10}.D_3(head,tail)$$

$$\begin{aligned}E_3(\widetilde{D}) \stackrel{def}{=} \; & \tau.\overline{Ind} \; head \; LAST.D_4(tail)\\ & + \; \tau.\overline{Ind} \; head \; FIRST.D_4(tail)\\ & + \; \tau^3.\overline{Ind} \; head \; LAST.D_{11}(head,tail)\\ & + \; \tau^3.\overline{Ind} \; head \; FIRST.D_{11}(head,tail)\end{aligned}$$

$$E_4(\widetilde{D}) \stackrel{def}{=} \tau^7.(D_5 \; + \; D_6(hd,tl) \; + \; D_{16}(hd,tl))$$

$$E_5(\widetilde{D}) \stackrel{def}{=} \tau.\overline{Conf} \; OK.\tau^3.D_0$$

$$E_6(\widetilde{D}) \stackrel{def}{=} \tau^5.D_{14}(hd,tl)$$

$$E_7(\widetilde{D}) \stackrel{def}{=} \tau^4.D_0$$

$$E_8(\widetilde{D}) \stackrel{def}{=} \tau^{10}.D_{19}(hd,tl) \; + \; \tau^5.D_{17}(hd,tl)$$

$$E_9(\widetilde{D}) \stackrel{def}{=} \tau^6.D_0$$

$$E_{10}(\widetilde{D}) \stackrel{def}{=} \tau^8.D_{19}(hd,tl) \; + \; \tau^3.D_{17}(hd,tl)$$

$$
\begin{aligned}
E_{11}(\widetilde{D}) \overset{def}{=}\ & \tau^3.(\tau^3.\overline{Conf}\ \ DONTKNOW.D_{12}\\
& \quad +\ \tau^3.\overline{Conf}\ \ NOTOK.D_{22}\\
& \quad +\ \tau^4.D_{13}(head, tail)\\
& \quad +\ D_{17}(head, tail))
\end{aligned}
$$

$$E_{12}(\widetilde{D}) \overset{def}{=} \tau^2.D_0$$

$$E_{13}(\widetilde{D}) \overset{def}{=} \tau.D_{19}(head, tail)$$

$$
\begin{aligned}
E_{14}(\widetilde{D}) \overset{def}{=}\ & \tau^7.\overline{Ind}\ \ hd\ \ LAST.D_4(tl)\\
& +\ \tau^7.\overline{Ind}\ \ hd\ \ INCOMPLETE.D_4(tl)\\
& +\ \tau^{10}.\overline{Ind}\ \ hd\ \ LAST.\tau.\overline{Conf}\ \ DONTKNOW.D_9\\
& +\ \tau^{10}.\overline{Ind}\ \ hd\ \ INCOMPLETE.\tau.\overline{Conf}\ \ NOTOK.D_{21}\\
& +\ \tau^9.\overline{Ind}\ \ hd\ \ LAST.D_{10}(hd, tl)\\
& +\ \tau^9.\overline{Ind}\ \ hd\ \ INCOMPLETE.D_{10}(hd, tl)
\end{aligned}
$$

$$E_{15}(\widetilde{D}) \overset{def}{=} \tau^4.(D_2(head, tail)\ +\ \tau^4.D_{15}(head, tail))$$

$$
\begin{aligned}
E_{16}(\widetilde{D}) \overset{def}{=}\ & \tau^8.(\tau^3.\overline{Conf}\ \ DONTKNOW.D_{18}\\
& \quad +\ \tau^3.\overline{Conf}\ \ NOTOK.D_{18}\\
& \quad +\ D_6(hd, tl)\\
& \quad +\ D_{16}(hd, tl))
\end{aligned}
$$

$$
\begin{aligned}
E_{17}(\widetilde{D}) \overset{def}{=}\ & \tau^8.(\tau^3.\overline{Conf}\ \ DONTKNOW.D_{12}\\
& \quad +\ \tau^3.\overline{Conf}\ \ NOTOK.D_{22}\\
& \quad +\ D_{23}(hd, tl)\\
& \quad +\ D_{17}(hd, tl))
\end{aligned}
$$

$$E_{18}(\widetilde{D}) \overset{def}{=} \tau^2.\overline{Ind\_err}.D_0$$

$$
\begin{aligned}
E_{19}(\widetilde{D}) \overset{def}{=}\ & D_4(tl)\\
& +\ \tau^6.\overline{Conf}\ \ DONTKNOW.D_7\\
& +\ \tau^6.\overline{Conf}\ \ NOTOK.D_{20}\\
& +\ D_8(hd, tl)
\end{aligned}
$$

$$E_{20}(\widetilde{D}) \overset{def}{=} \tau^4.\overline{Ind\_err}.D_0$$

$$E_{21}(\widetilde{D}) \overset{def}{=} \tau^6.\overline{Ind\_err}.D_0$$

$$E_{22}(\widetilde{D}) \overset{def}{=} \tau^2.\overline{Ind\_err}.D_0$$

$$E_{23}(\widetilde{D}) \overset{def}{=} \tau^5.D_{19}(hd, tl)$$

On définit les équations $D_i \stackrel{def}{=} E_i(\widetilde{D})$, $\forall\ i = 0\ ..\ 23$.

Comme chaque D_i est faiblement gardé, on sait que le système d'équations $\widetilde{D} \stackrel{def}{=} \widetilde{E}(\widetilde{D})$ possède une **solution unique** modulo l'équivalence forte.

Or, on sait aussi que $\widetilde{System} \sim \widetilde{E}(\widetilde{System})$. Il en résulte que $\widetilde{System} \sim \widetilde{D}$ et en particulier que $System \sim D_0$.

7.3.3 Regroupement des équations équivalentes du protocole

Avant de prouver que D_0 est observationnellement équivalent à la spécification abstraite du protocole, on va d'abord simplifier les équations récursives définissant le protocole à l'aide de la congruence observationnelle (i.e, équivalence faible non close $\simeq$).

On peut regrouper les équations équivalentes. Ainsi, en utilisant la loi (T0), on peut observer que:

$$E_7(\widetilde{D}) \ \simeq\ \tau.D_0$$

$$E_9(\widetilde{D}) \ \simeq\ \tau.D_0$$

$$E_{12}(\widetilde{D}) \ \simeq\ \tau.D_0$$

En utilisant la loi (T0), on peut observer également que:

$$E_{18}(\widetilde{D}) \ \simeq\ \tau.\overline{Ind\_err}.D_0$$

$$E_{20}(\widetilde{D}) \ \simeq\ \tau.\overline{Ind\_err}.D_0$$

$$E_{21}(\widetilde{D}) \ \simeq\ \tau.\overline{Ind\_err}.D_0$$

$$E_{22}(\widetilde{D}) \ \simeq\ \tau.\overline{Ind\_err}.D_0$$

De même, en utilisant la loi (T0), on peut observer que:

$$E_1(\widetilde{D}) \ \simeq\ \tau.\overline{Conf}\ \ OK.D_0$$

$$E_5(\widetilde{D}) \ \simeq\ \tau.\overline{Conf}\ \ OK.D_0$$

En utilisant la loi (T0), on a aussi:

$$E_8(\widetilde{D}) \ \simeq\ \tau.D_{19}(hd,tl)\ +\ \tau.D_{17}(hd,tl)$$

$$E_{10}(\widetilde{D}) \ \simeq\ \tau.D_{19}(hd,tl)\ +\ \tau.D_{17}(hd,tl)$$

On définit alors un nouveau système d'équations par:

$$
\begin{aligned}
E'_i(\widetilde{D}) \stackrel{def}{=} \; & \tau.\overline{Conf}\ OK.D_0,\ i = 1,5 \\
& \tau.D_0,\ i = 7,9,12 \\
& \tau.\overline{Ind\_err}.D_0,\ i = 18,20,21,22 \\
& \tau.D_{19}(hd,tl) \ + \ \tau.D_{17}(hd,tl),\ i = 8,10 \\
& E_i(\widetilde{D}),\ \forall\ i\ \in\ \{0,2,3,4,6,11,13,14,15,16,17,19,23\}
\end{aligned}
$$

Soient D'_i, $i = 0$ .. 23 de nouvelles constantes d'agents et $\widetilde{D'} \stackrel{def}{=} \widetilde{E'}(\widetilde{D'})$.
On sait maintenant que:
$\widetilde{E} \simeq \widetilde{E'}$,
$\widetilde{D} \stackrel{def}{=} \widetilde{E}(\widetilde{D})$
et $\widetilde{D'} \stackrel{def}{=} \widetilde{E'}(\widetilde{D'})$.
À l'aide de la règle (U0), on peut conclure que $\widetilde{D} \simeq \widetilde{D'}$ et en particulier que $D_0 \simeq D'_0$.

7.3.4 Suppression des boucles τ du protocole

Dans cette étape, on va supprimer les boucles internes (τ) du protocole. Une première boucle apparaît dans $D'_{15}(head, tail)$:

$$D'_{15}(head,tail) \stackrel{def}{=} \tau^4.(D'_2(head,tail) \ + \ \tau^4.D'_{15}(head,tail))$$

$$\simeq \ \tau.D'_2(head,tail) \ + \ \tau.D'_{15}(head,tail)$$

En appliquant la règle (L) d'élimination de boucles τ, on obtient:

$$D'_{15}(head,tail) \ \simeq \ \tau.D'_2(head,tail)$$

De la même manière, on va supprimer la boucle τ apparaissant dans $D'_{16}(hd,tl)$:

$$
\begin{aligned}
D'_{16}(hd,tl) \stackrel{def}{=} \; & \tau^8.(\tau^3.\overline{Conf}\ \ DONTKNOW.D'_{18} \\
& + \ \tau^3.\overline{Conf}\ \ NOTOK.D'_{18} \\
& + \ D'_6(hd,tl) \\
& + \ D'_{16}(hd,tl))
\end{aligned}
$$

$$
\begin{aligned}
\simeq \; & \tau.\overline{Conf}\ \ DONTKNOW.D'_{18} \\
& + \ \tau.\overline{Conf}\ \ NOTOK.D'_{18} \\
& + \ \tau.D'_6(hd,tl) \\
& + \ \tau.D'_{16}(hd,tl)
\end{aligned}
$$

En appliquant la règle (L) d'élimination de boucles τ, on obtient:

$$
\begin{aligned}
D'_{16}(hd,tl) \simeq \; & \tau.\overline{Conf}\ \ DONTKNOW.D'_{18} \\
& + \ \tau.\overline{Conf}\ \ NOTOK.D'_{18} \\
& + \ \tau.D'_6(hd,tl)
\end{aligned}
$$

On va aussi supprimer la boucle τ apparaissant dans $D'_{17}(hd,tl)$:

$$\begin{aligned} D'_{17}(hd,tl) \stackrel{def}{=} \tau^8.(&\tau^3.\overline{Conf}\ \ DONTKNOW.D'_{12} \\ +\ &\tau^3.\overline{Conf}\ \ NOTOK.D'_{22} \\ +\ &D'_{23}(hd,tl) \\ +\ &D'_{17}(hd,tl)) \end{aligned}$$

$$\begin{aligned} \simeq\ &\tau.\overline{Conf}\ \ DONTKNOW.D'_{12} \\ &+\ \tau.\overline{Conf}\ \ NOTOK.D'_{22} \\ &+\ \tau.D'_{23}(hd,tl) \\ &+\ \tau.D'_{17}(hd,tl) \end{aligned}$$

$$\begin{aligned} \simeq\ &\tau.\overline{Conf}\ \ DONTKNOW.D'_{7} \\ &+\ \tau.\overline{Conf}\ \ NOTOK.D'_{18} \\ &+\ \tau.D'_{23}(hd,tl) \\ &+\ \tau.D'_{17}(hd,tl) \end{aligned}$$

En appliquant la règle (L) d'élimination de boucles τ, on obtient:

$$\begin{aligned} D'_{17}(hd,tl) \simeq\ &\tau.\overline{Conf}\ \ DONTKNOW.D'_{7} \\ +\ &\tau.\overline{Conf}\ \ NOTOK.D'_{18} \\ +\ &\tau.D'_{23}(hd,tl) \end{aligned}$$

On définit alors un nouveau système d'équations par:

$$E''_{15}(\widetilde{D}) \stackrel{def}{=} \tau.D'_2(head,tail)$$

$$\begin{aligned} E''_{16}(\widetilde{D}) \stackrel{def}{=}\ &\tau.\overline{Conf}\ \ DONTKNOW.D'_{18} \\ +\ &\tau.\overline{Conf}\ \ NOTOK.D'_{18} \\ +\ &\tau.D'_{6}(hd,tl) \end{aligned}$$

$$\begin{aligned} D''_{17}(hd,tl) \stackrel{def}{=}\ &\tau.\overline{Conf}\ \ DONTKNOW.D'_{7} \\ +\ &\tau.\overline{Conf}\ \ NOTOK.D'_{18} \\ +\ &\tau.D'_{23}(hd,tl) \end{aligned}$$

`et` $\forall\ i \in$ `0 .. 23,` $i \neq 15$`,` $i \neq 16$ `et` $i \neq 17$`:`

$$E''_i(\widetilde{D}) \stackrel{def}{=} E'_i(\widetilde{D})$$

Soient D''_i, $i = 0$ .. 23 de nouvelles constantes d'agents et $\widetilde{D''} \stackrel{def}{=} \widetilde{E''}(\widetilde{D''})$. On a donc:
$\widetilde{E'} \simeq \widetilde{E''}$,
$\widetilde{D'} \stackrel{def}{=} \widetilde{E'}(\widetilde{D'})$
et $\widetilde{D''} \stackrel{def}{=} \widetilde{E''}(\widetilde{D''})$.
À l'aide de la règle (U0), on peut conclure que $\widetilde{D'} \simeq \widetilde{D''}$ et en particulier que $D'_0 \simeq D''_0$.

7.3.5 Équivalence observationnelle du protocole et de sa spécification

Maintenant, on va montrer que D_0'' satisfait les équations de définiton de S_0. La première étape consiste à simplifier les équations de définition D_i'', $i = 0 .. 23$ en utilisant les τ-lois du π-calcul et en identifiant et en substituant les expressions équivalentes dans les équations.

Considérons $D_6''(hd, tl)$:

$$D_6''(hd,tl) \stackrel{def}{=} \tau^5.D_{14}''(hd,tl)$$

$$D_6''(hd,tl) \simeq \tau.D_{14}''(hd,tl)$$

Considérons $D_{16}''(hd, tl)$:

$$\begin{aligned} D_{16}''(hd,tl) \stackrel{def}{=} & \ \tau.\overline{Conf}\ \ DONTKNOW.D_{18}'' \\ & + \tau.\overline{Conf}\ \ NOTOK.D_{18}'' \\ & + \tau.D_6''(hd,tl) \end{aligned}$$

On remplace $D_6''(hd, tl)$ par sa définition.

$$\begin{aligned} \simeq & \ \tau.\overline{Conf}\ \ DONTKNOW.D_{18}'' \\ & + \tau.\overline{Conf}\ \ NOTOK.D_{18}'' \\ & + \tau.\tau.D_{14}''(hd,tl) \end{aligned}$$

$$\begin{aligned} D_{16}''(hd,tl) \simeq & \ \tau.\overline{Conf}\ \ DONTKNOW.D_{18}'' \\ & + \tau.\overline{Conf}\ \ NOTOK.D_{18}'' \\ & + \tau.D_{14}''(hd,tl) \end{aligned}$$

Considérons $D_4''(tail)$:

$$D_4''(tail) \stackrel{def}{=} \tau^7.(D_5'' + D_6''(hd,tl) + D_{16}''(hd,tl))$$

$$\simeq \tau.D_5'' + \tau.D_6''(hd,tl) + \tau.D_{16}''(hd,tl)$$

On remplace D_5'' par D_1'' et on remplace $D_6''(hd, tl)$ et $D_{16}''(hd, tl)$ par leur définition.

$$\begin{aligned} \simeq & \ \tau.D_1'' + \tau.\tau.D_{14}''(hd,tl) \\ & + \tau.(\tau.\overline{Conf}\ \ DONTKNOW.D_{18}'' \\ & \quad + \tau.\overline{Conf}\ \ NOTOK.D_{18}'' \\ & \quad + \tau.D_{14}''(hd,tl)) \end{aligned}$$

$$\begin{aligned} \simeq & \ \tau.D_1'' + \tau.D_{14}''(hd,tl) \\ & + \tau.(\tau.\overline{Conf}\ \ DONTKNOW.D_{18}'' \\ & \quad + \tau.\overline{Conf}\ \ NOTOK.D_{18}'' \\ & \quad + \tau.D_{14}''(hd,tl)) \end{aligned}$$

En utilisant la loi (T2), on obtient:

$$\begin{aligned}
\simeq\ & \tau.D_1'' \\
& + \tau.(\tau.\overline{Conf}\ \ DONTKNOW.D''_{18} \\
& \qquad + \tau.\overline{Conf}\ \ NOTOK.D''_{18} \\
& \qquad + \tau.D''_{14}(hd,tl))
\end{aligned}$$

$$D''_4(tail) \simeq \tau.D''_1 + \tau.D''_{16}(hd,tl)$$

Considérons $D''_{14}(hd,tl)$:

$$\begin{aligned}
D''_{14}(hd,tl) \stackrel{def}{=}\ & \tau^7.\overline{Ind}\ hd\ \ LAST.D''_4(tl) \\
& + \tau^7.\overline{Ind}\ hd\ \ INCOMPLETE.D''_4(tl) \\
& + \tau^{10}.\overline{Ind}\ hd\ \ LAST.\tau.\overline{Conf}\ \ DONTKNOW.D''_9 \\
& + \tau^{10}.\overline{Ind}\ hd\ \ INCOMPLETE.\tau.\overline{Conf}\ \ NOTOK.D''_{21} \\
& + \tau^9.\overline{Ind}\ hd\ \ LAST.D''_{10}(hd,tl) \\
& + \tau^9.\overline{Ind}\ hd\ \ INCOMPLETE.D''_{10}(hd,tl)
\end{aligned}$$

$$\begin{aligned}
\simeq\ & \tau.\overline{Ind}\ hd\ \ LAST.D''_4(tl) \\
& + \tau.\overline{Ind}\ hd\ \ INCOMPLETE.D''_4(tl) \\
& + \tau.\overline{Ind}\ hd\ \ LAST.\tau.\overline{Conf}\ \ DONTKNOW.D''_9 \\
& + \tau.\overline{Ind}\ hd\ \ INCOMPLETE.\tau.\overline{Conf}\ \ NOTOK.D''_{21} \\
& + \tau.\overline{Ind}\ hd\ \ LAST.D''_{10}(hd,tl) \\
& + \tau.\overline{Ind}\ hd\ \ INCOMPLETE.D''_{10}(hd,tl)
\end{aligned}$$

On remplace D''_9 par D''_7, D''_{21} par D''_{18} et D''_{10} par D''_8.

$$\begin{aligned}
\simeq\ & \tau.\overline{Ind}\ hd\ \ LAST.D''_4(tl) \\
& + \tau.\overline{Ind}\ hd\ \ INCOMPLETE.D''_4(tl) \\
& + \tau.\overline{Ind}\ hd\ \ LAST.\tau.\overline{Conf}\ \ DONTKNOW.D''_7 \\
& + \tau.\overline{Ind}\ hd\ \ INCOMPLETE.\tau.\overline{Conf}\ \ NOTOK.D''_{18} \\
& + \tau.\overline{Ind}\ hd\ \ LAST.D''_8(hd,tl) \\
& + \tau.\overline{Ind}\ hd\ \ INCOMPLETE.D''_8(hd,tl)
\end{aligned}$$

On remplace $D''_4(tl)$ par sa définition. Supposons que $hdtl$ et $tltl$ sont respectivement la tête et le reste du fichier tl.

$$\begin{aligned}
\simeq\ & \tau.\overline{Ind}\ hd\ \ LAST.(\tau.D''_1 + \tau.D''_{16}(hdtl,tltl)) \\
& + \tau.\overline{Ind}\ hd\ \ INCOMPLETE.(\tau.D''_1 + \tau.D''_{16}(hdtl,tltl)) \\
& + \tau.\overline{Ind}\ hd\ \ LAST.\tau.\overline{Conf}\ \ DONTKNOW.D''_7 \\
& + \tau.\overline{Ind}\ hd\ \ INCOMPLETE.\tau.\overline{Conf}\ \ NOTOK.D''_{18} \\
& + \tau.\overline{Ind}\ hd\ \ LAST.D''_8(hd,tl) \\
& + \tau.\overline{Ind}\ hd\ \ INCOMPLETE.D''_8(hd,tl)
\end{aligned}$$

S'il s'agit du dernier message du fichier, on va dans l'état D''_1. Sinon, on va dans l'état D''_{16}.

$$\begin{aligned}\simeq\ & \tau.\overline{Ind}\ hd\ LAST.\tau.D''_1\\ &+ \tau.\overline{Ind}\ hd\ INCOMPLETE.\tau.D''_{16}(hdtl, tltl)\\ &+ \tau.\overline{Ind}\ hd\ LAST.\tau.\overline{Conf}\ DONTKNOW.D''_7\\ &+ \tau.\overline{Ind}\ hd\ INCOMPLETE.\tau.\overline{Conf}\ NOTOK.D''_{18}\\ &+ \tau.\overline{Ind}\ hd\ LAST.D''_8(hd, tl)\\ &+ \tau.\overline{Ind}\ hd\ INCOMPLETE.D''_8(hd, tl)\end{aligned}$$

$$\begin{aligned}D''_{14}(hd{,}tl)\ \simeq\ & \tau.\overline{Ind}\ hd\ LAST.D''_1\\ &+ \tau.\overline{Ind}\ hd\ INCOMPLETE.D''_{16}(hdtl, tltl)\\ &+ \tau.\overline{Ind}\ hd\ LAST.\tau.\overline{Conf}\ DONTKNOW.D''_7\\ &+ \tau.\overline{Ind}\ hd\ INCOMPLETE.\tau.\overline{Conf}\ NOTOK.D''_{18}\\ &+ \tau.\overline{Ind}\ hd\ LAST.D''_8(hd, tl)\\ &+ \tau.\overline{Ind}\ hd\ INCOMPLETE.D''_8(hd, tl)\end{aligned}$$

Considérons $D''_{19}(hd, tl)$:

$$\begin{aligned}D''_{19}(hd{,}tl)\ \stackrel{def}{=}\ & D''_4(tl)\\ &+ \tau^6.\overline{Conf}\ DONTKNOW.D''_7\\ &+ \tau^6.\overline{Conf}\ NOTOK.D''_{20}\\ &+ D''_8(hd, tl)\end{aligned}$$

$$\begin{aligned}\simeq\ & D''_4(tl)\\ &+ \tau.\overline{Conf}\ DONTKNOW.D''_7\\ &+ \tau.\overline{Conf}\ NOTOK.D''_{20}\\ &+ D''_8(hd, tl)\end{aligned}$$

On remplace D''_{20} par D''_{18} et $D''_8(hd, tl)$ par sa définition.

$$\begin{aligned}\simeq\ & D''_4(tl)\\ &+ \tau.\overline{Conf}\ DONTKNOW.D''_7\\ &+ \tau.\overline{Conf}\ NOTOK.D''_{18}\\ &+ \tau.D''_{19}(hd, tl)\ +\ \tau.D''_{17}(hd, tl)\end{aligned}$$

Appliquons la loi (L) pour supprimer la boucle τ dans $D''_{19}(hd, tl)$.

$$\begin{aligned}\simeq\ & \tau.D''_4(tl)\\ &+ \tau.\overline{Conf}\ DONTKNOW.D''_7\\ &+ \tau.\overline{Conf}\ NOTOK.D''_{18}\\ &+ \tau.D''_{17}(hd, tl)\end{aligned}$$

On remplace $D''_4(tl)$ par sa définition. Ci-dessous, $hdtl$ et $tltl$ représentent respectivement la tête et le reste du fichier tl.

$$\begin{aligned}
\simeq\ & \tau.(\tau.D''_1\ +\ \tau.D''_{16}(hdtl,tltl)) \\
& +\ \tau.\overline{Conf}\ \ DONTKNOW.D''_7 \\
& +\ \tau.\overline{Conf}\ \ NOTOK.D''_{18} \\
& +\ \tau.D''_{17}(hd,tl)
\end{aligned}$$

$$\begin{aligned}
D''_{19}(hd{,}tl)\ \simeq\ & \tau.D''_1 \\
& +\ \tau.D''_{16}(hdtl,tltl) \\
& +\ \tau.\overline{Conf}\ \ DONTKNOW.D''_7 \\
& +\ \tau.\overline{Conf}\ \ NOTOK.D''_{18} \\
& +\ \tau.D''_{17}(hd,tl)
\end{aligned}$$

Considérons $D''_{23}(hd,tl)$:

$$D''_{23}(hd,tl)\ \stackrel{def}{=}\ \tau^5.D''_{19}(hd,tl)$$

$$\simeq\ \tau.D''_{19}(hd,tl)$$

Comme toutes les parties de $D''_{19}(hd,tl)$ sont préfixées par τ:

$$D''_{23}(hd,tl)\ \simeq\ D''_{19}(hd,tl)$$

Considérons $D''_{17}(hd,tl)$:

$$\begin{aligned}
D''_{17}(hd,tl)\ \stackrel{def}{=}\ & \tau.\overline{Conf}\ \ DONTKNOW.D''_7 \\
& +\ \tau.\overline{Conf}\ \ NOTOK.D''_{18} \\
& +\ \tau.D''_{23}(hd,tl)
\end{aligned}$$

On remplace $D''_{23}(hd,tl)$ par $D''_{19}(hd,tl)$.

$$\begin{aligned}
\simeq\ & \tau.\overline{Conf}\ \ DONTKNOW.D''_7 \\
& +\ \tau.\overline{Conf}\ \ NOTOK.D''_{18} \\
& +\ \tau.D''_{19}(hd,tl)
\end{aligned}$$

On remplace $D''_{19}(hd,tl)$ par sa définition.

$$\begin{aligned}
\simeq\ & \tau.\overline{Conf}\ \ DONTKNOW.D''_7 \\
& +\ \tau.\overline{Conf}\ \ NOTOK.D''_{18} \\
& +\ \tau.(\tau.D''_1 \\
& \quad +\ \tau.D''_{16}(hdtl,tltl) \\
& \quad +\ \tau.\overline{Conf}\ \ DONTKNOW.D''_7 \\
& \quad +\ \tau.\overline{Conf}\ \ NOTOK.D''_{18} \\
& \quad +\ \tau.D''_{17}(hd,tl))
\end{aligned}$$

En appliquant la loi (T2), on obtient:

$$\begin{aligned}\simeq\ \tau.(\tau.&D''_1\\ &+\ \tau.D''_{16}(hdtl,tltl)\\ &+\ \tau.\overline{Conf}\ \ DONTKNOW.D''_7\\ &+\ \tau.\overline{Conf}\ \ NOTOK.D''_{18}\\ &+\ \tau.D''_{17}(hd,tl)\end{aligned}$$

Comme toutes les parties de $D''_{19}(hd,tl)$ sont préfixées par τ:

$$D''_{17}(hd,tl)\ \simeq\ D''_{19}(hd,tl)$$

Reprenons $D''_{19}(hd,tl)$:

$$\begin{aligned}D''_{19}(hd,tl)\ \simeq\ &\tau.D''_1\\ &+\ \tau.D''_{16}(hdtl,tltl)\\ &+\ \tau.\overline{Conf}\ \ DONTKNOW.D''_7\\ &+\ \tau.\overline{Conf}\ \ NOTOK.D''_{18}\\ &+\ \tau.D''_{17}(hd,tl)\end{aligned}$$

On remplace $D''_{17}(hd,tl)$ par $D''_{19}(hd,tl)$.

$$\begin{aligned}\simeq\ &\tau.D''_1\\ &+\ \tau.D''_{16}(hdtl,tltl)\\ &+\ \tau.\overline{Conf}\ \ DONTKNOW.D''_7\\ &+\ \tau.\overline{Conf}\ \ NOTOK.D''_{18}\\ &+\ \tau.D''_{19}(hd,tl)\end{aligned}$$

Appliquons la loi (L) pour supprimer la boucle τ dans $D''_{19}(hd,tl)$.

$$\begin{aligned}D''_{19}(hd,tl)\ \simeq\ &\tau.D''_1\\ &+\ \tau.D''_{16}(hdtl,tltl)\\ &+\ \tau.\overline{Conf}\ \ DONTKNOW.D''_7\\ &+\ \tau.\overline{Conf}\ \ NOTOK.D''_{18}\end{aligned}$$

Considérons $D''_8(hd,tl)$:

$$D''_8(hd,tl)\ \stackrel{def}{=}\ \tau.D''_{19}(hd,tl)\ +\ \tau.D''_{17}(hd,tl)$$

On remplace $D''_{17}(hd,tl)$ par $D''_{19}(hd,tl)$.

$$\simeq\ \tau.D''_{19}(hd,tl)\ +\ \tau.D''_{19}(hd,tl)$$

$$\simeq\ \tau.D''_{19}(hd,tl)$$

Comme toutes les parties de $D''_{19}(hd,tl)$ sont préfixées par τ:

$$D''_8(hd,tl)\ \simeq\ D''_{19}(hd,tl)$$

Considérons $D''_{13}(head, tail)$:

$$D''_{13}(head, tail) \stackrel{def}{=} \tau.D''_{19}(head, tail)$$

Or, toutes les parties de $D''_{19}(head, tail)$ sont préfixées par une action τ.

$$D''_{13}(head, tail) \simeq D''_{19}(head, tail)$$

Considérons $D''_{11}(head, tail)$:

$$\begin{aligned} D''_{11}(head, tail) \stackrel{def}{=} \tau^3.(&\tau^3.\overline{Conf}\ \ DONTKNOW.D''_{12} \\ &+ \tau^3.\overline{Conf}\ \ NOTOK.D''_{22} \\ &+ \tau^4.D''_{13}(head, tail) \\ &+ D''_{17}(head, tail)) \end{aligned}$$

$$\begin{aligned} \simeq\ &\tau.\overline{Conf}\ \ DONTKNOW.D''_{12} \\ &+ \tau.\overline{Conf}\ \ NOTOK.D''_{22} \\ &+ \tau.D''_{13}(head, tail) \\ &+ \tau.D''_{17}(head, tail) \end{aligned}$$

On remplace D''_{12} par D''_7, D''_{22} par D''_{18}, $D''_{13}(head, tail)$ par $D''_{19}(head, tail)$ et $D''_{17}(head, tail)$ par $D''_{19}(head, tail)$.

$$\begin{aligned} \simeq\ &\tau.\overline{Conf}\ \ DONTKNOW.D''_7 \\ &+ \tau.\overline{Conf}\ \ NOTOK.D''_{18} \\ &+ \tau.D''_{19}(head, tail) \\ &+ \tau.D''_{19}(head, tail) \end{aligned}$$

$$\begin{aligned} \simeq\ &\tau.\overline{Conf}\ \ DONTKNOW.D''_7 \\ &+ \tau.\overline{Conf}\ \ NOTOK.D''_{18} \\ &+ \tau.D''_{19}(head, tail) \end{aligned}$$

On remplace $D''_{19}(head, tail)$ par sa définition.

$$\begin{aligned} \simeq\ &\tau.\overline{Conf}\ \ DONTKNOW.D''_7 \\ &+ \tau.\overline{Conf}\ \ NOTOK.D''_{18} \\ &+ \tau.(\tau.D''_1 \\ &\quad + \tau.D''_{16}(hd, tl) \\ &\quad + \tau.\overline{Conf}\ \ DONTKNOW.D''_7 \\ &\quad + \tau.\overline{Conf}\ \ NOTOK.D''_{18}) \end{aligned}$$

En utilisant la loi (T2), on obtient:

$$\begin{aligned} \simeq\ &\tau.(\tau.D''_1 \\ &\quad + \tau.D''_{16}(hd, tl) \\ &\quad + \tau.\overline{Conf}\ \ DONTKNOW.D''_7 \\ &\quad + \tau.\overline{Conf}\ \ NOTOK.D''_{18}) \end{aligned}$$

Comme toutes les parties de $D''_{19}(head, tail)$ sont préfixées par τ:

$D''_{11}(head, tail) \simeq D''_{19}(head, tail)$

Reconsidérons $D''_{14}(hd, tl)$, maintenant on peut simplifier en:

$$
\begin{aligned}
D''_{14}(hd,tl) \simeq\ & \tau.\overline{Ind}\ hd\ LAST.D''_1 \\
& + \tau.\overline{Ind}\ hd\ INCOMPLETE.D''_{16}(hdtl, tltl) \\
& + \tau.\overline{Ind}\ hd\ LAST.\tau.\overline{Conf}\ \ DONTKNOW.D''_7 \\
& + \tau.\overline{Ind}\ hd\ INCOMPLETE.\tau.\overline{Conf}\ \ NOTOK.D''_{18} \\
& + \tau.\overline{Ind}\ hd\ LAST.D''_8(hd, tl) \\
& + \tau.\overline{Ind}\ hd\ INCOMPLETE.D''_8(hd, tl)
\end{aligned}
$$

On remplace $D''_8(hd, tl)$ par $D''_{19}(hd, tl)$.

$$
\begin{aligned}
\simeq\ & \tau.\overline{Ind}\ hd\ LAST.D''_1 \\
& + \tau.\overline{Ind}\ hd\ INCOMPLETE.D''_{16}(hdtl, tltl) \\
& + \tau.\overline{Ind}\ hd\ LAST.\tau.\overline{Conf}\ \ DONTKNOW.D''_7 \\
& + \tau.\overline{Ind}\ hd\ INCOMPLETE.\tau.\overline{Conf}\ \ NOTOK.D''_{18} \\
& + \tau.\overline{Ind}\ hd\ LAST.D''_{19}(hd, tl) \\
& + \tau.\overline{Ind}\ hd\ INCOMPLETE.D''_{19}(hd, tl)
\end{aligned}
$$

On remplace $D''_{19}(hd, tl)$ par sa définition.

$$
\begin{aligned}
\simeq\ & \tau.\overline{Ind}\ hd\ LAST.D''_1 \\
& + \tau.\overline{Ind}\ hd\ INCOMPLETE.D''_{16}(hdtl, tltl) \\
& + \tau.\overline{Ind}\ hd\ LAST.\tau.\overline{Conf}\ \ DONTKNOW.D''_7 \\
& + \tau.\overline{Ind}\ hd\ INCOMPLETE.\tau.\overline{Conf}\ \ NOTOK.D''_{18} \\
& + \tau.\overline{Ind}\ hd\ LAST.(... \\
& \qquad + \tau.\overline{Conf}\ \ DONTKNOW.D''_7 \\
& \qquad + ...) \\
& + \tau.\overline{Ind}\ hd\ INCOMPLETE.(... \\
& \qquad + \tau.\overline{Conf}\ \ NOTOK.D''_{18} \\
& \qquad + ...)
\end{aligned}
$$

La troisième et la quatrième ligne peuvent être obtenues en développant respectivement la cinquième et la sixième ligne. Elles sont alors supprimées.

$$
\begin{aligned}
\simeq\ & \tau.\overline{Ind}\ hd\ LAST.D''_1 \\
& + \tau.\overline{Ind}\ hd\ INCOMPLETE.D''_{16}(hdtl, tltl) \\
& + \tau.\overline{Ind}\ hd\ LAST.D''_{19}(hd, tl) \\
& + \tau.\overline{Ind}\ hd\ INCOMPLETE.D''_{19}(hd, tl)
\end{aligned}
$$

On remplace $D''_{19}(hd,tl)$ par sa définition:

$$\begin{aligned}
\simeq\ & \tau.\overline{Ind}\ hd\ LAST.D''_1 \\
& + \tau.\overline{Ind}\ hd\ INCOMPLETE.D''_{16}(hdtl,tltl) \\
& + \tau.\overline{Ind}\ hd\ LAST.(\tau.D''_1 \\
& \qquad\qquad + \tau.D''_{16}(hdtl,tltl) \\
& \qquad\qquad + \tau.\overline{Conf}\ DONTKNOW.D''_7 \\
& \qquad\qquad + \tau.\overline{Conf}\ NOTOK.D''_{18}) \\
& + \tau.\overline{Ind}\ hd\ INCOMPLETE.(\tau.D''_1 \\
& \qquad\qquad + \tau.D''_{16}(hdtl,tltl) \\
& \qquad\qquad + \tau.\overline{Conf}\ DONTKNOW.D''_7 \\
& \qquad\qquad + \tau.\overline{Conf}\ NOTOK.D''_{18})
\end{aligned}$$

La troisième et la quatrième ligne correspondent à des cas d'abort de la transmission.

$$\begin{aligned}
\simeq\ & \tau.\overline{Ind}\ hd\ LAST.D''_1 \\
& + \tau.\overline{Ind}\ hd\ INCOMPLETE.D''_{16}(hdtl,tltl) \\
& + \tau.\overline{Ind}\ hd\ LAST.\tau.\overline{Conf}\ DONTKNOW.D''_7 \\
& + \tau.\overline{Ind}\ hd\ INCOMPLETE.\tau.\overline{Conf}\ NOTOK.D''_{18}
\end{aligned}$$

$$\begin{aligned}
D''_{14}(hd,tl)\ \simeq\ & \tau.\overline{Ind}\ hd\ LAST.D''_1 \\
& + \tau.\overline{Ind}\ hd\ INCOMPLETE.D''_{16}(hdtl,tltl) \\
& + \tau.\overline{Ind}\ hd\ LAST.\overline{Conf}\ DONTKNOW.D''_7 \\
& + \tau.\overline{Ind}\ hd\ INCOMPLETE.\overline{Conf}\ NOTOK.D''_{18}
\end{aligned}$$

Considérons $D''_3(head,tail)$:

$$\begin{aligned}
D''_3(head,tail)\ \stackrel{def}{=}\ & \tau.\overline{Ind}\ head\ LAST.D''_4(tail) \\
& + \tau.\overline{Ind}\ head\ FIRST.D''_4(tail) \\
& + \tau^3.\overline{Ind}\ head\ LAST.D''_{11}(head,tail) \\
& + \tau^3.\overline{Ind}\ head\ FIRST.D''_{11}(head,tail)
\end{aligned}$$

$$\begin{aligned}
\simeq\ & \tau.\overline{Ind}\ head\ LAST.D''_4(tail) \\
& + \tau.\overline{Ind}\ head\ FIRST.D''_4(tail) \\
& + \tau.\overline{Ind}\ head\ LAST.D''_{11}(head,tail) \\
& + \tau.\overline{Ind}\ head\ FIRST.D''_{11}(head,tail)
\end{aligned}$$

On remplace $D''_4(tail)$ par sa définition et $D''_{11}(head,tail)$ par $D''_{19}(head,tail)$

$$\begin{aligned}
\simeq\ & \tau.\overline{Ind}\ head\ LAST.(\tau.D''_1\ +\ \tau.D''_{16}(hd,tl)) \\
& + \tau.\overline{Ind}\ head\ FIRST.(\tau.D''_1\ +\ \tau.D''_{16}(hd,tl)) \\
& + \tau.\overline{Ind}\ head\ LAST.D''_{19}(head,tail) \\
& + \tau.\overline{Ind}\ head\ FIRST.D''_{19}(head,tail)
\end{aligned}$$

S'il s'agit du dernier message du fichier, on va dans l'état D''_1. Sinon, on va dans l'état D''_{16}.

$$
\begin{aligned}
\simeq\ & \tau.\overline{Ind}\ head\ LAST.\tau.D''_1 \\
& +\ \tau.\overline{Ind}\ head\ FIRST.\tau.D''_{16}(hd,tl)) \\
& +\ \tau.\overline{Ind}\ head\ LAST.D''_{19}(head,tail) \\
& +\ \tau.\overline{Ind}\ head\ FIRST.D''_{19}(head,tail)
\end{aligned}
$$

$$
\begin{aligned}
\simeq\ & \tau.\overline{Ind}\ head\ LAST.D''_1 \\
& +\ \tau.\overline{Ind}\ head\ FIRST.D''_{16}(hd,tl)) \\
& +\ \tau.\overline{Ind}\ head\ LAST.D''_{19}(head,tail) \\
& +\ \tau.\overline{Ind}\ head\ FIRST.D''_{19}(head,tail)
\end{aligned}
$$

On remplace $D''_{19}(head,tail)$ par sa définition:

$$
\begin{aligned}
\simeq\ & \tau.\overline{Ind}\ head\ LAST.D''_1 \\
& +\ \tau.\overline{Ind}\ head\ FIRST.D''_{16}(hd,tl) \\
& +\ \tau.\overline{Ind}\ head\ LAST.(\tau.D''_1 \\
& \qquad\qquad\qquad +\ \tau.D''_{16}(hdtl,tltl) \\
& \qquad\qquad\qquad +\ \tau.\overline{Conf}\ DONTKNOW.D''_7 \\
& \qquad\qquad\qquad +\ \tau.\overline{Conf}\ NOTOK.D''_{18}) \\
& +\ \tau.\overline{Ind}\ head\ FIRST.(\tau.D''_1 \\
& \qquad\qquad\qquad +\ \tau.D''_{16}(hdtl,tltl) \\
& \qquad\qquad\qquad +\ \tau.\overline{Conf}\ DONTKNOW.D''_7 \\
& \qquad\qquad\qquad +\ \tau.\overline{Conf}\ NOTOK.D''_{18})
\end{aligned}
$$

La troisième et la quatrième ligne correspondent à des cas d'abort de la transmission.

$$
\begin{aligned}
\simeq\ & \tau.\overline{Ind}\ head\ LAST.D''_1 \\
& +\ \tau.\overline{Ind}\ head\ FIRST.D''_{16}(hd,tl) \\
& +\ \tau.\overline{Ind}\ head\ LAST.\tau.\overline{Conf}\ DONTKNOW.D''_7 \\
& +\ \tau.\overline{Ind}\ head\ FIRST.\tau.\overline{Conf}\ NOTOK.D''_{18}
\end{aligned}
$$

$$
\begin{aligned}
D''_3(head,tail)\ \simeq\ & \tau.\overline{Ind}\ head\ LAST.D''_1 \\
& +\ \tau.\overline{Ind}\ head\ FIRST.D''_{16}(hd,tl) \\
& +\ \tau.\overline{Ind}\ head\ LAST.\overline{Conf}\ DONTKNOW.D''_7 \\
& +\ \tau.\overline{Ind}\ head\ FIRST.\overline{Conf}\ NOTOK.D''_{18}
\end{aligned}
$$

Reconsidérons maintenant $D''_{16}(hd,tl)$:

$$
\begin{aligned}
D''_{16}(hd,tl)\ \simeq\ & \tau.\overline{Conf}\ DONTKNOW.D''_{18} \\
& +\ \tau.\overline{Conf}\ NOTOK.D''_{18} \\
& +\ \tau.D''_{14}(hd,tl)
\end{aligned}
$$

On remplace $D''_{14}(hd, tl)$ par sa définition.

$$\begin{aligned}
\simeq\ & \tau.\overline{Conf}\ \ DONTKNOW.D''_{18} \\
& + \ \tau.\overline{Conf}\ \ NOTOK.D''_{18} \\
& + \ \tau.(\tau.\overline{Ind}\ \ hd\ \ LAST.D''_1 \\
& \qquad + \ \tau.\overline{Ind}\ \ hd\ \ INCOMPLETE.D''_{16}(hdtl, tltl) \\
& \qquad + \ \tau.\overline{Ind}\ \ hd\ \ LAST.\overline{Conf}\ \ DONTKNOW.D''_7 \\
& \qquad + \ \tau.\overline{Ind}\ \ hd\ \ INCOMPLETE.\overline{Conf}\ \ NOTOK.D''_{18})
\end{aligned}$$

$$\begin{aligned}
D''_{16}(hd, tl) \ \simeq\ & \tau.\overline{Conf}\ \ DONTKNOW.D''_{18} \\
& + \ \tau.\overline{Conf}\ \ NOTOK.D''_{18} \\
& + \ \tau.\overline{Ind}\ \ hd\ \ LAST.D''_1 \\
& + \ \tau.\overline{Ind}\ \ hd\ \ INCOMPLETE.D''_{16}(hdtl, tltl) \\
& + \ \tau.\overline{Ind}\ \ hd\ \ LAST.\overline{Conf}\ \ DONTKNOW.D''_7 \\
& + \ \tau.\overline{Ind}\ \ hd\ \ INCOMPLETE.\overline{Conf}\ \ NOTOK.D''_{18}
\end{aligned}$$

Considérons $D''_2(head, tail)$:

$$D''_2(head, tail) \ \stackrel{def}{=}\ \tau^{10}.D''_3(head, tail)$$

$$D''_2(head, tail) \ \simeq\ \tau.D''_3(head, tail)$$

Comme toutes les parties de $D''_3(head, tail)$ sont préfixées par τ:

$$D''_2(head, tail) \ \simeq\ D''_3(head, tail)$$

Considérons $D''_{15}(head, tail)$:

$$D''_{15}(head, tail) \ \stackrel{def}{=}\ \tau.D''_2(head, tail)$$

On remplace $D''_2(head, tail)$ par $D''_3(head, tail)$.

$$\simeq\ \tau.D''_3(head, tail)$$

Comme toutes les parties de $D''_3(head, tail)$ sont préfixées par τ:

$$D''_{15}(head, tail) \ \simeq\ D''_3(head, tail)$$

Considérons maintenant D''_0:

$$\begin{aligned}
D''_0 \ \stackrel{def}{=}\ & \tau^7.(Req(Nil).D''_1 \ + \ Req(cons(head, tail)).D''_2(head, tail) \\
& \quad + \ Req(cons(head, tail)).\tau^4.D''_{15}(head, tail))
\end{aligned}$$

$$\begin{aligned}
\simeq\ & \tau.(Req(Nil).D''_1 \ + \ Req(cons(head, tail)).D''_2(head, tail) \\
& + \ Req(cons(head, tail)).D''_{15}(head, tail))
\end{aligned}$$

On remplace $D''_2(head, tail)$ et $D''_{15}(head, tail)$ par $D''_3(head, tail)$.

$$\begin{aligned}\simeq\ & \tau.(Req(Nil).D''_1 \ +\ Req(cons(head, tail)).D''_3(head, tail) \\ & +\ Req(cons(head, tail)).D''_3(head, tail))\end{aligned}$$

$$D''_0 \ \simeq\ \tau.(Req(Nil).D''_1 \ +\ Req(cons(head, tail)).D''_3(head, tail))$$

La dernière simplification consiste à remplacer D''_1, D''_7 et D''_{18} par leur définition dans les équations.

Reprenons D''_0:

$$D''_0 \ \simeq\ \tau.(Req(Nil).D''_1 \ +\ Req(cons(head, tail)).D''_3(head, tail))$$

$$\simeq\ \tau.(Req(Nil).\tau.\overline{Conf}\ OK.D''_0 \ +\ Req(cons(head, tail)).D''_3(head, tail))$$

$$D''_0 \ \simeq\ \tau.(Req(Nil).\overline{Conf}\ OK.D''_0 \ +\ Req(cons(head, tail)).D''_3(head, tail))$$

Reprenons $D''_3(head, tail)$:

$$\begin{aligned}D''_3(head, tail) \ \simeq\ & \tau.\overline{Ind}\ head\ LAST.D''_1 \\ & +\ \tau.\overline{Ind}\ head\ FIRST.D''_{16}(hd, tl) \\ & +\ \tau.\overline{Ind}\ head\ LAST.\overline{Conf}\ DONTKNOW.D''_7 \\ & +\ \tau.\overline{Ind}\ head\ FIRST.\overline{Conf}\ NOTOK.D''_{18}\end{aligned}$$

$$\begin{aligned}\simeq\ & \tau.\overline{Ind}\ head\ LAST.\tau.\overline{Conf}\ OK.D''_0 \\ & +\ \tau.\overline{Ind}\ head\ FIRST.D''_{16}(hd, tl) \\ & +\ \tau.\overline{Ind}\ head\ LAST.\overline{Conf}\ DONTKNOW.\tau.D''_0 \\ & +\ \tau.\overline{Ind}\ head\ FIRST.\overline{Conf}\ NOTOK.\tau.\overline{Ind\_err}.D''_0\end{aligned}$$

$$\begin{aligned}D''_3(head, tail) \ \simeq\ & \tau.\overline{Ind}\ head\ LAST.\overline{Conf}\ OK.D''_0 \\ & +\ \tau.\overline{Ind}\ head\ FIRST.D''_{16}(hd, tl) \\ & +\ \tau.\overline{Ind}\ head\ LAST.\overline{Conf}\ DONTKNOW.D''_0 \\ & +\ \tau.\overline{Ind}\ head\ FIRST.\overline{Conf}\ NOTOK.\overline{Ind\_err}.D''_0\end{aligned}$$

Reprenons $D''_{16}(hd, tl)$:

$$\begin{aligned}D''_{16}(hd, tl) \ \simeq\ & \tau.\overline{Conf}\ DONTKNOW.D''_{18} \\ & +\ \tau.\overline{Conf}\ NOTOK.D''_{18} \\ & +\ \tau.\overline{Ind}\ hd\ LAST.D''_1 \\ & +\ \tau.\overline{Ind}\ hd\ INCOMPLETE.D''_{16}(hdtl, tltl) \\ & +\ \tau.\overline{Ind}\ hd\ LAST.\overline{Conf}\ DONTKNOW.D''_7 \\ & +\ \tau.\overline{Ind}\ hd\ INCOMPLETE.\overline{Conf}\ NOTOK.D''_{18}\end{aligned}$$

$$
\begin{aligned}
\simeq\ & \tau.\overline{Conf}\ \ DONTKNOW.\tau.\overline{Ind\_err}.D''_0 \\
& +\ \tau.\overline{Conf}\ \ NOTOK.\tau.\overline{Ind\_err}.D''_0 \\
& +\ \tau.\overline{Ind}\ \ hd\ \ LAST.\tau.\overline{Conf}\ \ OK.D''_0 \\
& +\ \tau.\overline{Ind}\ \ hd\ \ INCOMPLETE.D''_{16}(hdtl, tltl) \\
& +\ \tau.\overline{Ind}\ \ hd\ \ LAST.\overline{Conf}\ \ DONTKNOW.\tau.D''_0 \\
& +\ \tau.\overline{Ind}\ \ hd\ \ INCOMPLETE.\overline{Conf}\ \ NOTOK.\tau.\overline{Ind\_err}.D''_0
\end{aligned}
$$

$$
\begin{aligned}
D''_{16}(hd, tl)\ \simeq\ & \tau.\overline{Conf}\ \ DONTKNOW.\overline{Ind\_err}.D''_0 \\
& +\ \tau.\overline{Conf}\ \ NOTOK.\overline{Ind\_err}.D''_0 \\
& +\ \tau.\overline{Ind}\ \ hd\ \ LAST.\overline{Conf}\ \ DONTKNOW.D''_0 \\
& +\ \tau.\overline{Ind}\ \ hd\ \ INCOMPLETE.\overline{Conf}\ \ NOTOK.\overline{Ind\_err}.D''_0 \\
& +\ \tau.\overline{Ind}\ \ hd\ \ LAST.\overline{Conf}\ \ OK.D''_0 \\
& +\ \tau.\overline{Ind}\ \ hd\ \ INCOMPLETE.D''_{16}(hdtl, tltl)
\end{aligned}
$$

Si on récapitule, alors on obtient les 3 équations suivantes:

$$
D''_0\ \simeq\ \tau.(Req(Nil).\overline{Conf}\ \ OK.D''_0\ +\ Req(cons(head, tail)).D''_3(head, tail))
$$

$$
\begin{aligned}
D''_3(head, tail)\ \simeq\ & \tau.\overline{Ind}\ \ head\ \ LAST.\overline{Conf}\ \ OK.D''_0 \\
& +\ \tau.\overline{Ind}\ \ head\ \ FIRST.D''_{16}(hd, tl) \\
& +\ \tau.\overline{Ind}\ \ head\ \ LAST.\overline{Conf}\ \ DONTKNOW.D''_0 \\
& +\ \tau.\overline{Ind}\ \ head\ \ FIRST.\overline{Conf}\ \ NOTOK.\overline{Ind\_err}.D''_0
\end{aligned}
$$

$$
\begin{aligned}
D''_{16}(hd, tl)\ \simeq\ & \tau.\overline{Conf}\ \ DONTKNOW.\overline{Ind\_err}.D''_0 \\
& +\ \tau.\overline{Conf}\ \ NOTOK.\overline{Ind\_err}.D''_0 \\
& +\ \tau.\overline{Ind}\ \ hd\ \ LAST.\overline{Conf}\ \ DONTKNOW.D''_0 \\
& +\ \tau.\overline{Ind}\ \ hd\ \ INCOMPLETE.\overline{Conf}\ \ NOTOK.\overline{Ind\_err}.D''_0 \\
& +\ \tau.\overline{Ind}\ \ hd\ \ LAST.\overline{Conf}\ \ OK.D''_0 \\
& +\ \tau.\overline{Ind}\ \ hd\ \ INCOMPLETE.D''_{16}(hdtl, tltl)
\end{aligned}
$$

On peut observer que D''_0 se produit toujours immédiatement après un préfixe τ dans les expressions ci-dessus. Ainsi, l'ensemble des expressions suivant:

$\widetilde{D'''}\ =\ <\ \tau.D''_0, D''_3(head, tail), D''_{16}(hd, tl)\ >$ satisfait les équations $\widetilde{D'''}\ \simeq\ \tilde{F}(\widetilde{D'''})$ (i.e, satisfait les équations $F_i(\tilde{S})$, $i = 0\ ..\ 2$ de la spécification abstraite du protocole).

Rappelons que $\tilde{S} \stackrel{def}{=} \tilde{F}(\tilde{S})$. On peut conclure que $\widetilde{D'''} \simeq \tilde{S}$ et en particulier que $D''_0 \simeq S_0$.

Or, on sait déjà que $D''_0 \simeq D'_0 \simeq D_0 \sim System$. On a donc prouvé que $System \simeq S_0$ (i.e, que le protocole et la spécification abstraite ont le même comportement observable).

8 Conclusions

8.1 Travaux précédents

Le BRP a fait l'objet de plusieurs travaux dans différents formalismes:

- **K.Havelund** et **N.Shankar** [HS96] ont combiné les techniques de model checking et de theorem proving pour faire la preuve du BRP. Ils ont d'abord analysé une version réduite du BRP (système à états finis) en utilisant Murϕ [MDN93], un outil d'exploration d'états. Après, ils ont traduit cette description dans PVS [ORS95] et généralisé le résultat de la traduction à la version complète du BRP (système à états infinis). La preuve de cette version complète dans PVS a nécessité un effort de trois homme-mois. Ensuite, à partir de cette spécification complète, ils ont déduit une abstraction (système à états finis) préservant certaines propriétés essentielles du protocole. L'abstraction dans leur cas consistait à borner les ressources non bornées du protocole à savoir: la donnée à transmettre, le nombre de tentatives de retransmission et la taille du fichier. Les propriétés qui doivent être préservées ont été prouvées dans PVS. Finalement, ils ont utilisé les model checkers SMV [MCM93], Murϕ et une extension de PVS avec le μ-calcul modal [Jan93] pour faire le model checking final de l'abstraction du protocole.

  Cependant, d'une part, leur description du protocole est trop détaillée (11 pages). La description du protocole dans le π-calcul a l'avantage d'être compacte et complètement formelle (2 pages seulement). De plus, le critère de correction dans le π-calcul est hautement plus informatif puisque le protocole est prouvé équivalent à la spécification représentant son comportement extérieur.

  D'autre part, la difficulté de l'application de leur approche pour prouver un protocole de communication quelconque réside dans la recherche de l'abstraction; aucune technique n'est fournie pour systématiser cette recherche. Ainsi, trouver l'abstraction du Sliding Window Protocol est un vrai challenge.

- Le travail de **J.F.Groote** et **J.V.De Pol** [GP96] est le plus proche de mon travail. Ils ont utilisé comme support formel μCRL, une combinaison d'algèbre de processus et de types de données abstraits. Le protocole et son comportement extérieur ont été spécifiés dans μCRL. La correspondance entre les deux a été démontrée manuellement en utilisant la théorie algébrique de μCRL: la branching bisimulation, une notion forte de la bisimulation faible (aucune différence n'est observable entre 2 processus équivalents). Ils ont également utilisé le principe de spécification récursive qui dit qu'un système d'équations récursives gardées possède au plus une solution.

  La preuve a été faite ensuite dans Coq [Coq95]. Pour cela, ils ont codé la syntaxe, les axiomes et les règles de μCRL dans Coq. Mais, le principe de spécification récursive n'a pas été utilisé; le système d'équations récursives a été codé par une seule équation dans Coq. De plus, les sortes booléens, entiers et listes ont été codées par des ensembles inductifs au lieu des types de données abstraits et les fonctions sur ces sortes ont été définies avec la récursion primitive.

En conclusion, leur spécification du protocole dans μCRL est compacte. Mais, la preuve dans Coq a nécessité un encodage très détaillé et la description Coq résultante est très longue.

- **Helmink**, **Sellink** et **Vaandrager** [HSV94] ont analysé le BRP dans les I/O automata [LT87], automates qui distinguent les entrées, les sorties et les actions internes et qui autorisent toutes les entrées possibles au niveau de chaque état. Ils ont prouvé l'absence de deadlock dans le BRP. La partie sûreté des preuves a été faite dans Coq et ils ont prouvé plusieurs propriétés d'invariance. Mais, leur résultat principal est leur critère de correction du protocole, qui est un argument de raffinement (en termes d'inclusion de traces) montrant qu'une spécification en I/O automata implémente une autre plus abstraite.

- Un travail récent a été réalisé par **D'Argenio**, **Katoen**, **Ruys** et **Tretmans** [DKRT97] pour prouver la version temporisée du BRP. Ils ont spécifié le service de transfert de fichier par des propriétés fournissant les relations entre les entrées et les sorties. Cette spécification de service a été validée par rapport à la spécification algébrique dans μCRL du comportement extérieur du BRP fournie dans [GP96]. Le protocole est modélisé par un réseau d'automates temporisés communiquant par canal à la CCS. Un automate temporisé est un automate classique équipé avec des variables d'horloge et des invariants d'état. La correction du protocole a été vérifiée en prouvant qu'il satisfait certaines propriétés temps réel exprimées sous forme de formules logiques dans UPPAAL [BLL96]. La correction du protocole, sans considérer les aspects temps réel, a été prouvée dans SPIN sachant que le service a été spécifié dans PROMELA [Hol91].

  Cependant, UPPAAL ne supporte pas la synchronisation entre processus avec passage de message et fournit uniquement les types horloge et integer. Pour ces raisons, des adaptations ont été nécessaires. Par exemple, la synchronisation entre deux processus avec passage de message a été modélisée par des variables partagées. De plus, les propriétés de la spécification du service ne sont pas des invariants et il leur était difficile de les exprimer dans UPPAAL.

- **J.R.Abrial** [Abr96a] a développé le BRP par raffinements successifs dans B [Abr96b]. Le protocole est spécifié d'abord sans la notion de temps en concentrant le comportement attendu dans la dernière étape de raffinement. En effet, la spécification est décrite par un invariant exprimant simplement que le fichier reçu côté récepteur est un préfixe du fichier transmis par l'émetteur. Ensuite, le protocole est raffiné en considérant ses différentes étapes. Les propriétés d'absence de deadlock et de terminaison du protocole ont été prouvées dans B.

  Cependant, la difficulté de ce travail réside d'une part dans la recherche des raffinements adéquats qui n'est pas triviale. D'autre part, la méthode est inapplicable dans le cadre de développement d'applications complexes nécessitant l'intervention de plusieurs personnes; une décomposition de l'application est alors indispensable.

8.2 La preuve par bisimulation

Le seul point qui peut paraître négatif dans l'approche adoptée dans ce papier est que la preuve par bisimulation de systèmes réels nécessite une analyse exhaustive et fastidieuse de tous les cas possibles. Ceci rend la preuve très difficile à gérer.

Cependant, l'approche possède plusieurs avantages:

- La description du système est compacte et complètement formelle.
- L'analyse exhaustive des cas possibles permet d'une part une vraie compréhension du système et d'autre part de détecter les erreurs d'implémentation.
- L'approche est modulaire: on ne considère jamais toutes les composantes du système en même temps.
- Le critère de correction est hautement informatif puisque le protocole est prouvé équivalent à la spécification représentant le comportement extérieur du protocole.
- La méthode est conduite par un calcul simple.
- La méthode est mécanisable car d'une part le calcul est simple; d'autre part, la vérification d'une bisimulation est purement procédurale. Plusieurs programmes de recherche de bisimulation (quand elle existe) ont été écrits, au moins pour les systèmes à états finis.

8.3 Perspectives

Il est clair que pour un protocole de taille raisonnable tel que le BRP, la preuve est énorme et il est très difficile de la gérer et de la maintenir, notamment en cas de changement de l'implémentation suite à la détection d'une erreur lors de l'analyse.

Un travail futur serait d'élaborer une métodologie générale de conception de protocoles de communication permettant de réduire les efforts à produire pour faire la preuve d'un protocole par bisimulation dans le π-calcul.

Plusieurs méthodologies de conception ont été étudiées dans le passé, parmi lesquelles:

1. la décomposition modulaire d'un protocole: le protocole est décomposé en unités fonctionnelles séquentielles s'exécutant en parallèle [LM87].
2. la décomposition en phases d'un protocole: le protocole est décomposé en phases (par exemple, les phases d'ouverture de connexion, de transfert de données et de fermeture de connexion) parallèlles s'exécutant de manière séquentielle [CGL85], [SDR89].
3. le raffinement: il existe deux types de raffinement:
   - le raffinement d'actions: une action est remplacée par un comportement composé de plusieurs actions. Deux systèmes, dont l'un est le raffinement de l'autre, sont

prouvés équivalents. Mais la restriction est que ces deux systèmes doivent communiquer avec l'environnement via le même ensemble d'actions [AH89], [Bac89], [Lam90], [JPZ91], [Ren93].

- le raffinement d'interfaces: cette approche permet de pallier à la restriction ci-dessus. L'interface entre deux composantes est décrite de manière abstraite pendant la phase de développement; les détails non importants sont alors ignorés. Les actions de communication (i.e, de l'interface) sont ensuite raffinées et la composition parallèle des composantes après le raffinement est prouvée équivalente à leur composition avant le raffinement [BJO91].

Le travail consisterait donc à étudier ces différentes approches et à choisir celle qui est la mieux adaptée au π-calcul.

Les preuves ont jusqu'à présent été effectuées à la main, il est également important d'en automatiser au moins une partie.

Un autre travail serait d'étudier d'autres propriétés des protocoles de communication telles que les propriétés de vivacité; par exemple, la terminaison et l'absence de livelock (cycles).

Références

[Abr96a] Specification and Design of the Bounded Retransmission Protocol, J-R.Abrial, Draft 1, 1996.

[Abr96b] The B-Book, J-R.Abrial, Cambridge University Press, 1996.

[AH89] Towards Action Refinement in Process Algebras, L.Aceto, M.Hennessy, In Proc. 4th IEEE Int. Symp. on Logic in Computer Science, p138-145, 1989.

[AHS96] Hybrid Systems III, R.Alur, T.Henzinger, E.D.Sontag, LNCS 1066, Springer-Verlag, 1996.

[Bac89] A Method for Refining Atomicity in Parallel Algorithms, R.J.R.Back, In Proc. PARLE 89, LNCS 365, p199-216, 1989.

[BJO91] Refining Interfaces of Communicating Systems, E.Brinksma, B.Jonsson, F.Orava, this paper is a revised and extended version of a paper that has appeared under the same title in Ambramsky and Maibaum, editors, Proc. Coll. on Combining Paradigms in Software Development, Brighton, LNCS 494, 1991.

[BLL96] UPPAAL. A tool suite for the automatic verification of real-time systems, J.Bengtsson, K.G.Larsen, F.Larsson, P.Pettersson, W.Yi, In [AHS96], p232-243.

[CGL85] A Discipline for Constructing Muti-phase Communication Protocols, C.H.Chow, M.G.Gouda, S.S.Lam, ACM Trans. on Computer Science, 3(4):315-343, November 1985.

[Coq95] The Coq Proof Assistant Reference Manual version 5.10, C.Cornes, J.Courant, J.C.Filliatre, G.Huet, P.Manoury, C.Paulin-Mohring, C.Munoz, C.Murthy, C.Parent, A.Saibi and B.Werner, Technical Report, INRIA Rocquencourt, France, February 1995.

[DKRT97] The Bounded Retransmission Protocol must be on Time!, P.R.D'Argenio, J.P.Katoen, T.C.Ruys, J.Tretmans, TACAS'97.

[GP96] A Bounded Retransmission Protocol for Large Data Packets, J.F. Groote, J.Van de Pol, CAV'96, LNCS 1101, 1996.

[Hol91] Design and Validation of Computer Protocols, G.J.Holzmann, Prentice-Hall, 1991.

[HS96] Experiments in Theorem Proving and Model Checking for Protocol Verification, K.Havelund, N.Shankar, In Proceeding of FME, March 1996, Oxford.

[HSV94] Proof checking a data link protocol, L.Helmink, M.P.A.Sellink, F.W.Vaandrager, In H.Barandregt and T.Nipkow, editors, Types for proofs and programs, LNCS 806, p127-165, Springer-Verlag, 1994.

[Jan93] ROBDD Software, G.Janssen, Department of Electrical Engineering, Eindhoven University of Technology, October 1993.

[JPZ91] Action Systems and Action Refinement in the Development of Parallel Systems: An Algebraic Approach, W.Janssen, M.Poel, J.Zwiers, CONCUR'91, LNCS 527, 1991.

[Lam90] A Theorem on Atomicity in Distributed Algorithms, L.Lamport, Distributed Computing, 4(2):59-68, 1990.

[LM87] A Complete Protocol Verification Using Relativized Bisimulation, K.Larsen, R.Milner, In Proceeding 14th Colloquium on Automata, Languages and Programming, LNCS 267, Springer-Verlag, 1987.

[LT87] Hierarchical Correctness Proofs for Distributed Algorithms, N.A.Lynch, M.R.Tuttle, In Proceeding of the 6th Annual Symposium on Principles of Distributed Computing, New York, p137-151, ACM Press, 1987.

[MCM93] Symbolic Model ChecKing, K.L.McMillan, Kluwer Academic Publishers, Boston, 1993.

[MDN93] Murphi Annotated Reference Manual, version 2.6, R.Melton, D.L.Dill, C.Norris Ip., Technical Report, Stanford University, Palo Alto, California, USA, November 1993.

[Mil89] Communication and Concurrency, R.Milner, Prentice-Hall, 1989.

[Mil91] The polyadic π-calculus: a tutorial, R.Milner, LFCS, technical report ECS-LFCS-91-180, October 1991.

[MPW89a] A calculus of mobile processes, Part 1, R.Milner, J.Parrow, D.Walker, LFCS, technical report ECS-LFCS-89-85, June 1989.

[MPW89b] A calculus of mobile processes, Part2, R.Milner, J.Parrow, D.Walker, LFCS, technical report ECS-LFCS-89-86, June 1989.

[OP92] An Algebraic Verification of a Mobile Network, F.Orava, , J.Parrow, Formal Aspects of Computing, 4(6):497-543, 1992.

[ORS95] Formal Verification For Fault-tolerant Architectures: Prolegomena to the Design of PVS, S.Owre, J.Rushby, N.Shankar, F.von Henke, IEEE Transactions on Software Engineering, 21(2):107-125, February 1995.

[Ren93] Models and Methods for Action Refinement, A.Rensink, PhD thesis, University of Twente, 1993.

[SDR89] Designing Distributed Algorithms by means of Formal Sequentially Phased Reasoning, F.A.Stomp, W.P.De Roever, 3th Workshop on Distributed Algorithms, LNCS 392, 1989.