ALEXANDRE LACASSE

Approche algébrique pour la prévention d'intrusions

Mémoire présenté
à la Faculté des études supérieures de l'Université Laval
dans le cadre du programme de maîtrise en informatique
pour l'obtention du grade de Maître ès Sciences (M. Sc.)

FACULTÉ DES SCIENCES ET DE GÉNIE
UNIVERSITÉ LAVAL
QUÉBEC

Février 2006

Résumé

Dans ce travail, nous définissons une nouvelle algèbre de processus basée sur CCS. Cette algèbre, qui est destinée à la sécurisation formelle de réseaux, est munie d'un opérateur de surveillance qui permet de contrôler les entrées et les sorties d'un processus, ou d'un sous-processus, à l'image d'un pare-feu dans un réseau informatique. L'algèbre permet donc de modéliser des réseaux munis de moniteurs, et également, n'importe quel système communicant devant être contrôlé par des moniteurs. Avant d'entrer dans le vif du sujet, nous débutons par une revue des approches globales en détection d'intrusions, soient l'approche comportementale et l'approche par scénarios. Nous parcourons par la suite différents langages formels destinés à la modélisation de systèmes communicants, en prêtant une attention particulière aux algèbres de processus.

Avant-propos

Je tiens à remercier mon directeur de recherche, le professeur Mohamed Mejri, premièrement pour m'avoir accepter dans ce projet qui rejoignait deux domaines de recherche dans lesquels je n'avais pratiquement aucune connaissance, et deuxièmement pour le temps qu'il m'a consacré durant ces deux années. Je tiens également à exprimer ma reconnaissance envers les professeurs Béchir Ktari et Kamel Adi, qui ont accepté d'évaluer ce mémoire.

Ce mémoire est le fruit d'un projet de maîtrise qui fut réalisé dans le cadre d'un programme de coopération franco-québécois ; je remercie les gens qui ont permis la concrétisation de ce projet, que ce soit par leur aide financière ou leur implication directe. Je remercie donc le Ministère des Relations Internationales du Québec ainsi que le Gouvernement Français pour le financement de ce projet. Je remercie également les professeurs André-Luc Beylot, Marc Boyer et Jérôme Ermont pour leur accueil à l'ENSEEIHT, ainsi que le personnel d'Égide, qui a grandement facilité notre intégration dans la ville de Toulouse.

Merci à Mathieu Couture, camarade et collègue durant les deux voyages en France, pour m'avoir invité à participer à ce projet, et pour ta motivation sans borne.

Merci à Gabrielle, pour un peu tout, pour avoir dépouillé ce mémoire d'une partie de ses fautes, mais surtout pour les salades variées que tu préparais les midis à Toulouse.

Table des matières

Liste des tableaux

Table des figures

Introduction

Motivation

La protection des réseaux informatiques est aujourd'hui un enjeu crucial, autant pour la confidentialité des informations qu'ils peuvent renfermer que pour simplement maintenir leur bon fonctionnement ; un réseau truffé de logiciels indésirables (espions, virus, vers) peut voir son trafic ralenti à un point inacceptable, et certains ports négligemment ou malicieusement ouverts, tout comme certains logiciels de transfert utilisant un protocole non sécuritaire, peuvent devenir une porte d'entrée à un intrus qui pourra alors s'approprier des données protégées. Bien que l'on n'ait guère besoin de justifier davantage la nécessité de fournir des efforts tangibles pour permettre la configuration de réseaux sécuritaires, le lecteur étant sûrement déjà au fait de cette réalité contemporaine, il importe cependant de noter que cet objectif est plus difficile à réaliser qu'il peut sembler à première vue.

À travers les deux dernières décennies, de nombreux travaux ont été réalisés et de nombreux outils (matériels ou logiciels) ont été développés dans le but de rendre les réseaux plus sécuritaires ; on parle ici principalement de pare-feu (*firewalls*) et de systèmes de détection d'intrusions (ou IDS, de l'anglais *Intrusion detection system*) qui ont les rôles suivants : les pare-feu contrôlent le trafic sur un réseau en détruisant des paquets indésirables, alors que les IDS servent à collecter des informations (sur le trafic du réseau ou l'état des machines) de sorte à prévenir certaines attaques potentielles.

Nous disposons donc de plusieurs outils pour sécuriser nos réseaux. Cependant, l'instauration de pare-feu ou d'IDS ne peut se faire de manière entièrement *ad hoc* dans des réseaux pouvant posséder, non seulement plusieurs machines, mais également plusieurs sous-réseaux, chacun possédant ses propres politiques de sécurité. En effet, si la topologie d'un réseau est trop complexe, il peut devenir impossible, sans une analyse approfondie, d'implanter une politique donnée ; l'un des problèmes étant qu'il faut s'assurer que les politiques de sécurité des différents sous-réseaux ne sont pas incom-

patibles. D'où l'intérêt de chercher à développer des outils formels pour modéliser les réseaux, ainsi que les politiques de sécurités, sous la forme d'un langage mathématique. Ces outils nous permettraient de valider sans ambiguïté l'implantation d'une politique de sécurité dans un réseau, et possiblement également de générer automatiquement une configuration sécurisée d'un réseau à partir d'une version non sécurisée du réseau et d'une politique de sécurité donnée.

Objectifs et méthodologie

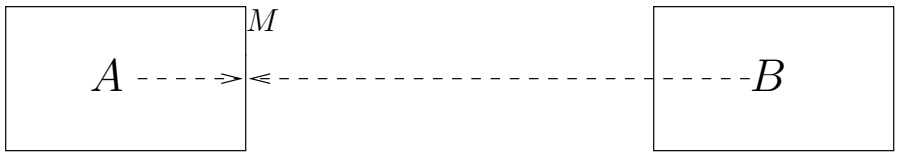
Le travail présenté dans ce mémoire s'inscrit dans le cadre d'un projet trop ambitieux pour être entièrement étudié à l'intérieur d'une maîtrise, l'objectif de ce projet étant de concevoir une méthode formelle, automatique et vérifiable pour configurer de façon sécuritaire des réseaux informatiques selon une politique de sécurité. Ce projet peut se découper en trois étapes :

1. développer une méthode algébrique pour modéliser formellement un réseau informatique ;
2. développer un langage de spécification adéquat pour l'écriture des politiques de sécurité ;
3. automatiser la génération d'un réseau sécurisé, c'est-à-dire respectant une politique de sécurité préalablement spécifiée, par ajout de moniteurs sur un réseau donné.

Dans ce présent travail, nous nous sommes concentrés sur la première de ces trois étapes, c'est-à-dire la conception d'une méthode algébrique dédiée à la modélisation de réseaux monitorés. Pour ce faire, nous avons décidé d'explorer les algèbres de processus ; nous avons alors choisi de nous baser sur CCS, que nous avons quelque peu modifié, principalement en lui ajoutant un nouvel opérateur, appelé opérateur de surveillance, qui permet de surveiller les communications entre les différents sous-processus. Ce nouvel opérateur permet donc d'implanter directement dans les processus, des moniteurs qui jouent un rôle équivalent aux pare-feu dans un réseau.

Comme nous voulons que nos modèles reflètent le fonctionnement d'un réseau, le rôle précis des moniteurs est un facteur important. Ceux-ci ne pourraient pas, par exemple, avoir pour comportement d'arrêter le déroulement d'un processus si quelque chose de mauvais se présentait, en effet, ceci correspondrait à un déni de service. Nous nous sommes donc inspirés des pare-feu pour définir le comportement des moniteurs : ils

FIG. 1 – Comportement des moniteurs



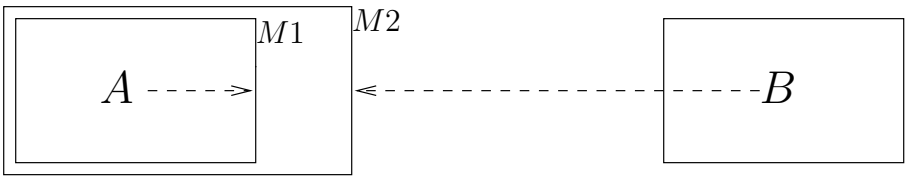
auront pour rôle d'absorber (de détruire) les messages indésirables voulant passer d'un processus à un autre. Nous avons alors créé une sémantique pour l'opérateur de surveillance allant en ce sens. Ainsi, lors d'une communication entre deux processus A et B, si un message partant du processus A est bloqué par le moniteur surveillant ce processus, le processus B n'aura aucune connaissance qu'une tentative d'envoi a été faite, et vice-versa si c'est le processus B qui envoi vers le processus A un message qui sera bloqué par le moniteur de ce dernier. La figure 1 illustre le fonctionnement des moniteurs lors de l'envoi d'un paquet indésirable selon que le moniteur est placé du côté de l'émetteur ou de celui du récepteur du paquet.

Emboîtement des pare-feu

Lorsque nous sommes en présence de plusieurs pare-feu emboîtés (voir la figure 2), le comportement d'une communication bloquée diffère quelque peu selon que cette communication est bloquée au niveau du processus émetteur ou au niveau du processus récepteur. En fait, la différence concerne le pare-feu en action. Supposons par exemple que les pare-feu $M1$ et $M2$ de la figure 2 bloquent tout paquet entrant ou sortant du réseau A. Un paquet émis par A et destiné à B sera bloqué par le pare-feu $M1$, alors qu'un paquet émis par B et destiné à A sera lui bloqué par le pare-feu $M2$.

Dans un premier temps, nous proposerons une algèbre simplifiée, dans laquelle les pare-feu seront de type *stateless firewall*, c'est-à-dire des pare-feu qui contrôlent chaque paquet de façon indépendante, par opposition aux pare-feu dits *statefull firewalls*, qui recherchent en plus dans la trace des séquences logiques de paquets qui peuvent correspondre à une attaque élaborée. Cette version de l'algèbre n'est pas concernée par les propos de cette section, puisque ses moniteurs auront un comportement fixe qui ne pourra évoluer. Cependant, une seconde version de l'algèbre permettra de définir des moniteurs pouvant évoluer de sorte à suivre une séquence d'événements, il sera alors important que dans le cas de moniteurs emboîtés, ce soit le bon moniteur qui évolue, conformément aux considérations liées à l'exemple de la figure 2.

FIG. 2 – Deux réseaux munis de pare-feu



Organisation

Dans la première partie de ce mémoire, nous présentons une revue de la littérature sur deux sujets liés à nos travaux : la détection d'intrusions et les méthodes formelles. Dans le chapitre 1, nous dressons un état de l'art des approches globales applicables à la détection d'intrusions, soient l'approche comportementale et l'approche par scénarios. L'analyse des outils employés en détection d'intrusions nous permettra d'avoir une vue plus détaillée des moyens disponibles pour sécuriser un réseau. Le chapitre 2 traitera pour sa part des méthodes formelles. Plus précisément, nous présentons dans ce chapitre différents langages permettant de modéliser des systèmes communicants, dont plusieurs algèbres de processus. Ce deuxième chapitre se termine par une classification des propriétés de sécurité, suivie d'une introduction à l'écriture formelle des propriétés de sécurité à l'aide de logique.

La seconde partie de ce mémoire, correspondant au chapitre 3, présente l'algèbre de processus que nous avons développée. Cette algèbre, qui se base sur CCS, possède un opérateur (appelé opérateur de surveillance) qui permet de modéliser des systèmes surveillés par des pare-feu. Le comportement de ces moniteurs se fonde sur l'analyse faite dans cette introduction, c'est-à-dire qu'il s'inspire du comportement d'un vrai pare-feu sur un réseau informatique. Cette algèbre a été développée avec pour objectif de l'appliquer à la sécurisation formelle de réseaux informatiques, il va de soi cependant qu'elle peut également être appliquée à la modélisation de n'importe quel système demandant d'être surveillé par des moniteurs. Le chapitre comporte, en plus de la définition de l'algèbre, plusieurs exemples de modélisation de petits réseaux, ainsi que quelques démonstrations de propositions concernant une équivalence de processus que nous avons définie pour comparer les processus de notre algèbre.

Chapitre 1

Détection d'intrusions

Dans ce premier chapitre de l'état de l'art, nous présentons le problème de la détection d'intrusions en informatique. Suite à une présentation de la problématique et des enjeux, nous aborderons les différentes approches pour attaquer le problème ; on parle alors principalement des deux grandes approches : l'approche comportementale et l'approche par scénarios. Puis, nous parlerons brièvement de quelques outils qui ont été développés dans le but de faire de la détection d'intrusions efficace et rigoureuse ; plus précisément, nous traiterons des outils STAT et IDIOT.

1.1 Présentation du problème

Puisque tout système informatique possède des failles potentielles, les probabilités augmentant avec la complexité grandissante des systèmes et le nombre d'usagers, la détection d'intrusions devient un enjeu majeur pour l'intégrité des systèmes.

On entend par intrusion, une tentative délibérée soit d'obtenir des informations confidentielles, soit d'exploiter malicieusement les ressources d'un système, soit simplement de nuire au bon fonctionnement d'un système, ce qui peut mener par exemple à un déni de service.

La détection d'intrusions a donc pour objectif d'identifier une tentative d'intrusion de sorte à pouvoir réagir en conséquence, si cela est possible, avant que le mal ne soit fait.

De nombreux travaux ont été effectués (particulièrement dans les 20 dernières années) sur le sujet, à titre indicatif, l'ouvrage de Mé et Michel (voir [44]), qui consiste en une bibliographie regroupant les travaux dans le domaine, contient 55 pages. Cet état de l'art ne visera donc pas à être complet, mais simplement à présenter les différentes approches à la base des modèles de détection d'intrusions, particulièrement l'approche comportementale et l'approche par scénarios. Nous aborderons également quelques outils, appelés systèmes de détection d'intrusions (ou IDS de l'anglais *Intrusion Detection System*), jugés plus en lien avec la suite de ce document.

Il faut rappeler que dans ce document, on ne vise pas directement la détection d'intrusions, mais plutôt la sécurisation formelle de réseaux par implantation de moniteurs spécialisés. Cependant, un moniteur peut être vu comme un outil de détection d'intrusions réagissant de manière précise face à une situation représentant une attaque potentielle. Il va donc de soi que la détection d'intrusions est très liée à notre problème, d'où l'importance d'y consacrer une attention particulière.

1.2 Le système d'audit

Pour déterminer s'il y a tentative d'intrusions sur un système, un IDS analyse ce que l'on appelle une trace d'audit. Celle-ci est constituée d'une suite d'actions, ou événements, qui ont été réalisées sur un système. Il peut s'agir par exemple du démarrage d'une application, d'un transfert de fichier, d'une tentative de connexion, du pourcentage de la capacité CPU utilisée à un moment donné, ou de toute autre information en lien avec le système.

L'analyse de la trace d'audit doit permettre de répondre à des questions relatives aux actions exécutées. On voudra connaître quelles opérations ont été faites sur le système, quel utilisateur les a faites et l'heure à laquelle il les a faites, ainsi que les ressources systèmes qui ont été impliquées, et bien entendu, sur quelle machine ces opérations ont été effectuées.

Mé et Alanou (voir [43]) se sont inspirés des travaux de Browne (voir [10]) pour établir une liste (non exhaustive) d'informations pertinentes à auditer. Cette liste consiste en les informations sur :

– les accès au système ;
– l'usage fait du système ;
– l'usage fait des fichiers du système ;

– l'utilisation faite des différentes applications ;
– les violations éventuelles de la sécurité ;
– les données statistiques sur le système.

Alors, en plus de prendre en note les commandes utilisées par l'usager et les applications démarrées, le système d'audit doit tenir compte du résultat des actions effectuées par l'utilisateur. On porte ici une attention particulière aux tentatives d'exécutions de commandes menant à un échec, par exemple, la tentative de lire ou d'effacer un fichier dont l'accès est interdit, ou bien des tentatives répétées de connexion.

La trace d'audit pourra être emmagasinée dans un fichier appelé le journal d'audit de sécurité. Cependant, la quantité d'informations pertinentes peut facilement être gigantesque, on ne peut alors conserver toutes les informations concernant les tâches exécutées sur un système. De plus, la question à savoir quand ces données seront analysées est des plus importantes. Une analyse journalière du fichier d'audit est certes inacceptable vue la complexité des systèmes actuels, on cherche en effet à détecter les attaques avant qu'elles puissent avoir des conséquences désastreuses. On tend alors à développer des IDS qui analysent la trace d'audit en temps réel, ce qui peut représenter un problème sérieux ; en effet, l'IDS devient lui-même nuisible s'il accapare une portion trop importante des ressources systèmes.

1.2.1 Audit système et audit réseau

Une distinction est faite entre les IDS orientés hôte et les IDS orientés réseau (dans la langue anglaise les termes *Host-based IDS (HIDS)* et *Network-base IDS (NIDS)* sont utilisés).

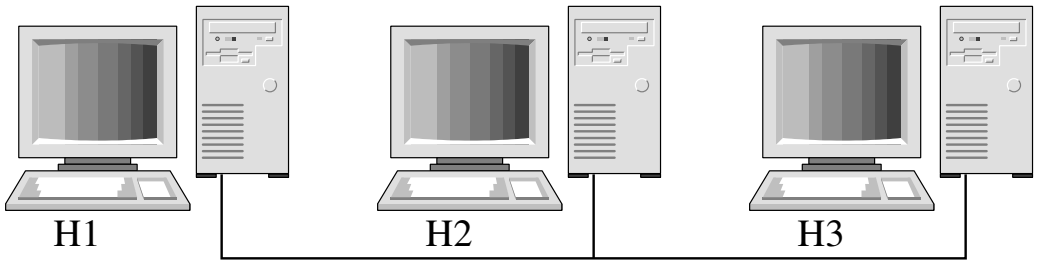
En effet un IDS peut soit se concentrer uniquement sur les données concernant une seule machine, ce qui va de soi pour un ordinateur personnel ne faisant pas partie d'un réseau autonome, soit baser ses traces d'audit sur des données concernant tous les ordinateurs d'un réseau donné (ou un certain nombre d'entre eux).

Le développement d'IDS hôte est facilité par le fait que les systèmes d'exploitation récents, comme Windows ou Linux, intègrent déjà un système d'audit de sécurité pour permettre la collecte d'événements prédéfinis dans un fichier d'audit, en plus de maintenir des historiques sur les commandes système et sur l'usage fait des ressources du système.

Cependant, bon nombre d'attaques ne se passent pas seulement au niveau d'une seule

machine, mais bien au niveau réseau. Considérons l'exemple de topologie réseau donné à la figure 1.1. Ce réseau est visiblement d'une simplicité extrême, il ne compte que trois machines, reliées entre-elles par un simple lien *Ethernet*, le réseau ne comporte aucun commutateur, routeur, ou autre composante du genre. Pourtant, dans ce petit réseau apparaît des problèmes, découlant de la topologie physique, qui ne peuvent surgir avec une machine seule.

FIG. 1.1 – Exemple de topologie réseau



Supposons, dans notre réseau de la figure 1.1, que l'ordinateur de droite, $H1$, envoie un paquet vers $H3$ en falsifiant son adresse pour se faire passer pour $H2$ (on parle alors en anglais de *spoofing* d'adresse). L'hôte $H3$ ne possède pas de moyen personnel pour déterminer que le paquet ne provient pas réellement de $H2$. Cependant, en instaurant un IDS réseau dans $H2$, celui-ci verrait passer le paquet pour en déduire, de la topologie du réseau, qu'il n'est pas normal qu'un paquet provenant obligatoirement de $H1$ ou de $H3$, soit identifié comme provenant de $H2$.

Si la topologie du réseau joue déjà un certain rôle dans un réseau si petit, il est évident qu'elle est d'une importance capitale dans un réseau possédant plusieurs dizaines, voire plusieurs centaines de machines, reliées entre-elles par différentes composantes réseau.

1.3 Les deux grandes approches en détection d'intrusions

Les deux approches étudiées en détection d'intrusions sont l'approche comportementale et l'approche par scénarios. L'approche comportementale est basée sur l'hypothèse qu'un utilisateur donné possède un comportement d'utilisation des ressources système qui lui est propre, et qu'un intrus pourra être repéré du fait qu'il aura un comportement qui ne sera pas conforme au comportement acceptable (celui définit pour l'utilisateur). À l'opposé, l'approche par scénarios est basée sur le fait qu'une attaque peut généralement être identifiée par une signature, soit une suite d'actions qui caractérisent

cette attaque. L'IDS implémentant l'approche par scénarios sera donc constitué d'une banque de signature et d'un moteur recherchant si l'activité du système correspond à une attaque répertoriée.

Ces approches peuvent être implémentée de diverses façons, chacune ayant ses forces et ses faiblesses, la suite de cette section sera consacrée à une étude plus détaillée de ces deux approches.

1.3.1 Approche comportementale

L'approche comportementale a été proposée en 1980 par Anderson (voir [5]). L'idée maîtresse de cette approche est de prendre en compte le comportement d'un utilisateur de sorte à détecter tout comportement anormal, que l'on considère alors comme une possible tentative d'attaquer le système.

Les outils basés sur cette approche comportent deux phases importantes. La première, la phase d'apprentissage, est celle durant laquelle l'on cherche à déterminer le profil d'un utilisateur. L'approche statistique développée par Denning (voir [21]), que nous étudierons plus particulièrement, utilise les statistiques d'utilisation du système pour déterminer le profil d'un utilisateur ; il existe également d'autres méthodes pour l'apprentissage du comportement. La seconde phase d'un système basé sur l'approche comportementale est la phase dite de reconnaissance. Dans cette phase, le système tente de déterminer si le comportement de l'utilisateur en cours dévie significativement du comportement de l'utilisateur normal.

Bien sûr, la seconde phase ne peut démarrer qu'après que la première eut caractérisé le profil de chacun des utilisateurs, un délai, qui n'est pas toujours facilement déterminable, est donc requis avant de passer à la seconde phase. Cependant, idéalement, la première phase ne doit pas prendre fin avec le début de la seconde, en effet, il peut être souhaitable de suivre le comportement d'un utilisateur, qui peut se modifier avec le temps, mais il est à noter que ceci peut être utilisé par un usager pour modifier graduellement son comportement de sorte à faire passer pour acceptable un comportement malveillant.

Nous allons maintenant parcourir différentes techniques pour implémenter l'approche comportementale, avec un regard particulier sur l'approche statistique.

Approche statistique

Le modèle statistique de Denning comporte les six composantes principales suivantes :

Sujet : L'utilisateur ;

Objet : Le système à surveiller ;

Enregistrements d'audit : Ceux-ci doivent être générés par le système pour toute action exécutée par un sujet sur un objet ;

Profils : Caractérise le comportement d'un sujet en fonction d'un objet donné ;

Enregistrement d'anomalies : Généré lorsqu'un comportement anormal est détecté ;

Règles d'activité : Ces règles permettent la mise à jour des profils, elles mettent en lien des anomalies avec des possibles attaques et produisent des rapports d'activité.

Les premières composantes du modèle sont communes à tout système de détection d'intrusions. Par contre, la composante pour les profils nous indique qu'il s'agit bien d'une approche de type comportementale.

Une question importante dans ce modèle concerne l'établissement des profils. Ceux-ci seront calculés pour chaque utilisateur en fonction de leur habitude d'utilisation du système et en fonction d'une métrique prédéterminée. Un comportement sera alors jugé anormal s'il s'éloigne trop d'un comportement normal suivant cette métrique.

La métrique peut être définie en fonction de différents éléments :

Comptage d'événements : on considère le nombre d'occurrence de certains événements dans un laps de temps donné ;

Intervalles de temps : on considère le temps séparant certains événements ;

Données sur les ressources : on tient alors compte de certaines informations découlant de l'utilisation du système, telles que l'utilisation plus ou moins intensive des ressources CPU.

Denning donne également différentes techniques pour définir la phase de reconnaissance du modèle. Celles-ci se basent sur l'application d'une métrique spécifique sur n observations $x_1, \ldots, x_n$. Ces techniques ont alors toutes pour objectif de déterminer si une $(n+1)^{\text{e}}$ observation x_{n+1} est normale ou non (cette décision étant basée sur les n premières observations).

Une première technique consiste à calculer la moyenne μ et l'écart type σ des n premières observations. L'observation x_{n+1} sera jugée anormale si elle n'appartient pas à un intervalle de confiance $\mu + d * \sigma$ où d est un paramètre permettant de s'éloigner plus ou moins de la moyenne selon la rigueur désirée.

D'autres techniques tentent de calculer la probabilité que l'événement x_{n+1} suive les événements $x_1, \ldots, x_n$. Un événement x_{n+1} sera alors jugé anormal si sa probabilité d'apparaître est trop faible.

Une façon encore plus simple d'implanter le modèle statistique consiste en la fixation de limite. Un comportement sera jugé anormal si, dans un intervalle de temps donné, certains événements se produisent un nombre trop élevé de fois. Par exemple, on peut penser à de nombreuses tentatives infructueuses de connexion effectuées en un court laps de temps.

1.3.2 Autres approches comportementales

L'approche statistique, bien qu'étant l'approche la mieux documentée, n'est pas la seule basée sur l'approche comportementale. Les systèmes experts, ainsi que les réseaux de neurones peuvent également être à la base d'un IDS implantant l'approche comportementale.

Les systèmes experts

En apparence, les systèmes experts semblent plus liés à l'approche par scénarios. En effet, un système expert possède une base de règles et détermine si un comportement est anormal en suivant ces règles. Ces dernières peuvent être définies manuellement, par l'administrateur réseau par exemple, pour refléter le comportement souhaité pour un utilisateur et imposer la politique de sécurité établie. Il n'y a donc pas de réelle phase d'apprentissage.

La base de règles peut également être déterminée dynamiquement pour refléter le comportement réel de l'utilisateur. La phase d'apprentissage consiste alors à déduire ces règles à partir du comportement de l'utilisateur. Cependant, cette base de règle peut être très complexe à gérer et à maintenir et elle peut conduire à des problèmes de performance si elle devient trop volumineuse.

Les réseaux de neurones

Les réseaux de neurones, très populaires dans certains domaines scientifiques, ont également été étudiés dès le début des années 90 pour leurs possibles applications à la détection d'intrusions, voir par exemple [11, 19, 20, 23]. Les réseaux de neurones peuvent être exploités pour obtenir une modélisation statistique du comportement d'un utilisateur. Cependant, lorsqu'un réseau de neurones signale une anomalie, il n'en précise pas la cause, ce qui ne facilite pas la prise de décision pour rectifier la situation. La question à savoir quand un réseau de neurones a complété sa phase d'apprentissage est également délicate, et cruciale pour le bon déroulement du système.

1.3.3 Approche par scénarios

Les algorithmes inspirés de l'approche par scénarios sont basés sur un ensemble de règles, ou signatures, qui identifient les différentes attaques ainsi que la mesure à prendre face à celles-ci. On parle alors entre autres de systèmes experts, soit un ensemble de règles de la forme « si telle signature se retrouve dans la trace d'audit, alors ... » Cependant, l'explosion combinatoire peut rendre inutilisable un tel mécanisme. C'est pourquoi différentes techniques ont été développées pour identifier plus efficacement les signatures d'attaque ou pour filtrer la trace d'audit de sorte à analyser seulement des éléments pertinents. Voici maintenant quelques techniques pour implanter l'approche par scénarios.

Reconnaissance de formes (*Pattern matching*)

La trace d'audit est traduite en un langage spécifique en ne gardant que les informations sensibles. Les attaques sont décrites comme des mots dans ce langage, la recherche d'une attaque devient alors en quelque sorte une recherche d'un mot dans une phrase. Cette technique est implantée dans plusieurs IDS sur le marché, c'est le cas par exemple de *RealSecure* (voir [26]) et *Cisco IDS* (voir [14]).

Analyse de transitions d'état

Cette technique base la représentation d'une attaque sur un automate dont l'état initial est dit non compromis et dont certaines transitions (qui représentent des éléments d'attaque) mènent à un ou plusieurs états compromis. Nous parlerons plus en détails de cette technique dans la section suivante portant sur la méthode STAT.

Réseaux de neurones

Tout comme pour l'approche comportementale, les réseaux de neurones peuvent être utilisés dans un algorithme basé sur l'approche par scénarios. L'avantage des réseaux de neurones étant qu'ils permettent d'obtenir un système très efficace du point de vue de la rapidité, cependant, ils ont les mêmes désavantages que pour l'approche comportementale, c'est-à-dire qu'il n'est pas simple de comprendre pourquoi un réseau de neurones déclenche une alerte.

Algorithmes génétiques

Ces algorithmes tentent de réduire l'impact des problèmes dus à l'explosion combinatoire, en se basant sur une technique de recherche d'un extremum inspirée de la théorie génétique de l'évolution. L'étude de l'application des algorithmes génétiques à la détection d'intrusions fait l'objet de la thèse de Ludovic Mé (voir [45]).

1.3.4 Coopération entre les différentes approches

La faiblesse des algorithmes basés sur l'approche par scénarios est qu'ils ne permettent de détecter que des attaques répertoriées, et donc connues. Ils sont donc sujets à produire plusieurs faux-négatifs (attaques non repérées). Par contre, puisqu'ils reposent sur une banque d'attaques, ils ne sont pas sujets, ou peu, à produire des faux-positifs (signal d'une fausse attaque), sauf bien sûr si la banque d'attaques comporte des erreurs.

Pour ce qui est de l'approche comportementale, sa force principale réside dans le fait qu'elle permet de détecter des attaques non répertoriées. Elle est cependant sujette à produire de nombreux faux-positifs et faux-négatifs, puisqu'il peut être très difficile de déterminer le comportement d'un utilisateur.

Dans [42], les auteurs signalent l'importance de faire collaborer différentes approches de sorte à combiner les forces de chacune tout en éliminant les faiblesses. Les outils développés en détection d'intrusions se basent principalement sur l'approche par scénarios, cependant, certains outils sérieux, tel que *Cisco IDS*, intègrent en plus de cette approche, l'approche comportementale.

1.4 Détection d'intrusions et méthodes formelles

Dans [46], les auteurs opposent la détection d'intrusions aux méthodes formelles pour sécuriser un réseau informatique. Cependant, les méthodes formelles peuvent être appliquées de façon à améliorer la détection d'intrusions. Il ne faut donc pas voir ces domaines comme étant concurrents, mais plutôt alliés dans l'optique de la sécurisation des réseaux.

En effet, nous devons avoir des outils pour nous assurer que le fait de rechercher une certaine séquence dans une trace à un endroit précis du réseau, permettra de contrer une attaque. Le problème devient très sérieux dans le cas des IDS réseaux, puisque le flux de données devient énorme, il faut donc se demander précisément où, dans le réseau, collecter les informations, et quelles informations doivent être collectées pour assurer la sécurité de toutes les machines du réseau sans engorger le système.

Nous présentons maintenant, en guise d'exemple, deux outils réalisés en détection d'intrusions qui exploitent les méthodes formelles.

1.4.1 STAT

STAT (de l'anglais *State Transition Analysis Technique*, voir [27] et [53]), est basé sur les techniques d'analyse de transition d'état. Une attaque sera modélisée par un automate augmenté (dont les états peuvent posséder des variables), l'automate possédant un état initial dit sécuritaire et peut aboutir, par suite de transitions correspondant à une attaque, vers un état dit compromis.

Les attaques sont définies à l'aide d'un langage de haut niveau appelé STATL (voir [22]). STATL est un langage extensible qui permet de représenter les attaques avec un certain niveau d'abstraction en prenant compte de quelques propriétés clés de l'attaque. De cette façon, un seul scénario d'attaque peut représenter une classe entière d'attaques.

Une extension de STAT, appelée USTAT (voir [25]), applique la méthode STAT à un système Unix. Cette extension permet de spécifier les signatures d'attaque au niveau hôte. Une seconde extension, NetSTAT (voir [54]), permet de spécifier des attaques au niveau réseau.

NetSTAT

L'approche NetSTAT requiert que le réseau soit modélisé formellement suivant un modèle basé sur les hypergraphes. Cela permet de tenir compte de la topologie particulière d'un réseau donné pour spécifier correctement les attaques. L'architecture de NetSTAT se compose des éléments suivants :

- Base de données du réseau : cette base de données gère tous les éléments de sécurité importants du réseau. Elle contient des informations sur la topologie du réseau ainsi que sur les différents services offerts sur le réseau.

- Base de scénarios de transition d'état : cette composante gère les ensembles de transitions d'état représentant les attaques à prévenir ;

- Les senseurs (*Probes*) : ces composantes permettent de capturer, à différents endroits dans le réseau, les informations pertinentes sur le trafic ;

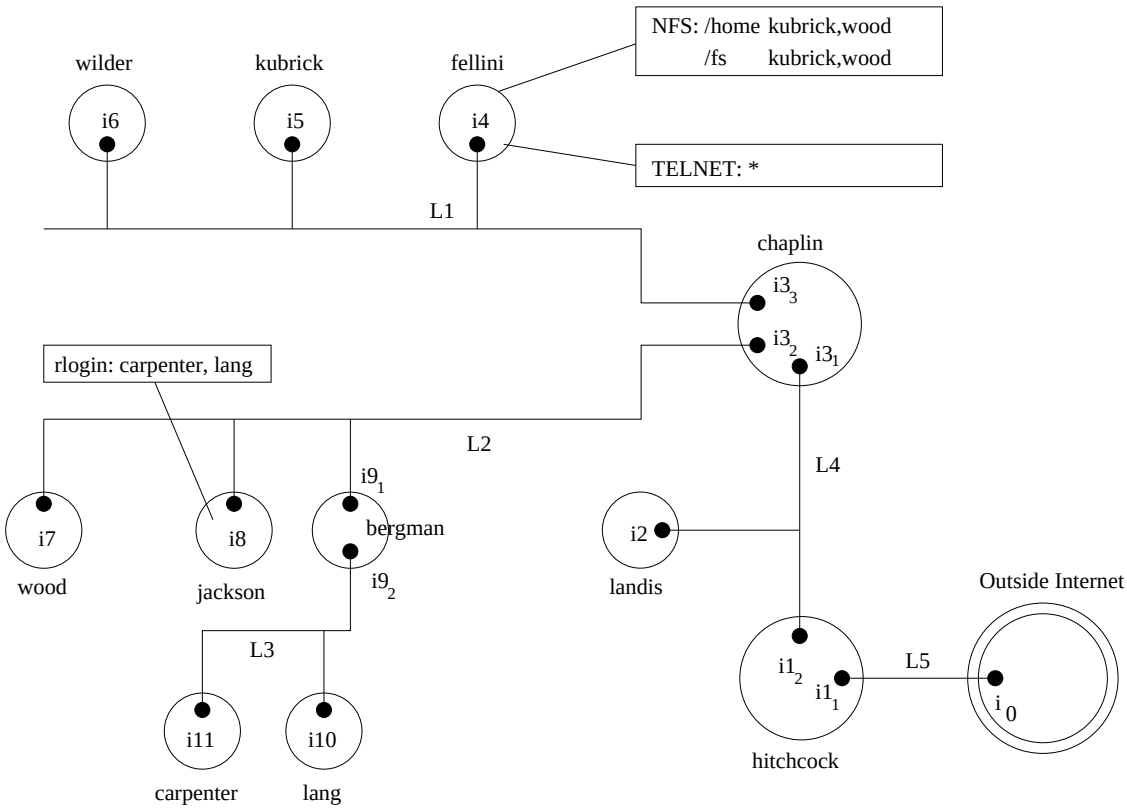
- L'analyseur : certes la partie la plus importante de l'architecture, l'analyseur a pour tâche, à partir des informations provenant de la base de données réseau et de la base de scénarios de transition d'état, de déterminer où placer les senseurs pour collecter les informations permettant de détecter les attaques.

L'approche NetSTAT consiste donc à modéliser le réseau ainsi que les attaques à prévenir, puis, l'analyseur indique où placer des senseurs pour permettre de collecter les informations nécessaires à détecter les attaques. La figure 1.2 fournit un exemple de représentation de réseau avec NetSTAT, cet exemple est tiré de [54].

1.4.2 IDIOT

Pour sa part, l'approche utilisée dans le projet IDIOT (voir [18, 28, 29, 30]), se base sur les réseaux de Petri colorés pour représenter les scénarios d'attaques. Ce qui permet

FIG. 1.2 – Exemple de topologie réseau avec NetStat



d'exploiter l'aisance avec laquelle on peut représenter les notions de parallélisme et de synchronisation avec des réseaux de Petri.

IDIOT est constitué des trois composantes suivantes :

showaudit.pl : script Perl qui convertit les traces d'audit dans un format spécifique ;

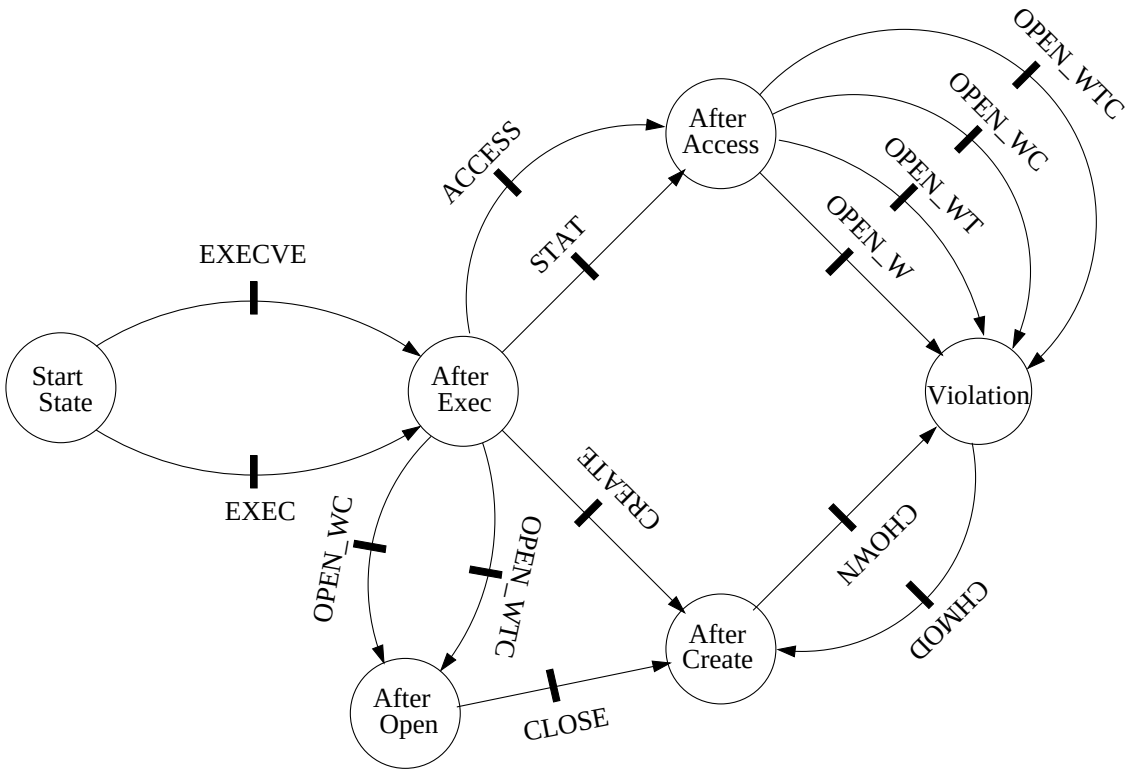
C2_server : le module principal, celui-ci lit la trace générée par le module précédent pour rechercher, par reconnaissance de formes, des attaques possibles ;

C2_appl : une application graphique permettant d'interagir avec le programme.

La trace d'audit prise en compte par le script *showaudit.pl* provient directement du système d'exploitation.

La figure 1.3 montre un exemple de représentation d'une attaque avec IDIOT.

FIG. 1.3 – Exemple de représentation d'une attaque avec IDIOT



1.5 Derniers mots sur la détection d'intrusions

Dans ce premier chapitre, nous avons exploré les différentes approches employées pour s'attaquer au problème de la détection d'intrusions. Nous avons présenté très peu d'outils pratiques pour nous concentrer sur les approches globales, cet ouvrage n'étant pas centré sur la détection d'intrusions en soi mais bien sur la sécurisation formelle de réseau. Le lecteur intéressé à découvrir d'autres outils pratiques implantant les approches parcourues ici est invité à consulter les ouvrages d'Axelsson (voir [6]) et de Couture (voir [17]).

Nous avons vu que les IDS peuvent se baser sur une approche comportementale ou par scénarios (ou une combinaison des deux) pour détecter les attaques sur un réseau ou une machine ; chacune de ces méthodes possédant ses forces et ses faiblesses. Cependant, pour la suite de cet ouvrage, l'approche comportementale étant plutôt difficile à modéliser formellement, elle sera écartée et nous nous baserons sur l'approche par scénarios pour représenter les attaques.

Notre objectif ne sera pas de représenter un système de détection d'intrusions, mais plutôt de développer un langage formel pour modéliser un réseau possédant des moniteurs (des pare-feu) pouvant suivre une trace, soit une suite de paquets, à différents endroits du réseau, de sorte à bloquer certains paquets pouvant faire partie d'une attaque, ou brimant la politique de sécurité du réseau. Nous ne ferons donc pas de la détection d'intrusions, mais plutôt de la prévention d'intrusions, en s'inspirant des travaux faits en détection d'intrusions.

Chapitre 2

Langages de spécification de systèmes

Dans ce chapitre, nous nous consacrons à l'étude des méthodes formelles. Dans un premier temps, nous décrivons différents langages destinés à la modélisation formelle des systèmes communicants tels qu'un réseau informatique. Nous portons une attention particulière à certaines algèbres de processus liées à CCS, soient CCS elle-même, le π-calcul et le calcul *ambient*. Nous voyons ensuite une façon de modéliser des systèmes communicants à l'aide d'automates munis d'un produit synchrone. Finalement, nous abordons le classement et l'écriture à l'aide de logique des politiques de sécurité.

2.1 Introduction

Nous avons vu, dans le premier chapitre, des approches pour développer des systèmes de détection d'intrusions. Ces systèmes nous permettent de repérer des tentatives d'intrusion sur un réseau informatique ; l'administrateur réseau, informé d'une tentative d'intrusion, peut alors réagir de sorte à contrer cette menace avant qu'elle ne cause des problèmes sérieux, si ce n'est pas déjà fait. Mieux encore, on peut concevoir des systèmes qui répondent automatiquement à des tentatives d'intrusion, par exemple en bloquant un port suite à une séquence suspecte d'actions ; on s'assure ainsi d'une réponse rapide limitant les dégâts et on parle donc maintenant de prévention d'intrusions. Une question se pose toutefois : comment s'assurer que l'outil que l'on a implanté dans un réseau permet réellement de contrer les attaques dont on veut se protéger ?

Pour s'assurer de la bonne implantation d'un système de prévention d'intrusions, il nous faut des outils permettant de vérifier formellement cette implantation. C'est-à-dire qu'étant donné un réseau informatique possédant des composantes de sécurité (pare-feu), nous avons besoin d'outils permettant de confirmer (ou d'infirmer) qu'une configuration donnée des composantes de sécurité permet réellement de faire respecter une politique de sécurité préalablement établie. Une politique de sécurité décrit en quelque sorte un code de conduite que notre réseau doit respecter, elle peut stipuler par exemple que les échanges de paquets entre deux sous-réseaux précis sont interdits, ou restreindre les sites Web sur lesquels une machine peut naviguer.

Bien que dans plusieurs cas simplistes il semble qu'il soit facile de configurer les composantes de sécurité de manière *ad hoc*, tout en gardant une certaine assurance que cette façon de faire est rigoureuse, il en est tout autrement dans des cas plus complexes. En effet, les réseaux d'aujourd'hui peuvent comporter plusieurs centaines de machines et des dizaines de sous-réseaux ; cette complexité des réseaux rend irréaliste l'approche *ad hoc* pour implanter une politique de sécurité. Comment s'assurer, par exemple, que les différentes configurations des différents pare-feu d'un réseau ne sont pas contradictoires ? Il n'y a pas à faire de telles vérifications s'il n'y a qu'un pare-feu sur un réseau, mais elles deviennent nécessaires s'il y en a des dizaines. Pour s'assurer pleinement de leur valeur, ces vérifications ne peuvent se faire que sur une base rigoureuse, une base mathématique.

On regroupe sous l'appellation de méthodes formelles les mathématiques appliquées à la modélisation et à l'analyse des systèmes d'information. L'idée associée à l'utilisation des méthodes formelles est de modéliser le système que l'on veut étudier (dans notre cas, il s'agit d'un réseau informatique) à l'aide d'un langage mathématique qui se doit d'être, autant que possible, simple, intelligible et expressif. Une fois le modèle obtenu, des outils mathématiques peuvent être appliqués sur ce modèle pour vérifier si celui-ci respecte les propriétés que l'on voudrait que le système initial respecte.

Les propriétés que l'on veut vérifier doivent également être spécifiées dans un langage clair et non ambigu, si l'on souhaite que la démarche soit complètement rigoureuse. En somme, nous avons recours à un langage de spécification de système pour modéliser le réseau, et à un langage de spécification de propriétés pour énoncer formellement les propriétés que l'on souhaite vérifier ; pour cette dernière tâche, les logiques temporelles telles que LCL sont particulièrement appropriées. Une fois toutes ces spécifications effectuées, la tâche revient à un vérificateur de modèle (*model-checker*) de vérifier si le modèle respecte bien les propriétés spécifiées.

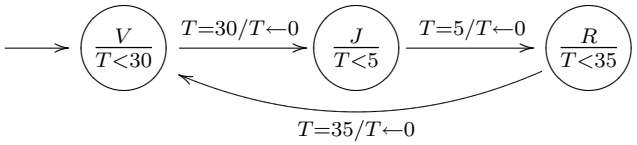
Bien sûr, les vérifications sont effectuées sur le modèle, et non pas sur le système initial.

Ainsi, le niveau d'abstraction adopté pour modéliser le système doit dépendre des propriétés à vérifier ; il est impossible, par exemple, de vérifier des propriétés qui tiennent compte du temps si notre modèle fait abstraction de cette notion. De plus, si une erreur se trouve dans la modélisation elle-même (si celle-ci ne reflète pas correctement le système initial), cela peut mener la vérification à des résultats erronés. L'abstraction du système peut également être perçue comme un avantage de la méthode. En effet, si les propriétés que l'on souhaite vérifier sur un système ne nécessitent pas la notion du temps, en abstrayant cette notion dans le modèle du système, on rend ce dernier beaucoup plus simple que le système original, ce qui rend donc également plus simple l'étape de vérification des propriétés sur le modèle.

2.1.1 Langages de spécification de systèmes

Plusieurs langages formels ont été développés dans le but de décrire mathématiquement, sans ambiguïté, des systèmes d'information ; on parle entre autres des réseaux de Petri, des automates, ainsi que des algèbres de processus. Ce dernier groupe de langage s'est particulièrement développé dans les deux dernières décennies ; on compte aujourd'hui des dizaines d'algèbres de processus dont la majorité ont pour point de base de considérer un système, aussi complexe qu'il puisse être, en fonction des actions dites élémentaires qu'il peut exécuter. Par exemple, lorsqu'on modélise un réseau informatique, si l'on s'intéresse aux communications entre les différentes machines d'un réseau, on pourra considérer un envoi d'un paquet IP sur le réseau comme une action élémentaire. Des opérateurs propres à chaque algèbre permettent de définir l'évolution des processus après l'exécution d'une action ; ces opérateurs peuvent représenter des concepts tels que le parallélisme, le synchronisme, le branchement conditionnel ou la communication entre différents processus.

Les automates sont également utilisés pour modéliser des systèmes de toute nature. Un système modélisé par un automate est vu comme un ensemble d'états reliés entre eux par des étiquettes, ces étiquettes définissent les transitions entre les états. L'automate possède un état initial (l'état du système au point de départ), par suite de transitions, qui peuvent être étiquetées, par exemple, par des mots pouvant représenter une action externe, ou par des conditions temporelles, l'automate traverse différents états. Par exemple, il est facile de modéliser un feu de circulation simple : le feu débute à l'état vert, puis il atteint les états jaune, rouge et à nouveau vert, suite à des délais respectifs de 30, 5 et 35 secondes. Sous la forme d'un automate, ce feu de décrit ainsi :



Il n'est pas difficile de voir comment on peut améliorer cette modélisation, par exemple en ajoutant un état pour le passage des piétons que l'automate pourra atteindre si le bouton est activé.

2.1.2 Contenu du chapitre

Cette section sur les langages formels traite principalement des algèbres de processus, celles-ci nous ont paru particulièrement appropriées pour la modélisation des systèmes communicants. Et comme l'ensemble des algèbres de processus développées à ce jour est trop vaste, nous nous sommes concentrés sur les algèbres dérivées (directement ou indirectement) de CCS, dont le π-calcul et le calcul *ambient*. Suite à une revue des différentes algèbres de processus, nous verrons comment les automates peuvent également être utilisés pour modéliser des systèmes communicants. La dernière section du chapitre sera consacrée aux propriétés de sécurité ; nous aborderons dans cette section les principales classes de propriétés, ainsi que leur écriture à l'aide d'une logique linéaire.

2.2 CCS

Les algèbres de processus sont des langages formels permettant de modéliser le fonctionnement de processus plus ou moins complexes en terme des actions qu'ils peuvent réaliser. La première algèbre de processus fut développée par Milner en 1980 et d'abord publiée dans [38], sous l'appellation de *Calculus of Communicating Systems*, soit *CCS*, puis republiée dans [39] et [41].

Un processus est vu comme un objet pouvant évoluer en un autre objet par l'exécution d'une action. Par exemple, le fait qu'un processus E puisse exécuter l'action a pour devenir le processus F sera noté $E \xrightarrow{a} F$. La force principale des algèbres de processus réside dans le fait qu'elles peuvent modéliser des systèmes d'une certaine complexité à l'aide d'un nombre très restreint d'opérateurs, qui peuvent décrire, par exemple, le parallélisme ou la communication entre sous-processus.

Un processus complexe sera alors considéré comme étant un ensemble de processus plus

simples dont l'interaction mutuelle est régie par certains opérateurs propres à l'algèbre.

2.2.1 Syntaxe de CCS

En CCS, les processus sont d'abord définis à l'aide d'un ensemble d'actions

$$\Sigma = \{a, b, c, \ldots\}$$

et par l'ensemble des co-actions

$$\overline{\Sigma} = \{\overline{a}, \overline{b}, \overline{c}, \ldots\}.$$

Par analogie à un groupe en mathématique, une co-action serait l'inverse d'une action, on a alors $a = \overline{\overline{a}}$, pour tout $a \in \Sigma \cup \overline{\Sigma}$. Pour donner un exemple plus imagé, supposons que l'action *monaie* représente l'action de recevoir une pièce pour une machine distributrice, alors, pour le client de cette machine, l'action $\overline{monaie}$ représentera l'action de donner une pièce à cette machine.

L'alphabet des actions de CCS est aussi muni d'une action spéciale dénotée par τ et dite silencieuse, ou invisible. Cette action possède la caractéristique d'être la seule action à être sa propre co-action, on a donc $\tau = \overline{\tau}$. Toujours par analogie à un groupe, τ serait l'élément neutre. Cette action résultera souvent, mais pas toujours, d'une communication entre sous-processus, elle pourra donc, dans certains cas, être exécutée par un processus même si aucun de ses sous-processus ne peut l'exécuter.

En somme l'alphabet complet des actions de CCS correspond à

$$Act = \Sigma \cup \overline{\Sigma} \cup \{\tau\}.$$

Le tableau 2.1 donne la syntaxe complète de CCS.

TAB. 2.1 – Syntaxe de CCS

$$\mathbf{P ::= 0 \mid x.P \mid \overset{\text{def}}{=} P \mid P_1 + P_2 \mid P_1 | P_2 \mid P[f] \mid P \setminus L}$$

La lettre majuscule P dénote un élément de l'algèbre, donc un processus, alors que le symbole 0, parfois également dénoté *nil*, correspond au processus de base, soit un processus qui ne peut exécuter aucune action, un processus vide, inerte. Les autres éléments de la syntaxe sont les opérateurs de préfixage, de définition, de choix, de parallélisme, de renommage et de restriction.

TAB. 2.2 – Sémantique opérationnelle de CCS

(Préfixage)	$\dfrac{\square}{a.P \xrightarrow{a} P}$
(Définition)	$\dfrac{E \xrightarrow{a} F,\ P \stackrel{\text{def}}{=} E}{P \xrightarrow{a} F}$
(Choix)	$\dfrac{E \xrightarrow{a} P}{E + F \xrightarrow{a} P} \qquad \dfrac{F \xrightarrow{a} P}{E + F \xrightarrow{a} P}$
(Entrelacement)	$\dfrac{E \xrightarrow{a} P}{E \mid F \xrightarrow{a} P \mid F} \qquad \dfrac{F \xrightarrow{a} P}{E \mid F \xrightarrow{a} E \mid P}$
(Communication)	$\dfrac{E \xrightarrow{a} E',\ F \xrightarrow{\overline{a}} F'}{E \mid F \xrightarrow{\tau} E' \mid F'}$
(Restriction)	$\dfrac{E \xrightarrow{a} E'}{E \setminus L \xrightarrow{a} E' \setminus L}$ si $a \notin L \cup \overline{L}$
(Renommage)	$\dfrac{E \xrightarrow{a} E'}{E[f] \xrightarrow{f(a)} E'[f]}$

2.2.2 Sémantique opérationnelle de CCS

Dans cette section, nous regardons en détails le fonctionnement et le rôle précis de chacun des opérateurs de CCS. La sémantique opérationnelle de ces opérateurs est affichée au tableau 2.2.

Le premier opérateur, et certes le plus important, est celui dit de préfixage, qui est dénoté par un point. Cet opérateur marque, pour un processus donné, la capacité d'exécuter une action. Par exemple, le processus $a.0$ peut exécuter l'action a pour devenir le processus 0, ce qui se note

$$a.0 \xrightarrow{a} 0.$$

De même, le processus $a.a.0$ peut exécuter deux fois l'action a, mais pas plus.

L'opérateur de définition, noté $\stackrel{\text{def}}{=}$, permet de définir récursivement des processus, ce qui

nous permet d'obtenir des processus capable d'exécuter une action une infinité de fois. Un exemple classique, qui de fait est l'exemple canonique, nous est présenté par Hoare (voir [24]) :

$$H \stackrel{\text{def}}{=} tic.H.$$

Cet exemple correspond à une horloge capable d'exécuter un nombre infini de *tic* et ne pouvant exécuter aucune autre action.

CCS est également muni de deux opérateurs non déterministes, l'opérateur de choix $+$ et l'opérateur d'entrelacement $|$. L'opérateur de choix permet de faire évoluer un processus suivant l'évolution d'un de ses sous-processus et en oubliant les autres. Par exemple, le processus $a.b.0 + b.a.0$ possède deux possibilités d'évolution, soient :

$$a.b.0 + b.a.0 \xrightarrow{a} b.0 \xrightarrow{b} 0$$

et

$$a.b.0 + b.a.0 \xrightarrow{b} a.0 \xrightarrow{a} 0.$$

Dans le premier cas, l'évolution du processus global se fait suivant celle du sous-processus de gauche, et dans le second cas, suivant celle du processus de droite. Dans cet exemple, le choix du sous-processus est déterminé par la première action exécutée. Ceci n'est pas toujours le cas, par exemple, pour le processus $a.0 + a.a.0$, les deux évolutions suivantes sont possibles : $a.0 + a.a.0 \xrightarrow{a} 0$ ou $a.0 + a.a.0 \xrightarrow{a} a.0 \xrightarrow{a} 0$. L'opérateur de choix en est donc un non déterministe.

Pour sa part, l'opérateur $|$ permet une évolution en parallèle des sous-processus. Par exemple, le processus $a.0|b.0$ peut évolution des deux manières suivantes :

$$a.0 \mid b.0 \xrightarrow{a} 0 \mid b.0 \xrightarrow{b} 0 \mid 0$$

et

$$a.0 \mid b.0 \xrightarrow{b} a.0 \mid 0 \xrightarrow{a} 0 \mid 0.$$

Cependant, cette évolution se fait par entrelacement, et non de façon simultanée. Par contre, le même opérateur peut définir une communication entre sous-processus, et dans ce cas, l'évolution se fait de façon simultanée, l'un des sous-processus envoie une action et l'autre la reçoit, comme dans l'exemple suivant :

$$a.0 \mid \overline{a}.0 \xrightarrow{\tau} 0 \mid 0.$$

L'action spéciale τ représente une communication interne. Il faut noter que la communication n'est pas forcée, le même processus pourrait exécuter l'action a sans sa co-action ou vice-versa : $a.0|\overline{a}.0 \xrightarrow{a} 0|\overline{a}.0$. Pour forcer la communication, on a recourt à l'opérateur

de restriction, ce même processus $(a.0|\overline{a}.0)$ lorsqu'il est restreint par l'action $\{a\}$ peut toujours exécuter une communication τ :

$$(a.0 \mid \overline{a}.0) \setminus \{a\}) \xrightarrow{\tau} (0 \mid 0) \setminus \{a\}$$

cependant l'exécution de a ou $\overline{a}$ sans sa co-action n'est plus permise.

Un dernier opérateur, dit de renommage, facilite l'écriture des processus en permettant de définir un processus à partir d'un autre processus et d'une fonction de renommage qui prend pour entrée une action et retourne une autre action (ou la même) en sortie.

2.2.3 Le passage de valeur

Une extension simple de CCS, se trouvant dans [51], permet le passage de valeur, soit l'utilisation de variables dans une action, ce qui fournit des actions ayant la forme $a(x)$. On considère alors que a est un canal de communication et que x est une donnée communiquée à travers le canal a. L'action entière $a(x)$ représente alors la réception de la valeur x à travers le canal a, alors que $\overline{a}(x)$ représente l'envoie de la valeur x sur le canal a. Voici un exemple de communication par passage de valeur entre trois sous-processus :

$$\begin{aligned} \overline{a}(1).0 \mid a(x).\overline{b}(x).0 \mid b(y).c(y).0 &\xrightarrow{\tau} 0 \mid \overline{b}(1).0 \mid b(y).\overline{c}(y).0 \\ &\xrightarrow{\tau} 0 \mid 0 \mid \overline{c}(1).0. \end{aligned}$$

Dans un premier temps, le sous-processus de gauche communique la valeur 1 à celui du milieu par le biais du canal a, et dans un second temps, le sous-processus du milieu communique cette même valeur à celui de droite par le biais du canal b.

Comme l'indique Stirling dans [51], le passage de valeurs n'ajoute cependant rien à l'expressivité de CCS, il ne fait que facilité l'écriture des processus. En effet, on peut voir le processus $a(x).E$ comme étant une écriture raccourcie du processus $\sum\{a(v).E\{v/x\} \ : v \in D\}$ où D est le domaine des valeurs possibles pour x. Pour la sémantique opérationnelle du passage de valeur, se référer au tableau 2.3.

2.2.4 Équivalence entre processus

Il existe différentes façons de définir l'équivalence entre processus, cela dépend en fait de nos besoins spécifiques. Par exemple, avec l'équivalence de trace, on considère comme

TAB. 2.3 – Sémantique du passage de valeur en CCS

$$\text{(Entrée)} \quad \frac{v \in D}{a(x).E \xrightarrow{a(v)} E\{x/c\}}$$

$$\text{(Sortie)} \quad \frac{\text{Val(e)} = \text{v}}{\overline{a}(e).E \xrightarrow{\overline{a}(v)} E}$$

étant équivalents deux processus engendrant le même ensemble de traces d'exécution. Les processus $a.0 + b.0$ et $a.0 \mid b.0$ sont donc équivalents de trace, puisque tous deux admettent l'ensemble $\{a, b, a.b, b.a\}$ comme ensemble de traces d'exécution, alors que les deux processus $a.0$ et $b.0$ ne sont clairement pas équivalents avec cette relation d'équivalence.

L'équivalence de trace n'est pas la seule relation permettant de comparer des processus, une équivalence bien connue pour CCS a été introduite par Milner, il s'agit de la relation de bissimulation. Cette relation est beaucoup plus stricte que l'équivalence de trace, en fait, si deux processus sont bissimilaires, ils sont également équivalents de trace, alors que l'inverse ne tient pas. La définition des relations de bissimulation forte et faible se retrouve dans [41], nous nous servirons de la proposition 2.2.1, pour donner un exemple de processus équivalent de trace mais non bissimilaires.

Proposition 2.2.1 (Relation de bissimulation)**.** *Deux processus E et F sont en relation de bissimulation, ce qui est noté $E \sim F$, si, peu importe l'action a nous avons les deux points suivants :*
– *si $E \xrightarrow{a} E'$ alors $F \xrightarrow{a} F'$ pour un certain F' tel que $E' \sim F'$*
– *si $F \xrightarrow{a} F'$ alors $E \xrightarrow{a} E'$ pour un certain E' tel que $E' \sim F'$.*

Considérons les deux processus suivants :

$$\begin{aligned} P &\stackrel{\text{def}}{=} a.(b.0 + c.0) \\ Q &\stackrel{\text{def}}{=} a.b.0 + a.c.0, \end{aligned}$$

ces deux processus sont bien équivalents de trace, ils admettent tous deux l'ensemble $\{a, a.b, a.c\}$ comme ensemble de traces d'exécution. Cependant, ils ne sont pas bissimilaires, ceci se voit par le fait que pour le processus Q, la première action exécutée, qui est forcément a, détermine la seconde action à pouvoir être exécutée, ce qui n'est pas le cas pour le processus P. Selon la proposition 2.2.1, pour que P et Q soient bissimilaires,

il faudrait que le processus $b.0 + c.0$ soit en relation de bissimulation avec le processus $b.0$ ou le processus $c.0$; il ne peut pas être bissimilaire au processus $b.0$, puisque $b.0 + c.0$ peut exécution l'action c, ce que ne peut pas faire le processus $b.0$, et il ne peut non plus être bissimilaire au processus $c.0$ puisque ce dernier ne peut exécuter l'action b.

Il existe également d'autres relations d'équivalences utilisées pour comparer des processus, les notes de Stirling (voir [51]) traitent de façon bien détaillée ces divers types d'équivalences. Sans entrer trop dans les détails, nous signalons que l'on s'intéresse particulièrement aux relations dites de congruence, ces relations respectent certaines propriétés intéressantes telles que celles énoncées dans la proposition ci-bas.

Proposition 2.2.2 (Relation de congruence)**.** *Si P_1 et P_2 sont deux processus tels que $P_1 \simeq P_2$, pour $\simeq$ une relation de congruence, alors*

1. $a.P_1 \simeq a.P_2$
2. $P_1 + Q \simeq P_2 + Q$
3. $P_1 \mid Q \simeq P_2 \mid Q$
4. $P_1 \setminus L \simeq P_2 \setminus L$
5. $P_1[f] \simeq P_2[f]$.

En somme, si dans un processus complexe l'on remplace un sous-processus par un processus qui lui est équivalent selon une relation de congruence, alors le nouveau processus sera équivalent au processus d'origine. Cette propriété reflète le fait que dans la vie courante, si l'on remplace par exemple une pièce de vélo par une autre pièce avec les mêmes caractéristiques (mais possiblement de marque différente), on doit s'attendre à ce que notre vélo ait le même comportement qu'avant. On note au passage que la relation de bissimulation ainsi que la relation d'équivalence de traces sont des relations de congruence.

2.2.5 Timed-CCS

Plusieurs extensions de CCS ont été développées par différents chercheurs. L'une d'elle, Timed CCS (ou TCCS), voir [13], ajoute à CCS la notion de temps. Le temps est en effet une notion d'une importance non négligeable si l'on a pour objectif de modéliser des systèmes parallèles complexes tels qu'un réseau informatique. Cependant, il ne faut pas perdre de vue que l'ajout du temps à un langage formel augmente considérablement sa complexité et du coup, la complexité de la vérification d'une propriété donnée. Dans cet ouvrage, nous préférons ne pas nous embarquer dans la modélisation de système

tenant compte du temps, ce qui nous obligera à nous concentrer uniquement sur des attaques linéaires, c'est-à-dire uniquement définies par une suite d'actions, et non par leur emplacement temporel.

Certaines attaques qui peuvent se produire dans un réseau nécessitent la notion du temps pour leur définition. Par exemple, on peut penser à une politique spécifiant qu'il ne doit pas y avoir plus de trois tentatives échouées de connexion provenant d'une certaine adresse dans un intervalle temporel de moins d'une minute. En se privant de la notion du temps, on perd donc la possibilité de représenter une certaine classe d'attaques (qui n'est pas négligeable), cependant, on gagne en simplicité de modélisation et de vérification.

2.2.6 Autres algèbres de processus

Il existe une multitude d'algèbres de processus en plus de CCS, et elles permettent toutes de modéliser mathématiquement des systèmes parallèles ou communicants. Les plus connues sont CSP (*Communication Sequencial Processes*) qui fut publiée par Hoare (voir [24]) en 1985 et ASP (*Algebra of Communicating Processes*) qui fut publiée, également en 1985, par Bergstra et Klop (voir [8]). Plusieurs autres algèbres de processus ont été développées dans les années 90 et dans les dernières années.

Pour la suite de cette section, nous n'aborderons pas chacune de ces algèbres (ce qui serait trop lourd), nous allons plutôt nous concentrer sur les algèbres directement inspirées de CCS, en particulier nous aborderons le π-calcul et le calcul *ambient* . Le lecteur désireux de découvrir d'autres algèbres de processus est invité à consulter l'ouvrage de Bersgtra (voir [9]) qui regroupe plusieurs articles traitant de différents aspects des algèbres de processus.

2.3 Le π-calcul

Le π-calcul a été introduit à la fin des années 80 par Milner et Parrow (voir [36, 37] ainsi que [48]). Ce calcul se base principalement sur CCS, sans toutefois en être une extension. Le π-calcul améliore considérablement l'aspect communication, déjà présent dans CCS, en permettant le passage non seulement de valeur, mais également de canal de communication, voire même de processus entier. Cette nouveauté amène une notion de mobilité qui n'était pas présente dans le CCS, mais que l'on retrouve dans d'autres

TAB. 2.4 – Syntaxe du π-calcul

Préfixes	α	::=	$\overline{a}x$	Sortie
			$a(x)$	Entrée
			τ	Silence
Agents	P	::=	0	Processus nul
			$\alpha.P$	Préfixe
			$P + P$	Somme
			$P \mid P$	Parallélisme
			if $x = y$ then P	Concordance
			if $x \neq y$ then P	Non concordance
			$(\nu x)P$	Restriction
			$A(y_1, \ldots, y_n)$	Identifiant
Définitions			$A(x_1, \ldots, x_n) \stackrel{\text{def}}{=} P$	(où $i \neq j \Rightarrow x_i \neq x_j$)

calculs développés ultérieurement, dont le calcul *ambient* que nous aborderons dans la prochaine section.

Nous décrivons maintenant la syntaxe du π-calcul ainsi que sa sémantique opérationnelle, ensuite, nous regarderons un petit exemple qui permettra de bien saisir le concept de communication introduit dans ce calcul.

2.3.1 Syntaxe du π-calcul

Le tableau 2.4 donne la syntaxe du π-calcul. Tout comme dans CCS, le processus 0 correspond au processus qui ne peut faire aucune action. Le préfixe $\overline{a}x$, qui est noté $\overline{a}(x)$ en CCS, permet à un processus d'envoyer une donnée x sur un canal a, alors que le préfixe d'entrée $a(x)$ permet de recevoir une donnée provenant du canal a. Dans le cas d'une réception, le symbole x représente une variable qui prendra la valeur de la donnée transmise. Il est à rappeler qu'avec le π-calcul, les données et les canaux peuvent se confondre, ainsi dans les écritures $\overline{a}x$ et $a(x)$, le symbole x peut être un nom de canal, donc, par exemple, $\overline{a}c$ peut correspondre à l'envoi du nom c, le nom d'un canal, sur le canal a. À l'instar de CCS, il existe une action spéciale, nommée action silencieuse et

dénotée par la lettre grecque τ. Cette action, qui proviendra souvent du résultat d'une communication entre deux processus, caractérise un processus capable d'évoluer sans interagir avec l'environnement externe.

Les opérateurs de choix et de parallélisme ont la même signification qu'en CCS. Ainsi, le processus $P + Q$ peut se comporter comme le processus P ou comme le processus Q. Le processus $P \mid Q$ peut pour sa part agir tantôt comme le processus P et tantôt comme le processus Q, par entrelacement d'actions. L'opérateur de parallélisme sert également à permettre des communications entre des processus, un processus pouvant émettre se synchronise avec un processus pouvant recevoir et le tout donne lieu à une communication représentée par l'action silencieuse.

Les opérateurs de concordance **if** ... **then** ne sont pas propre au π-calcul, ceux-ci se retrouvent déjà dans certaines extensions de CCS (voir par exemple [51]). Ces opérateurs permettent de produire des branchements conditionnels. Le processus **if** $x = y$ **then** P se comporte comme le processus P si x et y représentent le même nom et comme le processus 0 sinon, alors qu'au contraire, le processus **if** $x \neq y$ **then** P se comporte comme le processus P si x et y ne représente pas le même nom et comme le processus 0 sinon. L'opérateur de choix peut être utilisé en combinaison avec les opérateurs de concordance, ce qui permet de définir **if** $x = y$ **then** P **else** Q comme un simple raccourci pour représenter **if** $x = y$ **then** $P +$ **if** $x \neq y$ **then** Q. On retrouve également dans la littérature les notations $[x = y]P$ et $[x \neq y]P$ pour dénoter le branchement conditionnel.

L'opérateur de restriction joue dans le π-calcul un rôle analogue à l'opérateur $\setminus$ de CCS, c'est-à-dire qu'il permet de limiter la portée d'un canal de communication. Par exemple, le processus $(\nu a)(\overline{a}x.0 \mid a(x).0)$ peut évoluer par une action silencieuse pour devenir le processus $(\nu a)(0 \mid 0)$. Cependant, le processus $(\nu a)(\overline{a}x.0) \mid a(x).0$ ne peut pas évoluer par une communication, puisque la portée du canal a du processus de gauche est limitée par l'opérateur de restriction. La possibilité de transmettre des noms de canaux de communication permet de modifier, au court d'une communication, la portée d'un canal. Par exemple, considérons le processus CCS suivant : $P \setminus \{a\} \mid Q$. Si ni le processus $P \setminus \{a\}$, ni le processus Q ne peuvent participer à une communication sur le canal a, alors le processus entier $P \setminus \{a\} \mid Q$ ne le peut pas non plus, ce qui serait possible avec le π-calcul, en effet :

$$\begin{array}{rcl} (\nu a)(\overline{b}a.0) \mid b(x).\overline{x}e.0 & \xrightarrow{\tau} & (\nu a)(0) \mid \overline{a}e.0 \\ & \xrightarrow{\overline{a}e} & (\nu a)(0) \mid 0. \end{array}$$

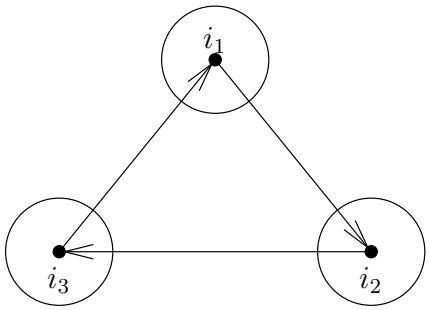
La notation $A(x_1, \ldots, x_n) \stackrel{\text{def}}{=} P$ définit un identifiant A d'arité n. Dans cette définition, les x_i doivent tous être distincts deux à deux. L'identifiant $A(x_1, \ldots, x_n)$ correspond

au processus $P\{y_1/x_1, \ldots, y_n/x_n\}$, qui correspond au processus P dans lequel les occurrences libres de x_1 ont été remplacées par y_1 et de même pour $i = 2, \ldots, n$. Les identifiants permettent donc de représenter plus facilement les processus variables. Par exemple, considérons l'identifiant $H(x, y) \stackrel{\text{def}}{=} \overline{x}d.H + y(d).H$ qui représente un système communiquant avec le canal x sur lequel il peut envoyer et le canal y duquel il peut recevoir des données. Le processus

$$R \stackrel{\text{def}}{=} H(i_1, i_3) \mid H(i_2, i_1) \mid H(i_3, i_2)$$

représente alors le petit réseau en anneau illustré à la figure 2.1.

FIG. 2.1 – Petit réseau en anneau



2.3.2 Sémantique opérationnelle

Le tableau 2.5 donne la sémantique opérationnelle du π-calcul. La majorité des règles du tableau 2.5 ne nécessitent pas d'explication puisqu'elles ne diffèrent guère des règles de CCS. Il serait bon cependant de définir les fonctions bn et fn qui apparaissent dans la règle pour le parallélisme ainsi que dans la règle de restriction.

L'opérateur de restriction (νx) et le préfixe d'entrée $a(x)$ ont pour conséquence de lier des occurrences dans un processus. Ainsi dans les processus $(\nu x)P$ et $a(x).P$ l'occurrence x est liée dans P. L'ensemble $\text{bn}(P)$ représente l'ensemble des occurrences liées dans P, alors que $\text{fn}(P)$ représente l'ensemble des occurrences libres dans P, c'est-à-dire l'ensemble des occurrences non liées dans P. De façon similaire, $\text{bn}(\alpha)$ et $\text{fn}(\alpha)$ correspondent respectivement aux ensembles d'occurrences liées et libres dans un préfixe α. Les appellations bn et fn proviennent de l'anglais *bound names* et *free names*.

Alors qu'en CCS aucune restriction n'est faite quant à l'usage de l'opérateur de parallélisme, le π-calcul exige pour que $P \stackrel{\alpha}{\rightarrow} P'$ implique que $P \mid Q \stackrel{\alpha}{\rightarrow} P' \mid Q$ qu'aucune

TAB. 2.5 – Sémantique opérationnelle du π-calcul

Équivalence	$\dfrac{P' \equiv P,\ P \xrightarrow{\alpha} Q,\ Q \equiv Q'}{P' \xrightarrow{\alpha} Q'}$
Préfixe	$\dfrac{}{\alpha.P \xrightarrow{\alpha} P}$
Choix	$\dfrac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$
Concordance	$\dfrac{P \xrightarrow{\alpha} P'}{\text{if } x = x \text{ then } P \xrightarrow{\alpha} P'}$
Non concordance	$\dfrac{P \xrightarrow{\alpha} P'}{\text{if } x \neq y \text{ then } P \xrightarrow{\alpha} P'}\ x \neq y$
Entrelacement	$\dfrac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \mathrm{bn}(\alpha) \cap \mathrm{fn}(Q) = \emptyset$
Communication	$\dfrac{P \xrightarrow{a(x)} P',\ Q \xrightarrow{\overline{a}u} Q'}{P \mid Q \xrightarrow{\tau} P'\{u/x\} \mid Q'}$
Restriction	$\dfrac{P \xrightarrow{\alpha} P'}{(\nu x)P \xrightarrow{\alpha} (\nu x)P'}\ x \notin \mathrm{bn}(\alpha) \cup \mathrm{fn}(\alpha)$
Ouverture	$\dfrac{P \xrightarrow{\overline{a}x} P'}{(\nu x)P \xrightarrow{\overline{\alpha}\nu x} P'}\ a \neq x$

occurrence liée dans le canal α ne soit libre dans le processus Q. Cette condition additionnelle permet d'empêcher que lors de la communication sur le canal α du processus $P|Q$ avec un tiers processus R, qu'un nom libre dans Q se trouvant accidentellement être le même qu'un nom lié dans α ne soit effecté.

Finalement, l'on remarque que la règle d'ouverture ne possède aucun équivalent en CCS. Cette règle permet d'éliminer la restriction d'un canal à un processus, et transforme du coup une action d'envoi libre $\xrightarrow{\overline{\alpha}x}$ en une action d'envoi lié $\xrightarrow{\overline{\alpha}\nu x}$.

Mots sur l'équivalence de processus

Tout comme cela a été fait pour CCS, toute une théorie a été développée pour définir la congruence de processus, la bissimulation faible, ainsi que la bissimulation forte. Nous invitons le lecteur intéressé à consulter à ce sujet les nombreux documents traitant du π-calcul que nous avons cités dans cette introduction au calcul.

2.3.3 Exemple

Regardons maintenant un petit exemple tiré de [48] qui illustre le fonctionnement du passage de canal disponible avec le π-calcul. La figure 2.2 représente un petit réseau formé d'un serveur, un client et une ressource, ici une imprimante. Avant l'interaction (réseau de gauche) entre le serveur et le client, le client ne dispose d'aucun moyen d'accéder à l'imprimante, alors que le serveur dispose du canal a directement relié à celle-ci. Durant l'interaction, le serveur fournit au client, par le biais du canal b qui les relie, un canal de communication pour accéder à l'imprimante, donc il lui donne le canal a.

En représentant le serveur par le processus $\overline{b}a.S$ et le client par le processus $b(c).\overline{c}d.P$, l'interaction décrite ci-haut se formule ainsi :

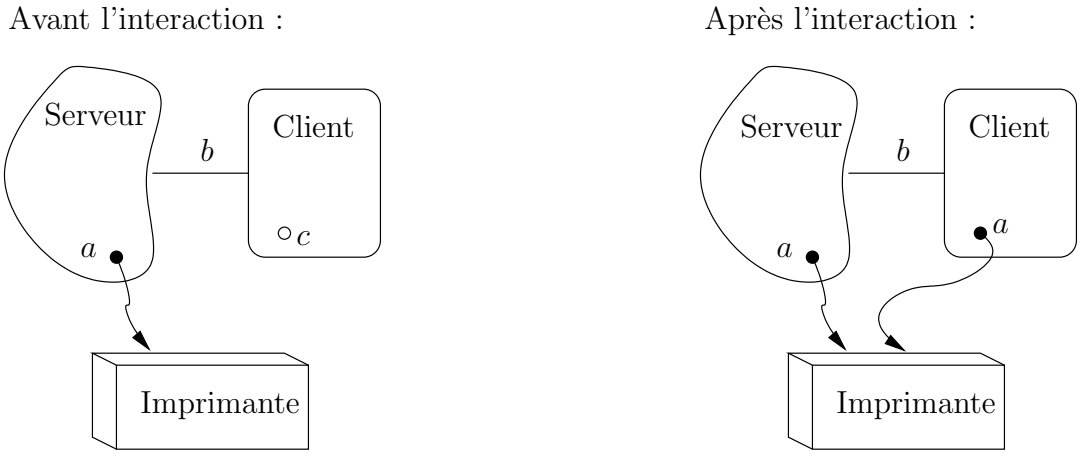
$$\overline{b}a.S \mid b(c).\overline{c}d.P \xrightarrow{\tau} S \mid \overline{a}d.P$$

donc ici $\xrightarrow{\tau}$ est une communication durant laquelle le serveur fournit au client le canal a, par la suite, le client peut utiliser le canal a pour envoyer la donnée d :

$$S \mid \overline{a}d.P \xrightarrow{\overline{a}d} S \mid P$$

Le symbole a joue donc d'abord le rôle d'une donnée transférée, puis celui d'un canal sur lequel il est possible de transférer des données.

FIG. 2.2 – Exemple de passage de canal



2.3.4 Ajouts au π-calcul

Le π-calcul polyadique

Le π-calcul polyadique a été développé par Milner (voir [40]) quelques années après la publication du π-calcul classique (ou monoadique). Ce calcul correspond au π-calcul auquel est ajouté la possibilité de transmettre non seulement une seule valeur, mais plusieurs valeurs lors d'une communication. Ce calcul permet donc le passage de plusieurs valeurs, que l'on peut voir comme le passage d'un vecteur.

Pour deux vecteurs $\tilde{x}$ et $\tilde{y}$, le π-calcul polyadique permet donc l'évolution suivante :

$$a(\tilde{x}).P \mid \overline{a}\langle\tilde{y}\rangle.Q \xrightarrow{\tau} P\{\tilde{y}/\tilde{x}\} \mid Q$$

où $\{\tilde{y}/\tilde{x}\}$ représente la fonction de substitution qui remplace chaque x_i par y_i, les x_i doivent donc être tous différents pour que cette substitution soit bien définie.

Dans cet exemple, le passage de valeurs multiples ne cause aucun problème si les deux vecteurs x_i et y_i sont de même dimension. Par contre, si les deux vecteurs possèdent

des dimensions différentes, la question se pose à savoir comment traiter cette incompatibilité. En fait, les solutions sont multiples et dépendent du contexte précis. Il y a donc lieu de cerner cette incompatibilité dans un système de types, ce qui fait l'objet principal de l'article de Milner.

Le calcul d'ordre supérieur

La première algèbre de processus que nous avons vu, CCS, permet le passage de valeur simple, mais ne permet pas le passage de canal de communication à l'instar du π-calcul. Certaines algèbres, telles que le calcul *ambient* , que nous verrons sous peu, permettent encore plus, soit l'échange de processus entier.

Une version du π-calcul nommée *calcul d'ordre supérieur* permet également le passage de processus entier. L'idée est de considérer des agents variables X et de procéder de façon similaire avec le passage de valeur classique, ce qui donne un préfixe de sortie de la forme $\overline{a}\langle P\rangle$, où P représente un processus, et un préfixe d'entrée de la forme $a(X)$, où X représente un processus variable. Par exemple, le calcul d'ordre supérieur peut donner lieu à l'évolution suivante tirée de [48] :

$$\overline{a}\langle\overline{b}u.0\rangle.b(x) \mid a(X).(X|\overline{c}v) \xrightarrow{\tau} b(x) \mid \overline{b}u.0 \mid \overline{c}v$$

dans laquelle le processus de gauche ($\overline{a}\langle\overline{b}u.0\rangle.b(x)$) envoie le processus $\overline{b}u.0$ au processus de droite par le biais de canal a.

Le passage de processus est une particularité d'une algèbre qui peut sensiblement simplifier certaines modélisations, notamment la modélisation d'un réseau informatique. En effet, on peut considérer que différentes machines dans un réseau échangent des paquets simples, mais on peut également voir ces paquets comme pouvant être dépendant les uns des autres et former ensemble un processus (concrètement, un programme). C'est ce qui se produit par exemple lorsque qu'une machine infectée transmet un virus à une autre machine.

2.4 Le calcul *ambient*

Développé par Cardelli et Gordon (voir [12]) et ayant pour inspiration le potentiel du calcul mobile à travers le *World Wide Web*, le calcul *ambient* a pour objectif de décrire différents aspects du calcul mobile à l'intérieur d'un unique outil de travail.

Le WEB crée de nombreuses et puissantes possibilités, mais également des complications dues à son partitionnement, car il ne s'agit plus aujourd'hui de connaître l'adresse IP d'une machine pour communiquer avec celle-ci. En effet, les pare-feu, omniprésents sur le WEB, séparent ce dernier en de nombreux domaines administratifs. Pour communiquer avec une machine appartenant à un autre domaine administratif, il faut d'abord avoir l'autorisation de quitter notre domaine, puis avoir l'autorisation de pénétrer dans le domaine de cette machine et possiblement obtenir une autorisation additionnelle pour communiquer avec la machine.

De plus, lorsque l'on parle de calculs mobiles, on ne parle plus directement de simples paquets passant d'une machine à une autre, mais bien de processus complets voyageant entre différentes machines. Le calcul *ambient* cadre bien cette réalité en permettant aux processus (*ambients*) de se déplacer.

Un *ambient*, au sens où les auteurs l'entendent, est un emplacement borné dans lequel peuvent se dérouler des calculs ; concrètement, un *ambient* peut représenter par exemple une page web, ou bien un système de fichiers Unix. La propriété importante d'un *ambient* consiste en la frontière qui borne son voisinage. Cette frontière détermine ce qui est interne et ce qui est externe à l'*ambient* ; pour effectuer la migration d'un calcul, il faut être capable de déterminer ce qui doit se déplacer, d'où l'importance de localités bien définies.

Un *ambient* est identifié par un nom, ce nom est utilisé pour contrôler les accès à l'*ambient*, on parle particulièrement des entrées et des sorties. Plusieurs *ambients* peuvent être emboîtés, en somme, un *ambient* peut posséder une collection d'*ambients*. Par exemple, un *ambient* peut représenter un réseau, ce réseau pourrait être composé de plusieurs machines toutes représentées par des *ambients*, de plus, chacune de ces machines pourrait contenir différentes composantes, également représentées par des *ambients*. Finalement, chaque *ambient* possède une collection d'agents locaux, ces derniers, plus simple que les *ambients*, en définissent le comportement, ils peuvent par exemple permettre à l'*ambient* de pénétrer à l'intérieur d'un autre *ambient*. Par analogie, la commande sftp permet à un ordinateur personnel d'entrer dans un réseau donné, à condition, bien sûr, d'en posséder les droits d'accès.

2.4.1 Syntaxe du calcul *ambient*

Dans le calcul *ambient*, l'opérateur $\rightarrow$ est toujours utilisé pour marquer l'évolution d'un processus en un autre processus, les règles régissant ces évolutions sont appelées règles de réduction. Ainsi, lorsque l'on note $P \rightarrow Q$, cela indique que le processus P évolue en

TAB. 2.6 – Syntaxe du calcul *ambient*

(a) Processus

P	::=	$(\nu n)P$	restriction
	|	0	inactivité
	|	$P \mid P'$	composition
	|	$!P$	réplication
	|	$N[P]$	*ambient*
	|	$M.P$	action
	|	$\langle M \rangle$	sortie d'une capacité
	|	$\langle\langle N \rangle\rangle$	sortie d'un nom
	|	$(x).P$	entrée d'une capacité
	|	$((u)).P$	entrée d'un nom

(b) Capacités

M	::=	*in* N	entrer dans N
	|	*out* N	sortir de N
	|	*open* N	ouvrir N
	|	x	variable
	|	ϵ	chemin vide
	|	$M.M'$	chemin

(c) Noms

N	::=	n	nom
	|	u	variable

le processus Q. La syntaxe du calcul *ambient* est présentée au tableau 2.6.

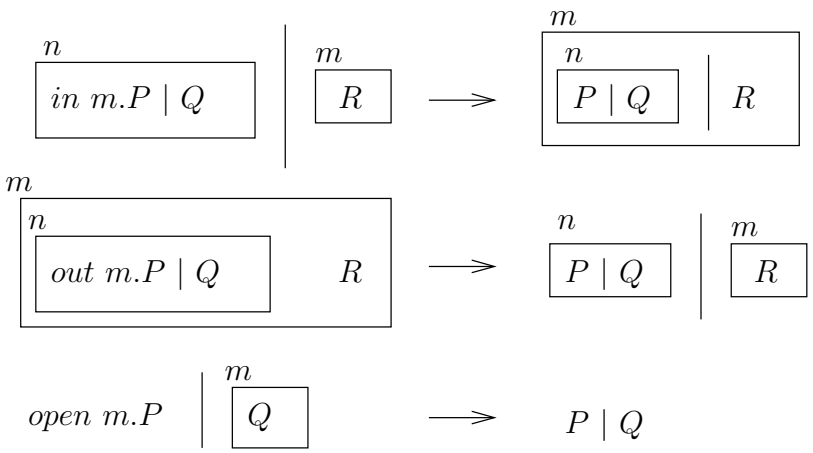
À l'instar de CCS et du π-calcul, le processus 0 dénote le processus inerte. De même, les opérateurs (νn) et | désignent, tout comme dans le π-calcul, les opérateurs de restriction et de parallélisme respectivement. À noter cependant que l'opérateur | n'est plus un opérateur de communication.

L'opérateur de réplication ! se retrouve parfois dans certaines extensions de CCS ou du π-calcul. Cet opérateur permet d'obtenir autant de duplications parallèles souhaitées d'un processus donné. Le processus $!P$ est donc équivalent à $P|!P$.

L'opérateur crochet [ ] permet la création d'un *ambient*, la notation $n[P]$ désigne alors un *ambient* nommé n et contenant le processus P. Le processus P peut à son tour être formé de plusieurs *ambients* ou processus en parallèle. Un *ambient* est donc un processus spécial, que l'on peut voir comme un processus possédant un rayon d'action borné, autrement dit, possédant une frontière. Un processus pourra franchir cette frontière s'il possède les capacités requises.

TAB. 2.7 – Règles de réduction

Règle de réduction de base :



Autres règles de réduction :

$$P \to Q \;\Rightarrow\; (\nu n)P \to (\nu n)Q$$

$$P \to Q \;\Rightarrow\; n[P] \to n[Q]$$

$$P \to Q \;\Rightarrow\; P \mid R \to Q \mid R$$

Les capacités

Un processus ayant la forme $M.P$ peut exécuter une action définie par la capacité M et se comporter par la suite comme le processus P. Cette capacité peut permettre au processus P d'entrer dans un *ambient* (*enter*), de sortir d'un *ambient* (*exit*), ou encore de briser les frontières d'un *ambient* (*open*). Un *ambient* peut être vu comme une localité à accès protégé, ce qui fait des capacités des opérations délicates, car elles changent la structure hiérarchique des *ambients* , et ceci peut se comparer à traverser un pare-feu ou à décoder un message crypté. Voici, plus formellement, le fonctionnement des capacités ainsi que leurs règles de réduction :

enter : La capacité *in* m permet d'entrer dans un *ambient* nommé m, la règle de réduction pour cette capacité est :

$$n[in\ m.P \mid Q] \mid m[R] \ \rightarrow\ m[n[P \mid Q] \mid R].$$

Dans cette évolution, l'*ambient* n utilise sa capacité *in* m pour pénétrer dans l'*ambient* m.

exit : Cette capacité est l'opposée de la précédente, *out* m permet de sortir d'un *ambient* nommé m. La règle de réduction associée est :

$$m[n[out\ m.P \mid Q] \mid R] \ \rightarrow\ n[P \mid Q] \mid m[R].$$

open : Cette dernière capacité, *open* m, permet de briser la frontière délimitant un *ambient* nommé m, la règle de réduction définissant cette action est :

$$open\ n.P \mid n[Q] \ \rightarrow\ P \mid Q.$$

Pour chacune de ces capacités, la réduction lui étant associée ne peut se faire que s'il se trouve un *ambient* nommé m correctement localisé (en fonction de la règle), dans le cas contraire, l'opération est bloquée jusqu'à ce qu'un tel *ambient* existe. Le tableau 2.7 présente les règles de réduction des différentes capacités en représentant les *ambients* par des boîtes au lieu des crochets, ce qui permet une meilleure visualisation de l'opération exécutée.

Comme les auteurs du calcul *ambient* le soulignent dans leur article, ce calcul peut être employé pour représenter des pare-feu, et donc pour vérifier des propriétés de sécurité sur un réseau. Cet aspect du calcul *ambient* nous intéresse particulièrement, nous consacrons donc la suite de cette brève introduction à un aperçu de quelques articles scientifiques approfondissant le sujet.

2.4.2 Applications du calcul *ambient*

2.4.3 Validation de pare-feu

L'application du calcul *ambient* à l'étude des pare-feu a été approfondie par Neilson et ses collègues (voir [47]). Dans cet article, les auteurs utilisent le calcul *ambient* pour représenter un pare-feu ainsi qu'un agent voulant le traverser, tout comme cela est fait dans [12]. Cependant, dans [12], seul l'aspect de s'assurer qu'un agent possédant la bonne forme pourra traverser le pare-feu est abordé, alors qu'il faut également s'assurer qu'un agent ne possédant pas cette caractéristique ne pourra pas traverser le pare-feu.

Pour ce faire, les auteurs introduisent une distinction entre un *nom* et un *nom stable*, distinction qui se compare à la celle entre une adresse internet sous la forme www.ift.ulaval.ca et l'adresse absolue 132.203.26.3, dans ce cas, le nom de domaine (www.ift.ulaval.ca) pourrait être modifié tout en correspondant à la même adresse absolue.

À cette fin, les auteurs développent une analyse de flux de contrôle ayant pour objectif d'obtenir pour chaque *ambient* P les informations suivantes :

– Quels *ambients* peuvent immédiatement être contenu dans P ;
– Quels transitions P peut-il exécuter ;

Les auteurs montrent finalement que leur analyse peut être employée pour concevoir un test permettant de déterminer si un pare-feu donné est fiable.

Contrôle de ressource

Teller et ses collaborateurs se sont intéressés (voir [52]) à l'analyse du contrôle de ressources, où les auteurs considèrent une ressource comme étant une entité pouvant être acquise, utilisée, puis libérée ; on peut alors naturellement penser à un serveur d'impression ou n'importe quelle application distribuée. L'objectif est de fournir un moyen de prévenir certaines attaques, on pense particulièrement aux dénis de service, qui peuvent nuire au bon fonctionnement des systèmes à ressources partagées.

Pour ce faire, les auteurs ont développé un calcul, qui est en fait une extension du calcul *ambient*, pour modéliser des *ambients* contrôlés (*controlled ambients* CA). Dans un *ambient* contrôlé, chaque déplacement est sujet à un contrôle effectué sur trois parties

différentes : l'*ambient* qui se déplace, l'*ambient* qui reçoit un nouveau sous-*ambient*, et l'*ambient* qui laisse partir un de ses sous-*ambients*. Ces contrôles se font grâce aux cinq nouvelles capacités ajoutées à l'algèbre :

$\overline{\text{in}}_{\uparrow} m$ **:** permet à un *ambient* m provenant d'un sous-*ambient* d'entrer ;

$\overline{\text{in}}_{\downarrow} m$ **:** permet à un *ambient* m provenant d'un *ambient* parent d'entrer ;

$\overline{\text{out}}_{\uparrow} m$ **:** permet à m de quitter l'*ambient* où il se trouve en sortant de celui-ci ;

$\overline{\text{out}}_{\downarrow} m$ **:** permet à m de quitter l'*ambient* où il se trouve en entrant dans un de ses sous-*ambients* ;

$\overline{\text{open}}\{m, h\}$ **:** permet à l'*ambient* parent h d'ouvrir l'*ambient* courant m.

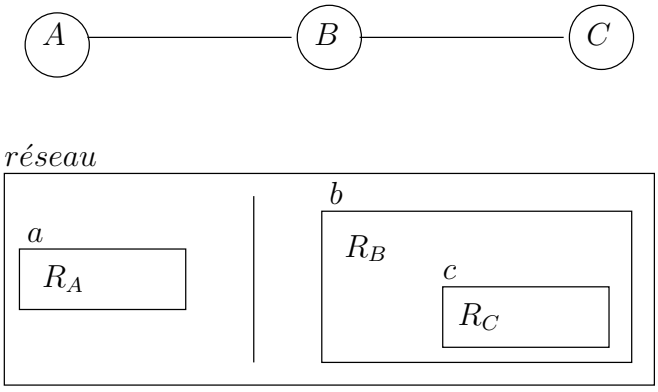
Une première application donnée par les auteurs est une amélioration de l'exemple de pare-feu fourni dans [12]. D'autres exemples sont fournis dans l'article, ainsi qu'un système de type pour le contrôle statique de ressources.

Vérification de pare-feu distribués

Les réseaux informatiques actuels sont constitués non seulement de plusieurs machines, mais également de plusieurs sous-réseaux ; des pare-feu doivent alors être installés pour contrôler le trafic entre ces différents sous-réseaux. Comme la topologie d'un réseau se complexifie avec le nombre d'interconnexions, plusieurs chemins peuvent reliés deux machines appartenant à deux sous-réseaux différents, ainsi un paquet passant d'une machine A vers une machine B peut passer au travers de pare-feu différents d'un autre paquet effectuant le même parcours. Il y a donc nécessité de s'assurer que les politiques de sécurités implantées dans les différents pare-feu ne soient pas contradictoires. Adi et ses collaborateurs (voir [1]) se sont intéressés au problème de la vérification des pare-feu distribués. Dans cet article, les auteurs ont utilisé le calcul *ambient* pour représenter un réseau ainsi que plusieurs pare-feu instaurant une politique de sécurité dans ce réseau dans le but de vérifier l'implantation de la politique de sécurité.

Les auteurs ont étudié la modélisation de réseaux avec le calcul *ambient* et ont défini la notion de vue pour un *ambient* . Pour bien comprendre, reprenons l'exemple de l'article ; considérons un petit réseau constitué de trois sous-réseaux : A, B et C placés en ligne (voir la figure 2.3). Les paquets voyageant de C vers A doivent traverser le réseau B, tout comme les paquets faisant le chemin inverse, donc pour A, le réseau C peut être considéré comme étant un sous-réseau du réseau B, ce qui amène la modélisation de la figure 2.3. Inversement, pour C, c'est le réseau A qui est un sous-réseau de B, donc, différentes modélisations du réseau sont possibles selon la vue que l'on choisit.

FIG. 2.3 – Illustration de la notion de vue



Une seconde contribution de l'article concerne la définition des macros $up < ab >= out\ a.in\ b$ et $down < ab >= in\ a.in\ b$. La macros $up < ab >$ représente la capacité pour un *ambient* de quitter l'*ambient* a dans lequel il se trouve pour entrer dans l'*ambient* b qui se trouve au même niveau que a, alors que la macros $down < ab >$ représente la capacité pour un *ambient* d'entrer dans l'*ambient* a, de même niveau, pour ensuite entrer dans b, un sous-*ambient* de a. Ces capacités se généralisent pour des chemins plus longs et s'adaptent à la notion de vue, de plus, elles permettent aux auteurs de représenter des propriétés de sécurité locales et d'analyser le comportement des processus pour rechercher des possibles incompatibilités.

2.5 Les automates

Les algèbres de processus ne sont pas le seul outil disponible pour représenter et analyser des systèmes communicants. Dans l'article [4], les auteurs se sont intéressés au problème du renforcement des politiques de sécurité dans un programme d'application devant contrôler une machine mécanique, soit un robot. Nous présentons dans cette section un résumé de la méthode développée par les auteurs, méthode qui se base sur des résultats généraux de la synthèse de contrôleur (voir [49]) ainsi que sur des résultats publiés par Maraninchi et Rémond (voir [35]) traitant des automates synchronisés. Les auteurs se sont penchés sur l'utilisation des techniques de synthèse de contrôleurs pour générer une interface entre le robot et le programme qui aura pour tâche de s'assurer que le programme ne dicte pas au robot des tâches brimant la politique de sécurité préalablement définie. Visiblement, un parallèle peut se faire entre le renforcement des politiques de sécurité pour le contrôle d'un robot et le renforcement des politiques de

sécurité dans un réseau, d'où l'intérêt pour nous de jeter un oeil sur cet ouvrage.

Nous présentons pour la suite de cette section, dans un premier temps les outils théoriques utilisés dans l'article [4] puis, dans un second temps, nous traiterons plus directement le sujet principal de l'article, soit le renforcement de politiques de sécurité par génération d'une interface.

2.5.1 Les outils théoriques

Les auteurs travaillent avec des automates qui permettent de modéliser un système communicant ; dans les faits, l'un des automates représentera l'interface faisant respecter les politiques de sécurité, alors qu'un autre automate représentera le robot. Ces deux systèmes doivent communiquer pour évoluer conjointement. Pour ce faire, une communication entre deux automates sera définie en faisant premièrement un produit synchrone des deux automates, puis en exécutant une encapsulation qui forcera la communication en éliminant les transitions correspondant à des communications incomplètes.

Définition 2.5.1 (Automate)**.** Un automate $\mathcal{A}$ est un tuple $\mathcal{A} = (\mathcal{Q}, s_{init}, \mathcal{I}, \mathcal{O}, \mathcal{T}, \mathcal{W})$ tel que $\mathcal{Q}$ est un ensemble d'états, $s_{init} \in Q$ est l'état initial, $\mathcal{I}$ et $\mathcal{O}$ sont respectivement des ensembles de variables booléennes d'entrée (Input) et de sortie (Output), $\mathcal{T} \subseteq \mathcal{Q} \times \mathcal{B}ool(\mathcal{I}) \times 2^{\mathcal{O}} \times \mathcal{Q}$ est l'ensemble des transitions et $\mathcal{W} : \mathcal{Q} \longrightarrow \mathbb{N}$ est une fonction qui étiquette chaque état par un poids. $\mathcal{B}ool(\mathcal{I})$ dénote l'ensemble des formules avec variables prises dans $\mathcal{I}$ où les formules sont construites par conjonction et négation. Des exemples d'automates sont donnés à la figure 2.4.

Une transition est définie naturellement comme étant un tuple (s, ℓ, O, s') dans lequel s et s' correspondent respectivement aux états source et destination, $\ell \in \mathcal{B}ool(\mathcal{I})$ est la condition à respecter pour effectuer la transition et $O \subseteq \mathcal{O}$ est l'ensemble des sorties émises lorsque la transition est effectuée. L'ensemble de toutes les traces d'un automate donné $\mathcal{A}$ sera noté $Trace(\mathcal{A})$. On entend par trace une suite de tuples $\{(v_i, O_i, S_i)\}_i$ telle que $s_1 = s_{init}$, les O_i sont des ensembles de sorties, et $(s_n, \ell, O_n, s_{n+1})$ est une transition valide de l'automate avec $v_n(\ell) = true$ pour tout n.

Un automate est dit :
- **réactif** si pour tout état s de celui-ci, peu importe l'état des variables d'environnement, il existe une transition (s, ℓ, O, s') dont les variables d'environnement respectent les conditions de transition.
- **déterministe** si pour chaque état s de celui-ci, s'il existe deux transitions partant de s, (s, ℓ_1, O_1, s_1) et (s, ℓ_2, O_2, s_2), telles que $\ell_1 = \ell_2$, alors on doit également avoir

$O_1 = O_2$ et $s_1 = s_2$.
Nous aurons besoin de ces notions puisque les auteurs s'intéressent à l'élaboration d'un programme, ce qui correspond à un automate réactif et déterministe.

Le produit synchrone de deux automates est défini de façon similaire au produit synchrone de deux automates de Büchi. Cependant, ici les automates gèrent les entrées et sorties. Cette gestion se fait de façon naturelle dans le produit synchrone qui fera une conjonction des conditions d'entrées et une union des ensembles de sorties. Pour ce qui est des étiquettes de poids, il y a différentes possibilités, l'une d'elle est de simplement les additionner. Voici, formellement, la définition du produit synchrone :

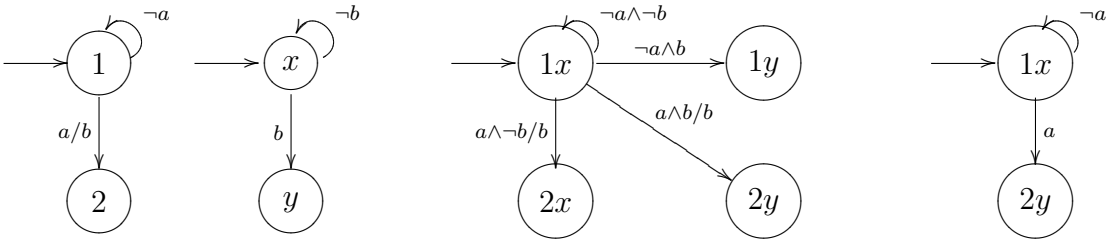
Définition 2.5.2 (Produit synchrone)**.** Soit $A_1 = (\mathcal{Q}, s_{init1}, \mathcal{I}_1, \mathcal{O}_1, \mathcal{T}_1, \mathcal{W}_1)$ et $A_2 = (\mathcal{Q}, s_{init2}, \mathcal{I}_2, \mathcal{O}_2, \mathcal{T}_2, \mathcal{W}_2)$ deux automates. Le produit synchrone de $\mathcal{A}_1$ et $\mathcal{A}_2$ est défini par l'automate $\mathcal{A}_1 \| \mathcal{A}_2 = (\mathcal{Q}_1 \times \mathcal{Q}_2, (s_{init_1} s_{init2}), \mathcal{I}_1 \cup \mathcal{I}_2, \mathcal{O}_1 \cup \mathcal{O}_2, \mathcal{T}, \mathcal{W})$ où $\mathcal{T}$ est défini par : $(s_1, \ell_1, O_1, s_1') \in \mathcal{T}_1 \wedge (s_2, \ell_2, O_2, s_2') \in \mathcal{T}_2 \iff (s_1 s_2, \ell_1 \wedge \ell_2, O_1 \cup O_2, s_1' s_2') \in \mathcal{T}$ et $\mathcal{W}$ est défini par : $\mathcal{W}(s_1 s_2) = \mathcal{W}_1(s_1) + \mathcal{W}(s_2)$.

Suite au produit synchrone, l'encapsulation est utilisée, à l'instar de la restriction dans CCS [38], pour forcer la communication, c'est-à-dire que si l'on fait le produit de deux automates et que l'on encapsule le résultat avec une variable a, une transition qui exige a dans l'automate produit ne sera gardée que si a se retrouve également en sortie dans cette transition (dans ce cas, on élimine les a pour rendre la communication transparente). De même, une transition qui rejette a en entrée sera éliminée si elle possède a en sortie.

Définition 2.5.3 (Encapsulation)**.** Soit $\mathcal{A} = (\mathcal{Q}, s_{init}, \mathcal{I}, \mathcal{O}, \mathcal{T}, \mathcal{W})$ un automate et $\Gamma \subseteq \mathcal{I} \cup \mathcal{O}$ un ensemble d'entrées et de sorties de $\mathcal{A}$. L'encapsulation de $\mathcal{A}$ en fonction de Γ est donnée par l'automate $\mathcal{A} \setminus \Gamma = (\mathcal{Q}, s_{init}, \mathcal{I} \setminus \Gamma, \mathcal{O} \setminus \Gamma, \mathcal{T}', \mathcal{W})$ où $\mathcal{T}'$ est défini par : $(s, \ell, O, s') \in \mathcal{T} \wedge \ell^+ \cap \Gamma \subseteq \ \wedge \ell^- \cap \Gamma \cap O = \emptyset \iff \exists \Gamma.\ (s, \ell \bmod \Gamma, O \setminus \Gamma, s') \in \mathcal{T}'$. Les ensembles ℓ^+ et ℓ^- sont respectivement définis comme étant les ensembles des entrées qui apparaissent positivement et négativement dans ℓ.

La figure 2.4 montre un exemple du produit synchrone de deux automates puis de l'encapsulation du résultat avec une variable, b dans ce cas. Dans cet exemple, la transition allant de $1x$ à $2x$ ne survit pas à l'encapsulation car en entrée elle rejette b et que b se trouve en sortie, alors que la transition de $1x$ à $1y$ est rejetée car en entrée elle exige d'avoir b et qu'elle ne s'y trouve pas en sortie.

FIG. 2.4 – Deux automates, leur produit synchrone et l'encapsulation avec b de celui-ci.



2.5.2 Synthèse de contrôleur

Nous abordons maintenant, dans cette section, plus directement le sujet principal de l'article, soit, pour une machine donnée (un robot) devant être contrôlée par un programme, obtenir une méthode pour s'assurer que les ordres que le programme dicte au robot ne contreviennent pas à sa politique de sécurité. Par exemple, pour un robot constitué d'un bras pouvant tourner et monter ou descendre, cela pourrait être simplement que le bras ne doit pas tourner lorsqu'il est en train de monter ou de descendre.

L'approche de l'article consiste premièrement à modéliser le robot sous la forme d'un automate (défini à la section précédente), puis de modéliser les politiques de sécurité locales pour les différentes parties du robot (les politiques qui ne concernent qu'une partie du robot), puis de générer un contrôleur (un automate spécial) qui fera respecter les politiques globales (celles faisant intervenir plusieurs parties du robot).

Donc, si R est un automate représentant un robot et Φ une formule représentant la politique de sécurité, le but est d'obtenir I_Φ (l'interface), de sorte que pour un programme d'application A :

$$A \| I_\Phi \| R \models \Phi.$$

Si le contrôleur I_Φ existe, il aura la forme $A_1 \| A_2 \| \ldots \| A_n \| C_{\Phi_{global}}$, où les A_i font respecter les politiques propres à une seule composante de R et $C_{\Phi_{global}}$ fait respecter les politiques concernant plusieurs composantes. À remarquer qu'en procédant ainsi, les auteurs s'assurent d'une bonne réutilisabilité : l'interface I_Φ est construite indépendamment du programme d'application A.

L'automate représentant le robot doit être construit de sorte à facilité la contrôlabitité de celui-ci. L'automate possédera donc des entrées contrôlées sur certaines transitions, par exemple, si on ajoute $\wedge OK$ aux entrées d'une transition, celle-ci ne sera gardée dans le produit avec le contrôleur que si ce dernier émet OK en sortie à la transition correspondante. En somme, les entrées de l'automate initial sont séparées en deux ensembles

$\mathcal{I}_u$ et $\mathcal{I}_c$, le premier représente les entrées non contrôlables, alors que le second représente les entrées contrôlables par le contrôleur. Un contrôleur est alors défini comme suit :

Définition 2.5.4. Un contrôleur pour un automate $\mathcal{A} = (\mathcal{Q}, s_{init}, \mathcal{I}, \mathcal{O}, \mathcal{T}, \mathcal{W})$ est un automate $\mathcal{C} = (\mathcal{Q}, s_{init}, \mathcal{I}_u, \mathcal{O} \cup \mathcal{I}_c, \mathcal{T}', \mathcal{W}')$ tel qu'il existe $t = (s, \ell_u \wedge \ell_c, \mathcal{O}, s') \in \mathcal{T}$ si et seulement s'il existe $\gamma \in \mathcal{I}_c$ et t' tels que $t' = (s, \ell_u, \mathcal{O} \cup \gamma, s') \in \mathcal{T}'$. Où ℓ_u est construite avec des variables de $\mathcal{I}_u$ et ℓ_c est construite avec des variables de $\mathcal{I}_c$.

Suivant cette définition, plusieurs contrôleurs peuvent être obtenus à partir d'un automate donné. Tous ces contrôleurs ne possèdent pas le même degré de déterminisme ou de réactivité. Cependant, sur ces points, les contrôleurs possèdent les propriétés suivantes :

Proposition 2.5.1. *Soit $\mathcal{A} = (\mathcal{Q}, s_{init}, \mathcal{I}, \mathcal{T}, \mathcal{W})$ un automate réactif et déterministe et $\mathcal{C}$ un contrôleur pour $\mathcal{A}$, alors :*

1. $Trace((\mathcal{A} \| \mathcal{C}) \setminus \mathcal{I}_c) \subseteq Trace_{\mathcal{I}_u}(\mathcal{A})$.
2. *Si $\mathcal{A}$ est réactif, alors $\mathcal{C}$ l'est aussi.*
3. *$(\mathcal{A} \| \mathcal{C}) \setminus \mathcal{I}_c$ est réactif (même si $\mathcal{C}$ ne l'est pas) et déterministe si $\mathcal{C}$ est déterministe.*

Où $Trace_{\mathcal{I}_u}(\mathcal{A})$ correspond à une trace de $\mathcal{A}$ dans laquelle on ne tient compte que des variables de $\mathcal{I}_u$. Le point 1 signifie donc que chaque trace de l'automate $\mathcal{A}$ contrôlé par $\mathcal{C}$ est aussi une trace de $\mathcal{A}$ sans tenir compte des valeurs de $\mathcal{I}_c$.

2.5.3 Politiques de sécurité

Les politiques de sécurité sont définies à l'aide d'une logique linéaire : CTL [15]. Les deux propriétés qui intéressent les auteurs sont, pour un ensemble d'état $S \in \mathcal{Q}$:

Invariance de S : $\Phi = AG(S)$. Un automate satisfait $AG(S)$ si pour toute trace possible, chaque état de la trace appartient à l'ensemble S.

Atteignabilité de S : $\Phi = EF(S)$. Un automate satisfait $EF(S)$ s'il existe une trace $t = s_0 s_1 s_2 \ldots$ et un état $s \in S$ tels que $s = s_i$.

La première formule est vraie si l'ensemble des états atteignables est restreint à S, alors que la seconde est vraie si l'ensemble S est atteignable. On peut alors, pour revenir à notre exemple précédent, définir l'ensemble S comme étant l'ensemble des états dans lesquels le bras du robot tourne en même temps de monter ou descendre, l'automate respectera la politique que si S n'est pas atteignable, donc seulement si la formule CTL $\neg EF(S)$ est satisfaite.

Pour l'élaboration des automates représentant les politiques de sécurité, les auteurs classent premièrement ces politiques selon qu'elles touchent une seule composante ou plusieurs composantes. Ce qui donne $\Phi = \Phi_1 \wedge \Phi_2 \wedge \cdots \wedge \Phi_n \wedge \Phi_{global}$. Les automates $A_1, A_2, \ldots, A_n$ sont construits sans difficulté pour faire respecter les formules simples. La tâche de générer l'automate faisant respecter Φ_{global}, plus difficile, est laissée à la synthèse de contrôleurs, ce qui peut mener à un automate non déterministe.

2.5.4 Réduction du non-déterminisme

Deux méthodes sont proposées pour réduire ou éliminer le non-déterminisme des contrôleurs.

Une première approche, dite statique, consiste à exploiter la composante de poids des automates. Ainsi, différents poids seront donnés aux états du contrôleur, on choisira alors, par exemple, la transition menant à l'état de poids le plus faible en cas de non-déterminisme.

La seconde approche, dite dynamique, consiste à ajouter des entrées, appelées *oracles*, aux transitions d'un contrôleur. Ainsi, si deux entrées sont étiquetées par la même entrée ℓ, l'une deviendra $\ell.i$ et l'autre $\ell.\bar{i}$, où i correspond à l'oracle. Cela donnera un automate déterministe pour lequel les variables de l'oracle doivent être déterminées par l'environnement.

2.6 Vérification formelle

Nous venons de voir différentes techniques pour modéliser des systèmes communicants à travers une classe complète de langages, les algèbres de processus, ainsi que les automates, en nous concentrant particulièrement sur les algèbres de processus dérivées de CCS. Sans trop insister sur le sujet, qui n'est pas la partie centrale de ce mémoire, nous croyons bon de mentionner quelques mots pour répondre aux questions suivantes : pourquoi voulons-nous faire ces modélisations ? qu'est-ce que cela peut nous apporter ?

En fait, les algèbres de processus, ainsi que les autres langages formels (automates, réseaux de Petri), permettent d'obtenir une modélisation mathématique d'un système concret comme un réseau informatique. Cette modélisation nous permet de voir le système avec un niveau d'abstraction permettant de focaliser notre attention sur des détails

précis du système. Par exemple si l'on veut traiter des suites séquentielles d'actions, nul besoin n'est de tenir compte du temps, ce que l'on veut savoir concerne seulement si une action donnée est avant ou après certaines autres. Cette abstraction permet de simplifier le système de sorte à pouvoir plus aisément vérifier si le système respecte certaines propriétés que l'on énonce dans une politique de sécurité.

Sur un ordinateur donné, les politiques de sécurités peuvent concerner, par exemple, les commandes qu'un usager a le droit d'utiliser ou les fichiers auxquels il a accès. De plus, si cette machine est reliée à un réseau, ce qui est aujourd'hui la norme, les politiques de sécurité peuvent restreindre les paquets pouvant entrer et sortir de la machine. Dans un réseau, des pare-feu peuvent être installés pour faire respecter une politique de sécurité, ceux-ci contrôlent les paquets entrant et sortant.

Donc, ayant en main une politique de sécurité et un système (un réseau), l'on modélise le système à l'aide d'un langage formel, par exemple une algèbre de processus, pour nous donner les outils mathématiques nécessaires pour mettre en évidence des caractéristiques du système et vérifier de façon sûre si le modèle respecte la politique de sécurité. En supposons que le modèle est cohérent avec le système, on en déduira que celui-ci respecte également la politique.

Dans la suite de cette section, nous décrivons brièvement ce qu'est une propriété de sécurité, en discutant des deux grandes classes de propriété : les propriétés de sûreté et et les propriétés de vivacité. Cette introduction en la matière s'inspire fortement de l'article de Schneider (voir [50]) et de l'article de Bauer et collaborateurs (voir [7]). Suite à cela, nous donnons un aperçu des outils employés pour spécifier une politique de sécurité : les logiques.

2.6.1 Classement des propriétés de sécurité

Une politique de sécurité dicte un ensemble de règles qu'un système (une machine, un robot, un programme, ...) doit respecter. Ces règles peuvent concerner différents aspects du système, on peut parler par exemple de classes de politiques :

- Politique de contrôle d'accès : spécifie que certains accès à des ressources (fichiers, serveurs) sont contrôlés ;
- Politique de disponibilité : politique qui gère la disponibilité d'une ressource, une telle politique peut spécifier par exemple qu'un programme doit rendre une ressource un certain temps après l'avoir accaparée ;
- Politique de contrôle de flux : politique qui concerne les entrées et les sorties d'un

programme.

De façon formelle, une politique de sécurité est un prédicat P sur un ensemble d'exécutions. C'est-à-dire qu'un ensemble Σ d'exécutions satisfait une politique donnée P si et seulement si $P(\Sigma)$. On s'intéresse ici aux politiques de sécurité qui traitent les exécutions indépendamment, on parle alors de propriété de sécurité. Dans ce cas, un ensemble d'exécutions Σ respecte une politique P si et seulement si chacune des exécutions σ respecte un prédicat $\hat{P}$, ce qui donne en écriture mathématiques :

$$P(\Sigma) = \forall \sigma \in \Sigma . \hat{P}(\sigma).$$

Suivant Lamport (voir [32, 2]), une attention particulière est portée à deux types de propriétés de sécurités : les propriétés de sûreté (*safety properties*) et les propriétés de vivacité (*liveness properties*). De façon informelle, ces deux classes de propriétés de sécurité s'énoncent ainsi :

Propriétés de sûreté : Il n'y a rien de mauvais qui peut arriver.

Propriétés de vivacité : Quelque chose de bon arrivera.

Il est également possible de définir plus formellement ces propriétés. Pour σ et σ' deux traces d'exécution, on dénote par $\sigma' \prec \sigma$ le fait que σ' soit un préfixe de σ. Avec ces notations, nous pouvons construire les définitions suivantes :

Définition 2.6.1 (Propriétés de sûreté)**.** $\hat{P}$ est une propriété de sûreté si et seulement si pour tout $\sigma \in \Sigma$

$$\neg \hat{P}(\sigma) \Rightarrow \forall \sigma' \in \Sigma . (\sigma \prec \sigma' \Rightarrow \neg \hat{P}(\sigma)).$$

Définition 2.6.2 (Propriétés de vivacité)**.** $\hat{P}$ est une propriété de vivacité si et seulement si

$$\forall \sigma \in \Sigma . \exists \sigma' \in \Sigma . (\sigma \prec \sigma' \hat{P}(\sigma)).$$

On comprend par ces définitions que les propriétés de sûreté sont caractérisées par le fait que si une trace ne respecte pas une politique de sûreté P, alors aucune extension de cette trace ne respectera P. Alors que les propriétés de vivacité sont caractérisées par le fait que si une trace satisfait une propriété de vivacité P, alors il existe forcément une extension fini de cette trace qui respecte également P.

L'importance de ces deux classes de propriétés de sécurité devient évidente en prenant connaissance d'un résultat de Alpern et Schneider (voir [3]) qui affirme que toute propriété de sécurité peut être décomposée en une conjonction de propriétés de sûreté et de propriétés de vivacité.

2.6.2 Implantation de propriétés de sécurité

Schneider (voir [50]), puis Bauer, qui s'est inspiré des travaux de Schneider (voir [7]) se sont intéressés au problème de déterminer quel type de propriétés de sécurité peuvent être imposées à un système par ajout de moniteurs. Donc à partir d'un système et d'une politique de sécurité, politique qui n'est pas nécessairement respectée par le système, ils se sont demandés s'il était possible d'ajouter des moniteurs au système de sorte à ce qu'il respecte la politique.

Pour rendre la question plus limpide, il importe de spécifier le rôle des moniteurs. Dans son article, Schneider considère la classe de système EM (pour **E***xecution* **M***onitoring*), dans cette classe, les systèmes évoluent étape par étape et le moniteur arrête le déroulement du système si l'étape en court mène à un état qui brime la politique de sécurité. Les résultats de Schneider sont les suivants :

- Le mécanisme d'imposition EM permet uniquement d'imposer des politiques de sécurité qui sont des propriétés de sûreté.
- Toutes les propriétés de sûreté ne sont pas imposables en utilisant ce mécanisme.

Le premier de ces résultats semble assez intuitif, en effet, si l'on pense à une propriété de la forme « chaque demande d'impression doit être traitée », qui est une propriété de vivacité, il est clair qu'un mécanisme à moniteur comme EM ne peut imposer à un système une telle politique. En effet, les moniteurs ne possèdent que le pouvoir d'arrêter le fonctionnement du système, ils ne possèdent donc pas le pouvoir de forcer un chemin d'exécution.

Dans ce mémoire, nous traitons également de moniteurs, mais ceux-ci auront un fonctionnement différent de ceux retrouvés dans les travaux de Schneider. Nos moniteurs ne posséderont pas le pouvoir d'arrêter le déroulement d'une exécution, mais celui d'absorber (de faire disparaître) des actions, à l'image des pare-feu qui absorbent des paquets circulant sur un réseau. Puisque nos moniteurs ne peuvent pas générer d'action, mais seulement en faire disparaître, on doit donc s'attendre, tout comme pour EM, à ne pouvoir imposer que des propriétés de sûreté, ce qui pour notre étude, n'est pas une contrainte sévère.

2.6.3 Écriture des politiques de sécurité

Nous le répétons, notre objectif est de développer une méthode formelle et vérifiable pour obtenir un réseau sécurisé (par ajout de moniteurs ou pare-feu) à partir d'un réseau et d'une politique de sécurité donnée. Nous avons vu précédemment des outils pour modéliser un réseau ; pour rendre la démarche complètement formelle, nous avons également besoin d'outils mathématiques pour représenter une politique de sécurité. Une classe d'outils ayant déjà fait l'objet de nombreux travaux est particulièrement appropriée pour une telle tâche : les logiques.

Pour terminer ce chapitre sur les méthodes formelles, nous ne fournissons pas un état de l'art sur les logiques, ceci nous éloignerait trop du sujet principal de ce document ; le lecteur intéressé à ce sujet peut se référer aux différents livres traitant du sujet. Nous plaçons cependant quelques mots pour montrer comment une logique bien connue, LTL, peut être utilisée pour définir des propriétés de sécurité. La question à savoir quelle logique est la mieux appropriée pour nos besoins, bien que très intéressante, n'a pas été étudiée dans ces travaux, nous utilisons la logique LTL seulement à titre indicatif.

La logique LTL

La logique LTL (de l'anglais *Linear Temporal Logic*, voir [34, 33]) est l'une des premières logiques temporelles linéaires qui fut développées. Cette logique opère sur une séquence infinie d'états, que l'on nomme trace et que l'on note σ, la notation $\sigma(i)$ dénote le $i^{ème}$ élément de la trace σ. Par exemple, lors de l'exécution d'un processus, cette trace consiste en la suite des actions exécutées.

Tab. 2.8 – Syntaxe de LTL

$$\phi ::= p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \mathrm{X}\phi \mid \phi_1 \,\mathsf{U}\, \phi_2$$

Le tableau 2.8 donne la syntaxe de la logique. Il s'agit en somme de la syntaxe du calcul propositionnel additionnée des opérateurs X (prochain, *neXt* en anglais) et U (jusqu'à ce que, *Until* en anglais). Ces opérateurs font de LTL une logique temporelle, car ils sont influencés par l'ordre des états dans la traces.

Le tableau 2.9 donne la sémantique formelle de LTL. Il n'y a pas de surprise sur la

TAB. 2.9 – Sémantique de LTL

$\sigma(i) \models p$	ssi	$p \in \sigma(i)$
$\sigma(i) \models \neg\phi$	ssi	$\sigma(i) \not\models \phi$
$\sigma(i) \models \phi_1 \vee \phi_2$	ssi	$\sigma(i) \models \phi_1$ ou $\sigma(i) \models \phi_2$
$\sigma(i) \models \mathrm{X}\phi$	ssi	$\sigma(i+1) \models \phi$
$\sigma(i) \models \phi_1 \mathsf{U} \phi_2$	ssi	$\exists k \geq i.\sigma(k) \models \phi_2$ et $\forall i \leq j < k.\sigma(j) \models \phi_1$

définition des opérateurs, on remarque par exemple que la formule $\sigma(i)$ satisfait $\mathrm{X}\phi$ si le prochain état, c'est-à-dire $\sigma(i+1)$ satisfait ϕ, de plus, un état $\sigma(i)$ satisfait $\phi_1 \mathsf{U} \phi_2$ si et seulement si ϕ_2 est satisfaite pour un élément $\sigma(k)$ suivant $\sigma(i)$ et que ϕ_1 est satisfaite par $\sigma(i)$ et par tous les états situés entre $\sigma(i)$ et $\sigma(k)$.

Analyse d'une trace avec LTL

Considérons une trace formée de paquets IP. Pour analyser cette trace avec la logique LTL, il faut premièrement définir des prédicats de base, prédicats que l'on définit en fonction bien sûr de nos besoins précis. Par exemple, on peut définir les prédicats de base suivants :

$TCP(@IP_s, @IP_d)$ **:** un paquet utilisant le protocole IP et ayant pour adresse source $@IP_s$ et comme adresse destination $@IP_d$.

$UDP(@IP_s, @IP_d)$ **:** un paquet utilisant le protocole UDP et ayant pour adresse source $@IP_s$ et comme adresse destination $@IP_d$.

TCP(SYN) **:** un paquet utilisant le protocole TCP et ayant l'étiquette SYN activée.

Une fois en main les prédicats nécessaires, il est possible d'exprimer des caractéristiques d'une trace à l'aide de la logique. Par exemple, la formule

$$\neg\mathrm{true}\,\mathsf{U}\,\mathrm{TCP}(132.203.114.214, 132.203.114.215)$$

sera satisfaite seulement par les traces ne contenant aucun paquet TCP d'adresse source 132.203.114.214 et d'adresse destination 132.203.114.215. Cette écriture peut facilement devenir lourde, c'est pourquoi certaines macros sont souvent bien utiles, dont les macros

Futur (ou nécessairement ou encore *eventually* en anglais) et Globalement (ou toujours) :

$$\begin{aligned} \mathsf{F}\Phi &\equiv \text{true}\,\mathsf{U}\,\neg\Phi \\ \mathsf{G}\Phi &\equiv \neg\mathsf{F}\neg\Phi. \end{aligned}$$

La formule $\mathsf{F}\Phi$ est satisfaite par les traces pour lesquelles la formule Φ devient nécessairement vraie à un certain état de la trace, alors que la formule $\mathsf{G}\Phi$ est satisfaite par les traces dont aucun état ne satisfait la formule $\neg\Phi$, ce qui revient à dire que tous les états satisfont la formule Φ. Avec ces macros, notre formule devient simplement

$$\mathsf{G}\neg TCP(132.203.114.214, 132.203.114.215)$$

qui dit simplement qu'aucun paquet ne pas doit correspondre à un paquet TCP d'adresse source 132.203.114.214 et d'adresse destination 132.203.114.215.

Dernier mots sur l'écriture des propriétés de sécurité

Le travail effectué durant cette maîtrise portait essentiellement sur l'élaboration d'une algèbre de processus permettant d'exprimer naturellement des moniteurs, et non pas sur l'écriture des politiques de sécurité. Nous avons cru bon cependant de glisser quelques mots à ce sujet pour que le lecteur voit clairement la direction que prennent nos travaux, et aussi parce que la question à savoir quelle logique est la mieux appropriée pour définir des politiques de sécurités pour un réseau informatique est des plus intéressantes, elle se devra donc d'être analysée rigoureusement.

2.7 Conclusion

Nous avons parcouru à travers ce chapitre plusieurs langages de spécification de systèmes, dont trois algèbres de processus (CCS, le π-calcul et le calcul *ambient*), ainsi qu'un type d'automate munis d'un produit synchrone. Ces langages peuvent être employés pour modéliser formellement divers systèmes de la vie courante tels qu'un ascenseur, un feu de circulation, un logiciel, ou un réseau informatique. Ces langages permettent d'obtenir une interprétation mathématique d'un système. D'autres langages, les langages de spécification de propriétés, permettent de spécifier formellement les propriétés que l'on souhaite vérifier sur un système ; un vérificateur de modèle s'occupe alors de vérifier si le modèle du système respecte les propriétés.

Dans le prochain chapitre, nous présentons une nouvelle algèbre de processus basée sur CCS. Cette algèbre se distingue par le fait qu'elle possède un opérateur (l'opérateur de

surveillance) qui permet de monitoré directement un processus en contrôlant ses entrées et ses sorties à la manière d'un pare-feu sur un réseau informatique. La configuration des moniteurs d'un processus correspondant à un réseau reflètera donc la configuration des pare-feu placés sur ce réseau.

Chapitre 3

Algèbre pour processus monitorés

Ce chapitre, qui concerne la partie principale de ce mémoire, reprend essentiellement l'algèbre de processus développée par l'auteur et publiée dans [31] en y améliorant le fonctionnement des moniteurs. En début de chapitre, nous présentons notre algèbre de processus, qui est basée sur CCS, en portant une attention particulière à son opérateur de surveillance, qui permet de monitorer directement un processus. Puis nous définissons une relation de bissimulation faible propre à cette algèbre et nous donnons quelques résultats sur l'équivalence de processus. Tout ceci se fera en parcourant des exemples de modélisation de réseau. Finalement, nous présentons une petite étude de cas d'implantation de pare-feu dans un réseau avant de conclure en présentant une version de moniteurs évolutifs, contrairement aux premiers qui seront fixes.

3.1 Syntaxe

L'algèbre développée dans ce chapitre nous permet de représenter des processus contrôlés par des moniteurs. Nous verrons dans cette section, et dans les sections suivantes, les règles de construction de ces processus, ainsi que le fonctionnement des moniteurs.

3.1.1 Les actions et les canaux de communication.

Les processus seront construits d'éléments de base appelés actions élémentaires. Ces actions peuvent décrire la capacité d'un processus à émettre un message sur un canal de communication ou bien la capacité de recevoir un message provenant d'un canal de communication (comme c'est le cas en CCS). Les canaux de communication pourront être identifiés simplement par des lettres de l'alphabet latin ou des mots ($a, b, in, out, ...$). Les actions d'envoi seront identifiées par le canal sur lequel elles agissent, alors que les actions de réception seront identifiées par le nom de ce canal surligné (à noter que nous employons la notation inverse de celle de CCS). Ainsi, un processus pouvant exécuter l'action a peut envoyer un message sur le canal a, alors qu'un processus pouvant exécuter l'action $\overline{a}$ peut recevoir un message provenant du canal a. Les communications entre processus pourront se réaliser lorsqu'un processus sera apte à recevoir un message provenant d'un canal de communication en même temps qu'un second processus sera apte à émettre un message sur ce même canal.

Nous noterons $\mathcal{A}$ l'ensemble des actions d'envoi, $\overline{\mathcal{A}}$ l'ensemble des actions de réception, et l'ensemble de toutes les actions sera noté $\mathcal{A}ct$. Par commodité, nous considérerons que $\overline{\overline{a}} = a$. Pour un processus P donné, l'ensemble $\mathcal{L}(P)$ désignera l'ensemble des actions utilisées pour construire le processus P. Finalement, $\mathcal{P}$ notera l'ensemble de tous les processus.

Nous utiliserons également le passage de valeur (qui sera semblable à celui de CCS avec passage de valeur), de sorte à alléger la notation. Dans ce cas, l'action $a(m)$ représentera l'envoi du message m sur le canal a. Ce message m pourra comporter différents champs, entre autres, il pourra contenir l'adresse d'un hôte auquel le message est destiné. Dans l'étude plus précise des réseaux, ce message représentera alors un paquet IP utilisant un des protocoles de transfert, TCP ou UDP par exemple.

Les opérateurs de communication et de surveillance ($\|$ et $[\ ]$) qui appartiennent à l'algèbre, introduiront d'autres types d'actions qui ne sont pas des actions élémentaires (que l'on ne peut pas utiliser pour construire un processus). Ces actions sont les actions de communication, que nous noterons $c(a)$ et qui correspondent à une communication effectuée sur un canal a, et l'action silencieuse qui sera notée τ ; cette action résultera de l'utilisation de l'opérateur de surveillance qui pourra *absorber* certaines communication ou faire migrer des actions d'un domaine délimité vers un autre.

Nous utiliserons la lettre grecque α pour désigner une action qui pourra être soit une action d'envoi sur le canal a, soit une action de réception sur ce même canal. La lettre grecque β pourra désigner, en plus d'une action d'envoi ou de réception, une communi-

TAB. 3.1 – Syntaxe de l'algèbre

$P ::= 0 \mid \alpha.P \mid P_1 + P_2 \mid P_1 \stackrel{\text{def}}{=} P_2 \mid P_1 \| P_2 \mid [P]_M^N$

cation sur le canal a, finalement la lettre grecque γ pourra désigner en plus une action silencieuse. Nous avons donc :

$$\begin{aligned} &a \in \mathcal{A} \\ &\overline{a} \in \overline{\mathcal{A}} \\ &\alpha \in \mathcal{A}ct \\ &\beta \in \mathcal{A}ct \cup C(\mathcal{A}) \\ &\gamma \in \mathcal{A}ct \cup C(\mathcal{A}) \cup \{\tau\} \end{aligned}$$

où le nouvel ensemble $C(\mathcal{A})$ désigne l'ensemble des communications sur les canaux de $\mathcal{A}$.

Passons maintenant à la syntaxe de l'algèbre, qui est illustrée au tableau 3.1.

3.1.2 Syntaxe de l'algèbre

La sémantique opérationnelle associée aux opérateurs de l'algèbre est donnée aux tableaux 3.2 et 3.5. Avant d'aborder cette sémantique, nous discutons brièvement de la syntaxe de l'algèbre.

Le processus 0 correspond au processus qui ne peut faire aucune action, soit le processus vide. Ce processus représente le processus de base à partir duquel on peut définir d'autres processus pouvant exécuter un nombre fini ou infini d'actions. Les différents opérateurs seront expliqués à la section traitant de la sémantique de l'algèbre, nous parlerons seulement, pour l'instant, du dernier opérateur de la syntaxe, soit l'opérateur de surveillance ($[\ ]_M^N$).

L'opérateur de surveillance crée une sorte de boîte dans laquelle peut opérer un processus. Le processus $[P]_M^N$ possédera le comportement du processus P modulo quelques restrictions et permissions. La lettre N désigne le nom du nouveau processus, de plus, sont associés à ce processus des canaux d'entrée ainsi que des canaux de sortie. Nous désignerons par I et O les fonctions qui, pour un nom donné, retourne l'ensemble des canaux d'entrée et l'ensemble des canaux de sortie respectivement. Finalement, la lettre

M désigne le moniteur qui contrôle les actions pouvant entrer ou sortir de la zône restreinte.

3.1.3 Les moniteurs

Dans cette première version de l'algèbre nous utiliserons des moniteurs dont le comportement sera fixe. Ces moniteurs correspondent en fait à des pare-feu que l'on dit dans la langue anglaise *stateless firewalls*, c'est-à-dire des pare-feu qui traitent chaque paquet individuellement, donc sans considération du temps ou de la trace globale, contrairement aux pare-feu dits *statefull firewalls*, qui peuvent traiter des suites logiques de paquets correspondant à une attaque potentielle en développement.

Comme ces moniteurs sont fixes, ils peuvent être identifiés par une fonction. Nous noterons simplement par f_M la fonction de pare-feu qui prend en entrée un canal de communication et qui rend en sortie une valeur booléenne identifiant si le moniteur M permet (retourne T) ou interdit (retourne F) la communication sur ce canal.

Nous utiliserons cette fonction pour définir les moniteurs. Par exemple, un moniteur M tel que

$$f_M(x) = \begin{cases} F & \text{si } x = a \\ T & \text{sinon} \end{cases}$$

est un moniteur qui interdit la communication sur le canal a et qui permet la communication sur tous les autres canaux.

3.2 Sémantique opérationnelle

3.2.1 Sémantique opérationnelle des opérateurs de base

Le tableau 3.2 donne la sémantique opérationnelle des quatre premiers opérateurs de la syntaxe. La sémantique des trois premiers opérateurs est exactement la même que celle des opérateurs correspondant de CCS. Cependant, la sémantique de l'opérateur de communication diffère de celle de CCS du fait que dans CCS, la communication résulte en une action silencieuse, alors que dans cette algèbre, elle donne plutôt naissance à un nouveau type d'action : les actions de communication. Ceci permet de pouvoir repérer, dans une trace d'exécution, toutes les communications qui ont eu lieu durant l'exé-

cution, cela nous est nécessaire pour analyser les traces d'exécution sur des processus représentant des réseaux ; ces traces d'exécutions représenterons en fait le trafic entre les différentes parties d'un réseau.

TAB. 3.2 – Sémantique opérationnelle des opérateurs de base

$\dfrac{\square}{\alpha.P \xrightarrow{\alpha} P}$	Préfixage
$\dfrac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$	Choix
$\dfrac{P \overset{\text{def}}{=} Q \quad Q \xrightarrow{\alpha} Q'}{P \xrightarrow{\alpha} Q'}$	Définition
$\dfrac{P \xrightarrow{\alpha} P'}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q}$	Entrelacement
$\dfrac{P \xrightarrow{a} P' \quad Q \xrightarrow{\overline{a}} Q'}{P \parallel Q \xrightarrow{c(a)} P' \parallel Q'}$	Communication
$\dfrac{P \equiv P_1 \quad P_1 \xrightarrow{\gamma} P_2 \quad P_2 \equiv P'}{P \xrightarrow{\gamma} P'}$	Équivalence

En plus des opérateurs de base, nous incluons dans la sémantique de notre algèbre la relation d'équivalence $\equiv$. Cette équivalence est définie comme suit :

$$\begin{aligned} P + Q &\equiv Q + P \\ P \parallel Q &\equiv Q \parallel P \\ P + 0 &\equiv P \\ P \parallel 0 &\equiv P \end{aligned}$$

En somme, cette équivalence rend commutatifs les opérateurs de choix et de parallélisme, et elle permet de simplifier le processus 0 lorsqu'il est lié à un autre processus par l'opérateur de choix ou l'opérateur de parallélisme.

Le comportement de ces quatre opérateurs de base est essentiellement le même que celui de leur équivalent dans CCS, nous rappelons donc brièvement le rôle qu'ils jouent.

L'opérateur de préfixage indique que suite à l'exécution de l'action a, le processus $a.P$ devient le processus P. L'opérateur de choix permet d'évoluer selon un processus ou selon un autre, par exemple le processus $P + Q$ peut évoluer comme le processus P ou comme le processus Q. L'opérateur de définition permet principalement la création de processus récursif. Et finalement, l'opérateur de communication permet soit l'entrelacement d'actions, soit l'établissement d'une communication complète entre deux processus. Lorsque nous voudrons mettre une emphase visuelle sur le fait que deux processus donnés ne peuvent pas communiquer ensemble, nous utiliserons le symbole $\mid$ au lieu du symbole $\parallel$ pour dénoter l'opérateur de communication.

Voici quelques exemples d'évolution de processus simple :

$$
\begin{array}{rcll}
a.a.0 + b.0 & \xrightarrow{a} & a.0 & \text{(Choix, Préfixage)} \\
a.a.0 + b.0 & \xrightarrow{b} & 0 & \text{(Choix, Préfixage)} \\
a.a.0 \mid b.0 & \equiv & b.0 \mid a.a.0 & \text{(Équivalence)} \\
 & \xrightarrow{b} & 0 \mid a.a.0 & \text{(Entrelacement, Préfixage)} \\
 & \equiv & a.a.0 & \text{(Équivalence)} \\
a.\overline{b}.0 \parallel \overline{a}.b.0 & \xrightarrow{c(a)} & \overline{b}.0 \parallel b.0 & \text{(Communication, Préfixage)} \\
 & \xrightarrow{c(b)} & 0 \parallel 0 & \text{(Communication, Préfixage)} \\
 & \equiv & 0 & \text{(Équivalence)}
\end{array}
$$

Si $P \stackrel{\text{def}}{=} a.P$ et $Q \stackrel{\text{def}}{=} \overline{a}.Q$, alors

$$P \parallel Q \xrightarrow{c(a)} P \parallel Q \quad \text{(Communication, Définition, Préfixage).}$$

3.2.2 Le passage de valeur

Nous allons utiliser, pour simplifier la définition et la compréhension des processus, le passage de valeur, comme cela se fait avec CCS. La sémantique du passage de valeur est donnée par les règles Entrée et Sortie du tableau 3.3. Le symbole D, utilisé dans la règle d'entrée, dénote l'ensemble dans lequel la variable x prend ses valeurs, alors que la fonction Val, utilisée dans la règle de sortie, représente une fonction de valuation ; l'écriture $\text{Val}(x) = v$ signifie donc que x prend la valeur v.

L'opérateur $\{v/x\}$, qui est utilisé dans les tableaux 3.3 et 3.4, est appelé opérateur de renommage. Cet opérateur provient de la syntaxe de CCS et il a la signification suivante :

TAB. 3.3 – Passage de valeur

$$\text{(Entrée)} \quad \frac{v \in D}{\overline{a}(x).P \overset{\overline{a}(v)}{\rightarrow} P\{v/x\}}$$

$$\text{(Sortie)} \quad \frac{\text{Val}(x) = v}{a(x).P \overset{a(v)}{\rightarrow} P}$$

$$\frac{P \overset{a}{\rightarrow} Q}{E\{f\} \overset{f(a)}{\rightarrow} Q\{f\}}$$

La fonction v/x est la fonction qui prend une entrée a et qui retourne cette entrée si elle est différente de x et retourne v si elle est égale à x. C'est donc une fonction qui remplace x par v et laisse intacte toute les autres valeurs. Nous utiliserons également le passage de valeurs multiples, soit celui décrit dans le tableau 3.4.

TAB. 3.4 – Passage de valeurs multiples

$$\text{(Entrée)} \quad \frac{v_1 \in D_1, \ldots, v_n \in D_n}{\overline{a}(x_1, \ldots, x_n).P \overset{\overline{a}(v_1,\ldots,v_n)}{\rightarrow} P\{v_1/x_1, \ldots, v_n/x_n\}}$$

$$\text{(Sortie)} \quad \frac{\text{Val}(x_1) = v_1, \ldots, \text{Val}(x_n) = v_n}{a(x_1, \ldots, x_n).P \overset{a(v_1,\ldots,v_n)}{\rightarrow} P}$$

Dans le tableau 3.4, les symboles $D_1, \ldots, D_n$ représentent respectivement les ensembles dans lesquels les variable $x_1, \ldots, x_n$ prennent leurs valeurs.

Exemple d'utilisation du passage de valeurs multiples

Bien que, dans le but d'alléger l'écriture, nous omettons les variables dans les actions pour définir les règles de la sémantique opérationnelle, l'utilisation de celles-ci devient pratiquement obligatoire dans l'écriture de modèles de réseaux. Par exemple, considérons deux hôtes, le premier pouvant envoyer un paquet au second par le biais de

l'interface int_1 et le second pouvant envoyer un paquet au premier par le biais de l'interface int_2. On se retrouve par exemple avec la modélisation suivante :

$$\begin{aligned} R &\stackrel{\text{def}}{=} H_1 \parallel H_2 \\ H_1 &\stackrel{\text{def}}{=} int_1.H_1 + \overline{int_2}.H_1 \\ H_2 &\stackrel{\text{def}}{=} int_2.H_2 + \overline{int_1}.H_2. \end{aligned}$$

Si nous voulons maintenant tenir compte de certains éléments contenus dans les paquets, il serait trop lourd de placer en parallèle toutes les actions nécessaires pour représenter les différentes possibilités de contenus des paquets. On a donc recourt au passage de valeur pour obtenir une écriture concise et facile de compréhension pour exprimer plusieurs paquets dans une action. Par exemple, supposons que l'on s'intéresse uniquement aux adresses IP source et destination des paquets, ainsi qu'au protocole de transfert utilisé, par exemple le protocole TCP, UDP, ou ICMP. On obtiendra alors pour H_1 et H_2, les modélisations suivantes :

$$\begin{aligned} R &\stackrel{\text{def}}{=} H_1 \parallel H_2 \\ H_1 &\stackrel{\text{def}}{=} int_1(@_s, @_d, p).H_1 + \overline{int_2}(@_s, @_d, p).H_1 \\ H_2 &\stackrel{\text{def}}{=} int_2(@_s, @_d, p).H_2 + \overline{int_1}(@_s, @_d, p).H_2 \end{aligned}$$

où $@_d$ et $@_s$ prennent leurs valeurs dans l'ensemble des adresses IP, alors que p prend ses valeurs dans l'ensemble des protocoles de transfert $\{TCP, UDP, ICMP, \ldots\}$. Ainsi, l'action de communication $c(int_1(192.168.2.101, 192.168.2.156, TCP))$ indique la transmission d'un paquet partant de H_1 et se rendant à H_2, ce paquet utilise le protocole IP, son adresse source est 192.168.2.101 alors que son adresse destination est 192.168.2.156 ; on ne sait pas cependant si ces adresses correspondent réellement aux adresses de H_1 et H_2.

Définition explicite des ensembles de valeurs

Parfois, l'ensemble des valeurs que pourra prendre une variable sera directement inscrit dans le processus, par exemple, si l'on note

$$H_1 \stackrel{\text{def}}{=} int_1(@_s, @_d, p).H_1 + \overline{int_2}(@_s, @_d, p \in \{TCP, UDP\}).H_1$$

cette modélisation est équivalente à

$$H_1 \stackrel{\text{def}}{=} int_1(@_s, @_d, p).H_1 + \overline{int_2}(@_s, @_d, p_2).H_1$$

en précisant que $p_2 \in \{TCP, UDP\}$. Cette notation devient très pratique lorsque dans un même processus, la même variable pourra prendre ses valeurs dans différents ensembles dépendamment de l'endroit où elle se trouve dans le processus. Un cas particulier est lorsqu'une variable prend sa valeur dans un ensemble possédant un seul élément,

soit lorsqu'il s'agit d'une constante et dans ce cas, la variable sera notée en gras. Par exemple, si dans H_1, un paquet reçu doit utiliser le protocole TCP, on aura

$$H_1 \stackrel{\text{def}}{=} int_1(@_s, @_d, p).H_1 + \overline{int_2}(@_s, @_d, \mathbf{TCP}).H_1$$

Ainsi, toute communication avec H_1 sur le canal int_2 devra nécessairement avoir la forme $c(int_2(@_s, @_d, TCP))$.

Variables et canaux

Lorsque nous n'utilisons pas de passage de valeur, l'action a correspond à la fois à un canal de communication et à l'action d'envoyer un message sur ce canal. Avec le passage de valeur, l'action $a(x)$ se lit maintenant comme l'envoi d'un message x sur le canal a, alors que $a(x, y, z)$ se lit comme l'envoi des messages x, y et z sur le canal a.

Il y a donc maintenant une distinction entre le canal de communication et les messages envoyés sur le canal. Lorsque nous traiterons des moniteurs, soit à la section suivante, il faudra bien distinguer entre ce qui concerne un canal et ce qui concerne les communications sur un canal. Par exemple, les moniteurs posséderont des contrôles d'entrée et de sortie sur les canaux de communication, ces contrôles se font donc indépendamment du message porté par une communication et concernent uniquement le canal sur lequel se fait la communication. Par contre, les moniteurs pourront intercepter (ou absorber) certains messages transigeant sur un canal donné et en laisser passer d'autres, le choix de laisser passer ou absorber les paquets se faisant en fonction de leur contenu, ces actions du moniteur ne concerneront donc pas uniquement les canaux. Il faut donc bien distinguer les messages et les canaux, même s'il se confondent dans la notation sans passage de valeur, qui est utilisée dans la sémantique.

3.2.3 Sémantique opérationnelle de l'opérateur de surveillance

Le tableau 3.5 décrit la sémantique opérationnelle de l'opérateur de surveillance. Ces règles méritent quelques explications sur leur rôle et leur sens propre dans l'évolution que nous avons voulu donner aux différents processus monitorés.

Les deux premières règles sont les plus simples. La première règle dit simplement qu'un moniteur n'a aucun contrôle sur les communications internes d'un processus, par exemple, même si le moniteur M interdit les communications sur le canal a, nous avons $[a.0 \parallel \overline{a}.0]^N_M \stackrel{c(a)}{\rightarrow} [0 \parallel 0]^N_M$. Par conséquent, les moniteurs décrits ici ne contrôlent

TAB. 3.5 – Sémantique opérationnelle de l'opérateur de restriction ([ ])

$(M1)$	$\dfrac{P \xrightarrow{c(a)} P'}{[P]_M^N \xrightarrow{c(a)} [P']_M^N}$	Communication
$(M2)$	$\dfrac{P \xrightarrow{\tau} P'}{[P]_M^N \xrightarrow{\tau} [P']_M^N}$	Action silencieuse
$(M3)$	$\dfrac{P \xrightarrow{a} P'}{[P]_M^N \xrightarrow{\tau} a.0 \| [P']_M^N}\ a \in \mathrm{O}(N),\ f_M(a) = V$	Sortie
$(M4)$	$\dfrac{P \xrightarrow{a} P'}{[P]_M^N \xrightarrow{\tau} [P']_M^N}\ a \in \mathrm{O}(N),\ f_M(a) = F$	Sortie bloquée
$(M5)$	$\dfrac{P \xrightarrow{a} P'}{P \parallel [Q]_M^N \xrightarrow{\tau} P' \parallel [a.0 \| Q]_M^N}\ a \in \mathrm{I}(N),\ f_M(a) = V$	Entrée
$(M6)$	$\dfrac{P \xrightarrow{a} P'}{P \parallel [Q]_M^N \xrightarrow{\tau} P' \parallel [Q]_M^N}\ a \in \mathrm{I}(N),\ f_M(a) = F$	Entrée bloquée

aucunement les communications internes d'un processus, ils ne contrôlent que les entrées et les sorties de celui-ci. Donc, les moniteurs contrôlent les communications qu'un processus peut effectuer avec d'autres processus. Concrètement, lorsqu'on parle de communications entre processus, on pense particulièrement, dans l'étude des réseaux, aux paquets qui peuvent voyager d'une machine à une autre ; il suit que les moniteurs définis ici contrôlent un processus tout comme un pare-feu contrôle une machine ou un groupe de machines.

De façon semblable, la règle $M2$ indique qu'un moniteur n'a aucun contrôle sur les actions silencieuses. Ce type d'action, qui n'est pas une action de base, est introduit par les règles $M3$ à $M6$. Ces actions ne correspondent pas à des éléments de communication, elles ne peuvent pas servir directement à établir une communication par le biais de l'opérateur de communication. Cependant, elles peuvent résulter de l'absorption d'une action par un moniteur ou bien de la migration d'une action à travers un moniteur, soit pour pénétrer à l'intérieur d'un domaine contrôlé, soit pour en sortir. Ces actions

ne sont pas contrôlées puisqu'elles ne font pas partie intégrante d'une communication, cependant, il est à noter qu'elles sont tout de même requises pour l'établissement d'une communication entre processus contrôlés. En effet, comme nous le verrons avec les autres règles, avant que la communication ne soit exécutée (avec l'opérateur de communication) plusieurs actions silencieuses peuvent être nécessaires pour faire migrer une action d'envoi de sorte qu'elle soit directement liée à l'action de réception qui lui correspond.

Les règles $M3$ et $M4$ décrivent quand une action d'envoi voulant sortir d'un domaine délimité par l'opérateur de surveillance est autorisée à le faire. Dans ces règles, jouent un rôle pour la première fois le moniteur qui contrôle une zone ainsi que l'ensemble de sortie associé à un nom. En fait, une action qui n'appartient ni à $\mathrm{I}(N)$ ni à $\mathrm{O}(N)$, soient les ensembles des canaux de sortie et des canaux d'entrée, est considérée comme une action interne. Seules les actions qui appartiennent à l'ensemble des canaux sortants sont autorisées à sortir de la zone. De plus, pour qu'une action soit autorisée à sortir d'une zone délimitée, elle doit être permise par le moniteur associé à cette zone. La règle $M3$ décrit ce qui se passe lorsqu'une action d'envoi est autorisée à sortir d'une zone contrôlée par un moniteur : le processus qui émet cette action évolue de façon normale et un processus formé seulement de cette action est créé en parallèle, à l'extérieur de la zone (ce sous-processus représente un message voyageant dans le processus global) ; le tout se traduit dans l'évolution du processus global par l'émission d'une action silencieuse. De son côté, la règle $M4$ décrit ce qui se passe lorsque le moniteur ne permet pas la sortie d'une action, cette action est absorbée : au niveau du processus qui se trouve à l'intérieur de la zone, l'évolution se fait comme si l'action était réellement exécutée, alors qu'au niveau du processus global, l'action d'envoi est remplacée par une action silencieuse, qui ne peut donner lieu à une communication future. On remarque également, à la règle $M4$, que pour qu'une action sortante puisse être absorbée par un moniteur, elle doit être associée à un canal sortant, il faut donc que ce soit une action qui, sans moniteur, aurait la capacité de sortir de la zone.

Finalement, les règles $M5$ et $M6$ se consacrent à définir le comportement d'un processus lorsqu'une action tente de pénétrer à l'intérieur d'un domaine contrôlé. Comme il n'est pas possible de procéder de façon similaire aux règles définissant la sortie d'action ($M4$ et $M5$), c'est-à-dire en faisant migrer les actions de réception, nous devons faire intervenir, en plus de la partie qui reçoit l'action, celle qui l'émet. Il est à noter que dans le cas des règles $M3$ et $M4$, le processus qui émet l'action est le même que le processus contrôlé, alors que dans le cas des règles $M5$ et $M6$ il s'agit de deux processus différents, d'où une différence de conception entre les règles. Ceci dit, la règle $M5$ nous indique qu'une action provenant d'un processus quelconque placé en communication avec un processus contrôlé, pourra pénétrer à l'intérieur du domaine contrôlé que si elle appartient à un canal entrant ($a \in \mathrm{I}(N)$) et qu'elle est autorisée par le moniteur ($f(a) = T$). Maintenant,

TAB. 3.6 – Syntaxe des composantes

$$C ::= [P]^N_M \;\mid\; C_1 \parallel C_2 \;\mid\; [C]^N_M$$

la règle $M6$ représente le cas où l'action est interdite par le moniteur : tout comme dans la règle $M4$, le processus qui envoi l'action évolue normalement, cependant l'action d'envoi ne traverse pas le moniteur, elle est absorbée par celui-ci, ce qui donne lieu à une action silencieuse. Encore une fois, pour qu'une action soit ainsi absorbée par le moniteur, elle doit se dérouler sur un canal entrant de la zone contrôlée.

3.2.4 Les composantes

Nous appellerons *composantes* les processus monitorés, soit les processus ayant la forme

$$[P]^N_M$$

et, par extension, nous appellerons également composantes, tout processus C ayant la syntaxe donnée au tableau 3.6.

Un résultat intéressant, que nous démontrerons lorsque nous aborderons l'équivalence entre processus, dit que si C_1 et C_2 sont deux composantes telles que $C_1 \approx C_2$, où $\approx$ dénote la relation de bissimulation faible, alors toute composante C possédant C_1 comme sous-composante est équivalente à la composante obtenue en remplaçant C_1 par C_2 dans C, et vice-versa. Autrement dit, la relation de bissimulation sera une congruence pour l'opérateur $\parallel$.

3.3 Quelques exemples et commentaires sur le fonctionnement des moniteurs

Par convention, le moniteur vide [ ] désignera le moniteur qui possède tous les canaux en entrée ainsi que tous les canaux en sortie et qui n'effectue aucun contrôle sur les actions. Donc, pour tout $a \in \mathcal{A}$

$$a.0 \parallel [0] \xrightarrow{\tau} 0 \parallel [a.0] \quad \text{(M5)}$$

et

$$[a.0] \xrightarrow{\tau} [0] \parallel a.0 \quad \text{(M3)}.$$

Lorsque le nom d'une composante ne sera pas important, nous écrirons parfois directement les canaux d'entrée et de sortie sur le moniteur. Les canaux d'entrée seront identifiés par une flèche vers le haut $\uparrow$, alors que les canaux de sortie seront identifiés par une flèche vers le bas. Par exemple, le processus

$$[P]^{\{a\uparrow,b\uparrow,b\downarrow,c\downarrow\}}$$

possède les canaux a et b comme canaux d'entrée et les canaux b et c comme canaux de sortie. Le canal b est donc à la fois un canal d'entrée et de sortie.

De même, lorsque le moniteur effectuant le contrôle d'une composante sera simple, nous l'identifierons directement par l'ensemble des actions qu'il contrôle. Ainsi, le moniteur $\{a, b\}$ permet toute action sauf a et b. Lorsqu'utilisé en parallèle avec le passage de valeur, ce qui est le cas par exemple avec le processus $[a(x).P]^{\{a\downarrow\}}_{\{a\}}$, il faut voir le moniteur $\{a\}$ comme bloquant l'ensemble des communications sur le canal a, alors que le moniteur $\{a(3)\}$ laisserait passer tous les messages sur le canal a, excepté les messages qui contiennent la valeur 3.

Voici maintenant quelques exemples d'évolutions de processus monitorés.

$$\begin{array}{lll}
[a.0]^{\{a\downarrow\}} \parallel [P \parallel [\overline{a}.0]^{\{a\uparrow\}}]^{\{a\uparrow\}} & \xrightarrow{\tau} & [0]^{\{a\downarrow\}} \parallel a.0 \parallel [P \parallel [\overline{a}.0]^{\{a\uparrow\}}]^{\{a\uparrow\}} \quad \text{(M3)} \\
& \xrightarrow{\tau} & [0]^{\{a\downarrow\}} \parallel [a.0 \parallel P \parallel [\overline{a}.0]^{\{a\uparrow\}}]^{\{a\uparrow\}} \quad \text{(M5)} \\
& \xrightarrow{\tau} & [0]^{\{a\downarrow\}} \parallel [P \parallel [a.0 \parallel \overline{a}.0]^{\{a\uparrow\}}]^{\{a\uparrow\}} \quad \text{(M5)} \\
& \xrightarrow{c(a)} & [0]^{\{a\downarrow\}} \parallel [P \parallel [0]^{\{a\uparrow\}}]^{\{a\uparrow\}} \quad \text{(M1)} \\
\\
[P \parallel [a.0]^{\{a\downarrow\}}]^{\{a\downarrow\}}_{\{a\}} & \xrightarrow{\tau} & [P \parallel a.0 \parallel [0]^{\{a\downarrow\}}]^{\{a\downarrow\}}_{\{a\}} \quad \text{(M3)} \\
& \xrightarrow{\tau} & [P \parallel [0]^{\{a\downarrow\}}]^{\{a\downarrow\}}_{\{a\}} \quad \text{(M4)}
\end{array}$$

Dans le premier exemple, l'action a a la permission de franchir deux moniteurs pour établir une communication avec l'action $\overline{a}$. Alors que dans le second exemple, l'action a traverse un premier moniteur puis est absorbée par le second. C'est un avantage que procure la sémantique de cette algèbre, il est possible de suivre étape par étape le déplacement d'une action traversant différents moniteurs ou pouvant être absorbée par l'un d'eux.

Le premier exemple est à l'image d'un paquet partant d'un réseau A et entrant dans un réseau B puis pénétrant dans un sous-réseau C. Ce paquet devra d'abord avoir

la permission de quitter le réseau A, puis devra avoir la permission d'entrer dans le réseau B et finalement, il devra avoir la permission d'entrer dans le sous-réseau C. La sémantique montre très bien où le paquet est rendu dans son parcours.

3.3.1 Lien avec l'opérateur de restriction de CCS

En CCS, l'opérateur de restriction permet de forcer la communication sur un canal. Par exemple, bien que le processus $a.0$ puisse exécuter l'action a et que le processus $\overline{a}.0$ puisse exécuter l'action $\overline{a}$, le processus CCS $(a.0 \mid \overline{a}.0) \setminus \{a\}$ ne peut exécuter aucune de ces actions. Par contre, il lui est permis d'exécuter une action silencieuse marquant une communication sur le canal a.

Avec notre algèbre, le même phénomène peut être obtenu par le biais de l'opérateur de surveillance. Par exemple, le processus $[a.0 \parallel \overline{a}.0]^{\emptyset}$ ne peut exécuter les actions a ou $\overline{a}$, ni faire sortir ou entrer une action a (par les règles $M3$ ou $M5$), cependant, il peut exécuter l'action $c(a)$, qui représente une communication sur le canal a. Notre algèbre donne, en ce sens, plus de flexibilité que CCS, en permettant, par exemple, des processus tels que $[a.0]^{\{a\uparrow\}}$, qui permettent la circulation des actions dans un seul sens. Ici, l'action a peut pénétrer le processus, mais ne peut pas en sortir.

3.4 Premier exemple de modélisation de réseaux

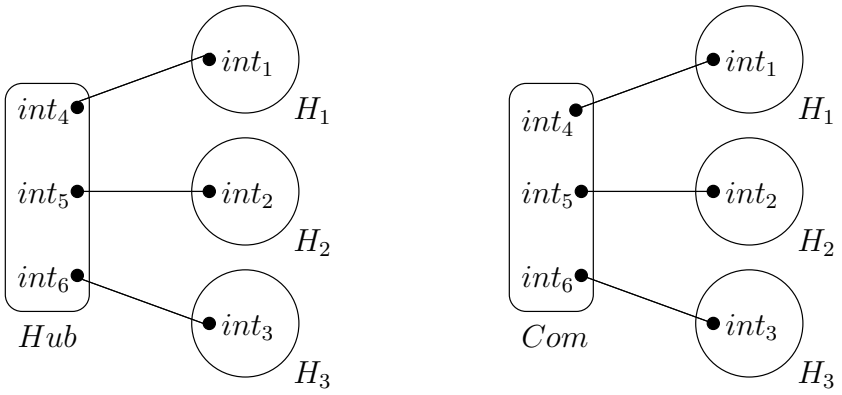
Dans ce premier exemple, nous allons modéliser deux réseaux formés de trois machines et d'une composante réseau. Dans le premier réseau cette composante sera un répéteur (mieux connu sous le nom anglais *hub*) et dans le second, un commutateur. Dans chacun des cas, cette composante aura pour fonction de relier les différents hôtes entre eux. Bien sûr, il en découlera deux réseaux aux fonctionnements différents, en effet, un répéteur qui reçoit un message sur l'une de ses interfaces le duplique et le retransmet sur chacune de ses autres interfaces, alors qu'un commutateur qui reçoit un message sur l'une de ses interfaces ne fait que le retransmettre sur l'interface communiquant avec l'hôte possédant l'adresse destination du message, s'il en existe un et si cette interface n'est pas la même que celle qui a reçu le message, dans les autres cas, le message sera simplement ignoré par le commutateur.

Ainsi, dans le réseau de gauche de la figure 3.1, si H_1 envoi un message à H_3, ce message sera également capté par H_2, puisqu'il lui sera communiqué par le répéteur qui

le transmettra aussi à H_3, qui est l'hôte destinataire du message. Cependant, dans le réseau de droite, si H_1 envoi un message à H_3, ce message sera inaccessible à H_2, puisque le commutateur prendra soin de ne communiquer le message qu'à l'hôte destinataire, c'est-à-dire H_3.

Dans la suite de cette section, nous procéderons dans un premier temps à la modélisation de ces deux petits réseaux. Puis, dans un second temps, nous regarderons comment munir le répéteur du réseau de gauche d'un moniteur de sorte à ce que ce dernier se comporte exactement comme le commutateur du second réseau. Il deviendra alors impossible, dans le réseau monitoré, pour un hôte d'intercepter un message qui ne lui est pas destiné, ce qu'il peut faire dans le réseau non monitoré.

FIG. 3.1 – Deux réseaux constitués de trois hôtes : R_H et R_C



3.4.1 Modélisation des réseaux

La figure 3.1 montre le schéma de deux réseaux constitués de trois hôtes qui communiquent par le biais d'une composantes réseaux : un répéteur pour le réseau de gauche (R_H) et un commutateur pour celui de droite (R_C). Voici une modélisation de ces deux réseaux :

$$H_i \stackrel{\text{def}}{=} int_i(@_s, @_d).H_i + \overline{int}_{i+3}(@_s, @_d).H_i, \quad 1 \leq i \leq 3$$

$$H \stackrel{\text{def}}{=} (H_1 \mid H_2 \mid H_3)$$

$$\begin{aligned} Hub \stackrel{\text{def}}{=}\ & \overline{int_1}(@_s, @_d).int_5(@_s, @_d).int_6(@_s, @_d).Hub \\ +\ & \overline{int_2}(@_s, @_d).int_4(@_s, @_d).int_6(@_s, @_d).Hub \\ +\ & \overline{int_3}(@_s, @_d).int_4(@_s, @_d).int_5(@_s, @_d).Hub \end{aligned}$$

$$
\begin{aligned}
Com &\stackrel{\text{def}}{=} \sum_{i=1}^{3} \overline{int}_i(@_s, @_d \in \text{Rés} \setminus \{@_i\}).int_{p(@_d)}(@_s, @_d).Com \\
&\quad + \sum_{i=1}^{3} \overline{int}_i(@_s, @_d \notin \text{Rés} \setminus \{@_i\}).Com \\
R_H &\stackrel{\text{def}}{=} Hub \parallel H \\
R_C &\stackrel{\text{def}}{=} Com \parallel H
\end{aligned}
$$

Dans le réseau R_H comme dans le réseau R_C, on considère un hôte comme étant simplement un processus capable de recevoir un paquet par le biais d'une interface communicante (par exemple int_4 dans le cas de H_1) et capable d'envoyer un paquet par sa propre interface (int_1 pour H_1).

Le répéteur est un processus pouvant recevoir un paquet provenant de l'interface d'un des trois hôtes (H_1, H_2 ou H_3) et capable de retourner ce paquet vers les deux autres hôtes par le biais de ses interfaces reliées avec ceux-ci (par exemple, si le message est envoyé par H_1, c'est-à-dire par le biais de l'interface int_1, il sera retourné par les interfaces int_5 et int_6 vers H_2 et H_3.)

Le commutateur pour sa part a un fonctionnement un peu plus délicat que le répéteur. Il reçoit un paquet provenant d'un des trois hôtes, puis, si le paquet possède l'adresse d'un des deux autres hôtes du réseau, il lui sera expédié par le biais de l'interface communiquant avec celui-ci (par exemple, si H_1 envoie par int_1 un paquet dont l'adresse destination est égale à celle de H_3, ce paquet sera communiqué à ce dernier par le biais de l'interface int_6). Dans le cas où le paquet n'est pas destiné à l'un des deux autres hôtes, il est tout simplement ignoré.

La fonction $p(@)$ utilisée dans la modélisation du commutateur prend en entrée une adresse IP et renvoie en sortie l'interface du commutateur communiquant avec l'hôte possédant cette adresse. On a donc : $p(@_1) = int_4$, $p(@_2) = int_5$ et $p(@_3) = int_6$.

3.4.2 Pare-feu pour le répéteur

Nous souhaitons que le réseau R_H se comporte comme le réseau R_C, c'est-à-dire que nous voulons que le réseau R_H respecte la politique disant qu'un hôte ne doit pas recevoir un paquet qui ne lui est pas destiné. Clairement, R_H ne respecte pas cette

politique de sécurité, en effet, on peut le voir par la trace suivante :

$$\begin{array}{lll} R_H & \xrightarrow{c(int_1(@_1,@_3))} & int_5(@_1,@_3).int_6(@_1,@_3).Hub \parallel H \quad \text{(Com.)} \\ & \xrightarrow{c(int_5(@_1,@_3))} & int_6(@_1,@_3).Hub \parallel H \quad \text{(Com.)} \\ & \xrightarrow{c(int_6(@_1,@_3))} & R_H \quad \text{(Com.)} \end{array}$$

Comme nous pouvons le constater, avant que le répéteur ne livre à H_3 un message provenant de H_1 et lui étant destiné, il livre d'abord ce message à H_2, contrairement au commutateur qui envoie directement un tel paquet à H_3 et qui ne l'enverra jamais à H_2 :

$$\begin{array}{lll} R_C & \xrightarrow{c(int_1(@_1,@_3))} & int_6(@_1,@_3).Com \parallel H \quad \text{(Com.)} \\ & \xrightarrow{c(int_6(@_1,@_3))} & R_C \quad \text{(Com.)} \end{array}$$

On doit remplacer le processus Hub dans le réseau R_H par une composante qui aura la forme

$$[Hub]_M^N.$$

Le nom N doit posséder les canaux int_1, int_2 et int_3 en entrée et les canaux int_4, int_5 et int_6 en sortie. On doit donc avoir $\mathrm{I}(N) = \{int_1, int_2, int_3\}$ et $\mathrm{O}(N) = \{int_4, int_5, int_6\}$. Il reste à déterminer le moniteur M adéquat.

Ce moniteur jouera le rôle d'un pare-feu qui bloque les messages partant du répéteur et allant vers un hôte qui n'est pas le destinataire du message. Le pare-feu M effectuant ce travail est donné par la fonction de pare-feu suivante :

$$f_M(x) = \begin{cases} false & \text{si} \quad x = int_4(@, @_d \neq @_1) \\ & \qquad\; x = int_5(@, @_d \neq @_2) \\ & \qquad\; x = int_6(@, @_d \neq @_3) \\ true & \text{sinon} \end{cases}$$

On note R'_H le réseau alors obtenu, soit le réseau

$$R'_H \stackrel{\text{def}}{=} [Hub]_M^N \parallel H$$

Ainsi, on obtient lorsque H_1 envoie un paquet à H_3 :

$$\begin{array}{lll} R'_H & \xrightarrow{\tau} & [Hub \parallel int_1(@_1,@_3).0]_M^N \parallel H \quad \text{(M5)} \\ & \xrightarrow{c(int_1(@_1,@_3))} & [int_5(@_1,@_3).int_6(@_1,@_3).Hub]_M^N \parallel H \quad \text{(M1, Com.)} \\ & \xrightarrow{\tau} & [int_6(@_1,@_3).Hub]_M^N \parallel H \quad \text{(M4)} \\ & \xrightarrow{\tau} & [Hub]_N^M \parallel int_6(@_1,@_3).0 \parallel H \quad \text{(M3)} \\ & \xrightarrow{c(int_6(@_1,@_3))} & R'_H \quad \text{(Com.)} \end{array}$$

Dans cette trace d'actions, la troisième action exécutée, soit la deuxième action silencieuse, correspond à l'absorption par le moniteur M du paquet partant de Hub et destiné à H_2. Alors que pour l'action suivante, qui correspond à l'envoi d'un paquet destiné à H_3, le paquet est autorisé à franchir le moniteur de sorte à communiquer directement avec H_3.

Nous verrons plus loin que le processus $[Hub]_M^N$ est équivalent au processus $[Com]^N$. Cette équivalence correspondra à la bissimulation faible que nous définissons à la section 3.6. À noter qu'il est nécessaire d'incorporer le processus Com dans une composante pour avoir cette équivalence, cette composante possédera les mêmes canaux d'entrées et de sorties que dans le cas du processus Hub, mais elle ne possédera aucun moniteur.

3.5 Les actions captées

3.5.1 Premier pas vers l'établissement d'une relation d'équivalence

Nous voulons définir une relation d'équivalence entre les processus de notre algèbre. En CCS, que l'on parle de l'équivalence de trace, de bissimulation forte ou faible, d'équivalence de blocage, ou autres relations d'équivalences entre processus, ce qui est mis en évidence demeure toujours la capacité, ou encore l'incapacité, des processus à pouvoir exécuter une action donnée à un moment donné.

Nous aimerions faire ainsi avec notre algèbre. Cependant, on s'aperçoit assez rapidement qu'il nous manque un élément important pour pouvoir bien distinguer les processus. Voici un petit exemple pour illustrer ces propos. Considérons les trois processus suivants :

$$[0], \quad [0]^{\emptyset} \quad \text{et} \quad [0]^{\{a\uparrow\}}_{\{a\}}$$

la question est de savoir si l'on considère ces processus comme étant équivalents ou non. Aucun de ces trois processus n'est apte à évoluer, on peut donc croire qu'ils peuvent être assimilés au processus 0. Cependant, si l'on place ces processus en parallèle avec un processus qui peut émettre l'action a, par exemple le processus $a.0$, on constate qu'ils donnent lieu à des processus aux comportements différents :

$$\begin{array}{lcll} a.0 \parallel [0] & \xrightarrow{\tau} & [a.0] & \text{(M5, Équiv.)} \\ a.0 \parallel [0]^{\emptyset} & \overset{\tau}{\not\rightarrow} & & \\ a.0 \parallel [0]^{\{a\uparrow\}}_{\{a\}} & \xrightarrow{\tau} & [0]^{\{a\uparrow\}}_{\{a\}} & \text{(M6, Équiv.)} \end{array}$$

ce qui nous mène donc à introduire un nouveau type d'action : les actions captées. La définition et l'analyse de ces nouvelles actions, qui est l'objet du reste de cette section, nous permettra de définir une relation d'équivalence convenable (ce qui se fera à la section 3.6).

3.5.2 Les actions captées

Nous venons de voir que les processus $[0]^{\emptyset}$, $[0]$ et $[0]^{\{a\uparrow\}}_{\{a\}}$ sont des processus distincts même s'ils possèdent le même ensemble de traces d'exécution. Cette distinction provient du fait qu'ils interagissent différemment avec d'autres processus. Plus précisément, ils diffèrent par leur capacité à capter différentes actions pouvant provenir d'un processus externe.

Par exemple, le processus $[0]^{\emptyset}$ ne peut capter aucune action, la seule évolution possible du processus $a.0 \parallel [0]^{\emptyset}$ se fait par l'exécution de l'action d'envoi a, alors que les processus $a.0 \parallel [0]$ et $a.0 \parallel [0]^{\{a\uparrow\}}_{\{a\}}$ peuvent évoluer par une action τ. Le processus $a.0 \parallel [0]$ peut évoluer par la règle d'entrée pour devenir le processus $[a.0]$, alors que le processus $a.0 \parallel [0]^{\{a\uparrow\}}_{\{a\}}$ peut évoluer par la règle d'entrée bloquée pour devenir le processus $[0]^{\{a\uparrow\}}_{\{a\}}$.

On dit alors que ces processus peuvent capter l'action a ; en somme, ils peuvent acquérir une action d'envoi a venant d'un processus externe sans l'exécuter et sans la faire participer dans une communication.

Un processus pouvant capter une action peut soit la laisser passer, soit absorber cette action ; celle-ci disparaît alors du modèle sans avoir été impliquée dans une communication entre deux processus. Nous donnons ici, de façon plus formelle, les définitions des terminologies employées ici, soit les définitions des actions acceptées, absorbées et captées.

Définition 3.5.1. Un processus de la forme $[P]^N_M$ peut accepter (ou laisser passer) l'action a si

$$a.0 \parallel [P]^N_M \xrightarrow{\tau} [a.0 \parallel P]^N_M \qquad \text{(M5)}.$$

Donc, puisque seule la règle d'entrée permet cette évolution, le processus accepte a si $a \in \mathrm{I}(N)$ et $f_M(a) = T$. Un processus de la forme $P \parallel Q$ ou $P + Q$ peut accepter a si l'un des processus P ou Q peut l'accepter.

Définition 3.5.2. Un processus de la forme $[P]^N_M$ peut absorber l'action a si

$$a.0 \parallel [P]^N_M \xrightarrow{\tau} [P]^N_M \qquad \text{(M6)}.$$

Alors qu'un processus de la forme $P \parallel Q$ ou $P + Q$ peut absorber l'action a si l'un des processus P ou Q peut l'absorber.

Définition 3.5.3. Un processus peut capter l'action a s'il peut l'accepter ou s'il peut l'absorber. On note

$$P \xrightarrow{\widetilde{a}} Q$$

le fait qu'un processus P puisse capter une action a pour devenir le processus Q. À noter que cette définition introduit un nouveau type d'action, les actions captées, qui ne sont pas des actions élémentaires. À noter également que l'opérateur $\rightarrow$ de cette définition n'est pas le même que celui défini par la sémantique de la section 3.2, il s'agit donc ici d'un abus de notation. Nous recourrons à la même flèche, au lieu de noter le captage d'action par un symbole différent, dans le but d'uniformiser la notation dans les prochaines sections. C'est donc la présence du tilde qui indiquera s'il s'agit d'une action captée ou d'une action exécutée.

Quelques exemples

Comparons, à l'aide de l'actions spéciale $\widetilde{a}$, les processus $[0]^{\emptyset}$, $[0]$ et $[0]^{\{a\uparrow\}}_{\{a\}}$.

$$[0]^{\emptyset} \quad \overset{\widetilde{a}}{\not\rightarrow}$$

$$[0] \quad \xrightarrow{\widetilde{a}} \quad [a.0]$$

$$[0]^{\{a\uparrow\}}_{\{a\}} \quad \xrightarrow{\widetilde{a}} \quad [0]^{\{a\uparrow\}}_{\{a\}}$$

Il devient maintenant évident que ces trois processus peuvent facilement être différenciés. En effet, on voit d'abord que le premier processus n'est pas apte à capter l'action a alors que les deux autres le sont, ce qui distingue ce processus. Maintenant, les deux derniers processus peuvent également être distingués, puisque le processus $[0]$, après avoir capté l'action a, est capable de l'émettre :

$$[0] \xrightarrow{\widetilde{a}} [a.0] \xrightarrow{\tau} [0] \parallel a.0 \xrightarrow{a} [0]$$

alors que le processus $[0]^{\{a\uparrow\}}_{\{a\}}$, bien que pouvant également capter l'action a, sera incapable de l'émettre par la suite :

$$[0]^{\{a\uparrow\}}_{\{a\}} \xrightarrow{\widetilde{a}} [0]^{\{a\uparrow\}}_{\{a\}}.$$

On remarque qu'en fait le processus $[0]$ laisse passer l'action a, alors que le processus $[0]^{\{a\uparrow\}}_{\{a\}}$ absorbe cette action. Mais cette distinction entre absorber et laisser passer ne joue pas un rôle majeur dans la définition d'une relation de bissimulation entre processus. L'élément fondamental n'est pas que l'action soit en fait absorbée ou acceptée, celui-ci concerne plutôt les capacités d'évolution d'un processus après qu'il aura capté une action. Donc, l'élément principal distinguant les processus $[0]$ et $[0]^{\{a\uparrow\}}_{\{a\}}$ n'est pas que le premier accepte a et non le second, mais bien que $[0]$ peut utiliser l'action a après l'avoir captée, ce que ne peut faire le processus $[0]^{\{a\uparrow\}}_{\{a\}}$.

Maintenant ces concepts bien compris, nous sommes prêts à définir une relation de bissimulation pour comparer les processus de notre algèbre.

3.6 Bissimulation

3.6.1 L'opérateur $\Rightarrow$

Nous définissons l'opérateur $\Rightarrow$ de manière similaire à sa définition en CCS. Cette définition se base sur la capacité des processus à exécuter des actions, ainsi que sur leur capacité à capter des actions, comme nous venons de le définir dans la section précédente. Ainsi, dans le tableau 3.7 l'action γ peut aussi bien représenter une action d'envoi a, une action de réception $\overline{a}$, une communication $c(a)$, une action silencieuse τ, ou une action captée $\widetilde{a}$. Le symbole ϵ dénote pour sa part une absence d'action, une trace vide.

TAB. 3.7 – Sémantique opérationnelle de l'opérateur $\Rightarrow$

$$P \overset{\epsilon}{\Rightarrow} P \qquad \frac{P \overset{\tau}{\rightarrow} P' \quad P' \overset{\epsilon}{\Rightarrow} Q}{P \overset{\epsilon}{\Rightarrow} Q}$$

$$\frac{P \overset{\epsilon}{\Rightarrow} P' \quad P' \overset{\gamma}{\rightarrow} Q' \quad Q' \overset{\epsilon}{\Rightarrow} Q}{P \overset{\gamma}{\Rightarrow} Q}$$

3.6.2 La bissimulation faible

Pour dénifir la bissimulation faible, nous utiliserons des actions chapeautées $\hat{\gamma}$ avec l'opérateur $\Rightarrow$. En fait, comme défini par Milner pour CCS, cet opérateur chapeau agit sur des traces pour en enlever toutes les actions silencieuses. Par exemple, si $t = a.\tau.\overline{a}.b.c(a).\tau.\tau$, alors $\hat{t} = a.\overline{a}.b.c(a)$. Pour une trace contenant uniquement des actions silencieuses, nous obtiendrons alors une trace vide, par exemple si $t = \tau.\tau.\tau$, alors $\hat{t} = \epsilon$. Pour des traces qui ne contiennent qu'une seule action, comme ce sera le cas lorsque nous utiliserons cet opérateur pour nos démonstrations, la trace demeure inchangée si elle est différente d'une action silencieuse, et elle est remplacée par ϵ sinon. Donc, pour une action différente d'une action silencieuse, nous avons $\gamma = \hat{\gamma}$, dans les autres cas, nous avons $\hat{\tau} = \epsilon$.

Définition 3.6.1. Une relation binaire $\mathcal{S} \subset \mathcal{P} \times \mathcal{P}$ entre deux processus P et Q est une bissimulation faible si, lorsque $(P, Q) \in \mathcal{S}$, cela implique que pour tout γ l'on ait :

1. Si $P \xrightarrow{\gamma} P'$, alors pour un certain $Q' \in \mathcal{P}$, $Q \stackrel{\hat{\gamma}}{\Rightarrow} Q'$ et $(P', Q') \in \mathcal{S}$
2. Si $Q \xrightarrow{\gamma} Q'$, alors pour un certain $P' \in \mathcal{P}$, $P \stackrel{\hat{\gamma}}{\Rightarrow} P'$ et $(P', Q') \in \mathcal{S}$.

Définition 3.6.2. Deux processus P et Q sont dit faiblement bissimilaires, ce qui sera noté $P \approx Q$, si $(P, Q) \in \mathcal{S}$ pour une certaine bissimulation faible $\mathcal{S}$. Par conséquent, la relation $\approx$ peut s'exprimer comme suit :

$$\approx = \bigcup \{\mathcal{S} \ : \ \mathcal{S} \text{ est une bissimulation faible } \}$$

Il est également possible de définir une relation de bissimulation forte, en fait, il suffit pour cela de remplacer les symboles $\stackrel{\hat{\gamma}}{\Rightarrow}$ de la définition 3.6.1 par des symboles $\xrightarrow{\gamma}$. Cependant, la bissimulation forte est une relation trop restrictive pour nos besoins. Par exemple, à la section 3.6.4, nous montrerons que les deux processus $[Com]^N$ et $[Hub]^N_M$ définis à la section 3.4 sont faiblement bissimilaires, c'est-à-dire qu'ils sont équivalents au sens de la bissimulation faible. Il est par contre évident que ces deux processus ne sont pas fortement bissimilaires. Puisque nous nous intéressons plus particulièrement à l'étude des réseaux, soit à l'étude des paquets pouvant voyager entre des réseaux ou entre des hôtes appartenant à un réseau, il semble logique de considérer ces deux composantes comme équivalentes, puisqu'un modèle de réseaux construit avec l'une ou l'autre de ces composantes ne changera pas de comportement face aux communications réalisables, et donc face aux paquets pouvant circuler.

3.6.3 Bissimulation et congruence

Voici maintenant quelques propositions établissant la congruence de la relation de bissimulation faible avec les opérateurs de préfixage, de communication et de surveillance.

Proposition 3.6.1. *La bissimulation est une congruence pour l'opérateur de préfixage.*

Démonstration.

- P et Q sont deux processus, $\alpha \in \mathcal{A} \cup \overline{\mathcal{A}}$
  $\langle$ Par hypothèse $\rangle$
- $P \approx Q$
  $\Rightarrow$ $\langle$ Par définition de $\approx$ $\rangle$
- $\exists \mathcal{S}$ une bissimulation telle que $(P, Q) \in \mathcal{S}$
  La seule règle qui s'applique pour faire évoluer le processus $\alpha.P$ est la règle de préfixage, et la seule évolution possible de ce processus est $\alpha.P \xrightarrow{\alpha} P$. Il en est de même pour le processus $\alpha.Q$ dont la seule évolution est $\alpha.Q \xrightarrow{\alpha} Q$. On obtient par conséquent que
- $(\alpha.P, \alpha.Q) \cup \mathcal{S}$ est une bissimulation
  $\Rightarrow$ $\langle$ Par définition de $\approx$ $\rangle$
- $\alpha.P \approx \alpha.Q$

□

Proposition 3.6.2. *La bissimulation est une congruence pour l'opérateur* $\|$.

Démonstration. Il suffit de montrer que

$$\mathcal{S} = \{(P\|R,\ Q\|R) : P, Q, R \in \mathcal{P} \text{ et } P \approx Q\}$$

est une bissimulation.

Selon la sémantique de l'algèbre, l'évolution du processus $P \parallel R$ doit faire intervenir soit la règle d'entrelacement, soit la règle de communication, soit l'une des règles $M5$ ou $M6$. Toute évolution du processus $P \parallel R$ est de la forme $P \parallel R \xrightarrow{\gamma} P' \parallel R'$, voyons les différentes formes en lesquelles peut consister une telle évolution, selon la règle utilisée pour faire évoluer le processus.

Cas 1 Entrelacement avec $P \xrightarrow{\gamma} P'$ et $R \equiv R'$.
Puisque $P \approx Q$, on aura $Q \stackrel{\hat{\gamma}}{\Rightarrow} Q'$ pour un certain $Q' \in \mathcal{P}$ avec $P' \approx Q'$. Alors, toujours selon la règle d'entrelacement, $Q \parallel R \stackrel{\hat{\gamma}}{\Rightarrow} Q' \parallel R$ et $(P' \parallel R,\ Q' \parallel R) \in \mathcal{S}$.

Cas 2 Entrelacement avec $R \xrightarrow{\gamma} R'$ et $P' \equiv P$.
Alors $Q \parallel R \xrightarrow{\gamma} Q \parallel R'$ et $(P \parallel R',\ Q \parallel R') \in \mathcal{S}$.

Cas 3 Communication avec $P \parallel R \xrightarrow{c(a)} P' \parallel R'$, $P \xrightarrow{\alpha} P'$ et $R \xrightarrow{\overline{\alpha}} R'$.
Puisque $P \approx Q$, on aura $Q \stackrel{\alpha}{\Rightarrow} Q'$ pour un certain $Q' \in \mathcal{P}$ avec $P' \approx Q'$. Donc, par application des règles d'entrelacement et de communication, on obtient que $Q \parallel R \stackrel{c(a)}{\Rightarrow} Q' \parallel R'$ et alors $(P' \parallel R',\ Q' \parallel R') \in \mathcal{S}$.

Cas 4 $M5$ ou $M6$ avec $P \parallel R \xrightarrow{\tau} P' \parallel R'$, $P \xrightarrow{\widetilde{a}} P'$ et $R \xrightarrow{a} R'$.
Ce cas est similaire au cas précédent, en effet, puisque $P \approx Q$, on aura $Q \stackrel{\widetilde{a}}{\Rightarrow} Q'$ pour un certain $Q' \in \mathcal{P}$ avec $P' \approx Q'$. Donc $Q \parallel R \stackrel{\tau}{\Rightarrow} Q' \parallel R'$, ce qui nous donne $(P' \parallel R', Q' \parallel R') \in \mathcal{S}$.
Il en est de même dans le cas où $P \parallel R \xrightarrow{\tau} P' \parallel R'$, $P \xrightarrow{a} P'$ et $R \xrightarrow{\widetilde{a}} R'$.

Nous avons donc démontré que l'ensemble $\mathcal{S} = \{(P \| R,\ Q \| R) : P, Q, R \in \mathcal{P} \text{ et } P \approx Q\}$ est bien une bissimulation. Il suit que si P et Q sont deux processus tels que $P \approx Q$, alors pour tout processus R nous avons que $P \parallel R \approx Q \parallel R$, ce qui signifie que la bissimulation est une congruence pour l'opérateur $\parallel$. □

Proposition 3.6.3. *La bissimulation est une congruence pour l'opérateur* $[\ ]_M^N$.

Démonstration. Notons $\mathcal{P}_A$ l'ensemble des processus de la forme $a_1.0 \| a_2.0 \| a_3.0 \| \ldots$ avec $A = \{a_1, a_2, a_3, \ldots\} \subseteq \mathcal{A}$. À noter que l'ensemble A peut être vide, donc $0 \in \mathcal{P}_{\mathcal{A}}$. Nous allons montrer que

$$\mathcal{S} = \{([P]_M^N \parallel R,\ [Q]_M^N \parallel R : P \approx Q \text{ et } R \in \mathcal{P}_{\mathcal{A}}\}$$

est une bissimulation, comme elle contient $([P]_M^N,\ [Q]_M^N)$ ceci démontrera la proposition.

Pour procéder à la démonstration, on sépare les cas selon la règle de l'opérateur de surveillance impliquée dans l'évolution du processus $[P]_M^N$. Nous traitons d'abord les cas impliquant seulement le processus $[P]_M^N$, c'est-à-dire, les évolutions permises par la règle d'entrelacement dans lesquelles le processus R n'évolue pas.

Cas 1 Règles $M1$ ou $M2$ avec $[P]_M^N \xrightarrow{\gamma} [P']_M^N$, $\gamma \in C(\mathcal{A}) \cup \{\tau\}$ et $P \xrightarrow{\gamma} P'$.
Puisque $P \approx Q$, on aura $Q \stackrel{\hat{\gamma}}{\Rightarrow} Q'$ pour un certain $Q' \in \mathcal{P}$ avec $Q' \approx P'$. Comme l'évolution $\stackrel{\hat{\gamma}}{\Rightarrow}$ représente une suite d'exécutions d'action silencieuse et possiblement d'une action de communication, les règles $M1$ et $M2$ s'appliquent pour déduire que $[Q]_M^N \stackrel{\hat{\gamma}}{\Rightarrow} [Q']_M^N$, ce qui implique que $([P']_M^N,\ [Q']_M^N) \in \mathcal{S}$.

Cas 2 Règle $M3$ avec $[P]_M^N \xrightarrow{\tau} a.0 \parallel [P']_M^N$.
Pour que l'application de la règle $M3$ soit permise, il faut que $a \in \mathrm{O}(N)$, $f_M(a) = T$ et $P \xrightarrow{a} P'$. Puisque $P \approx Q$, on aura $Q \overset{a}{\Rightarrow} Q'$ pour un certain $Q' \in \mathcal{P}$ avec $Q' \approx P'$. Comme l'évolution $\overset{a}{\Rightarrow}$ contient une suite d'exécutions contenant l'exécution de l'action a suivie et précédée par des exécutions d'action silencieuse, les règles $M2$ et $M3$ s'appliquent pour obtenir $[Q]_M^N \overset{\tau}{\Rightarrow} a.0 \parallel [Q']_M^N$, et donc $([P']_M^N,\ [Q']_M^N) \in \mathcal{S}$.

Cas 3 Règle $M4$ avec $[P]_M^N \xrightarrow{\tau} [P']_M^N$ et $P \xrightarrow{a} P'$.
L'application de la règle $M4$ signifie qu'une tentative du processus P d'émettre une action d'envoi a été bloquée par le moniteur M, pour ce faire, il faut que $a \in \mathrm{O}(N)$ et $f_M(a) = F$. Puisque $P \approx Q$, on doit avoir $Q \overset{a}{\Rightarrow} Q'$ pour un certain $Q' \in \mathcal{P}$ avec $Q' \approx P'$. Les règles $M2$ et $M4$ s'appliquent pour obtenir $[Q]_M^N \overset{\tau}{\Rightarrow} [Q']_M^N$. Cela signifie que $([P']_M^N,\ [Q']_M^N) \in \mathcal{S}$.

Cas 4 Règle $M5$ avec $[P]_M^N \xrightarrow{\widetilde{a}} [a.0 \parallel P]_M^N$.
L'application de la règle $M5$ implique que l'action a possède la permission de pénétrer le moniteur M, il faut donc que $a \in \mathrm{I}(N)$ et $f_M(a) = T$. La règle $M5$ s'applique donc pour donner $[Q]_M^N \xrightarrow{\widetilde{a}} [a.0 \parallel Q]_M^N$, et puisque $a.0 \parallel P \approx a.0 \parallel Q$ (par la proposition 3.6.1), on a donc $([a.0 \parallel P]_M^N,\ [a.0 \parallel Q]_M^N) \in \mathcal{S}$.

Cas 5 Règles $M6$ avec $[P]_M^N \xrightarrow{\widetilde{a}} [P]_M^N$.
Dans ce cas, il s'agit de l'absorption de l'action a, il faut donc que $a \in \mathrm{I}(N)$ et $f_M(a) = F$; ce qui signifie que le processus $[Q]_M^N$ peut également absorber l'action a, ce qui donne $[Q]_M^N \xrightarrow{\widetilde{a}} [Q]_M^N$, et le couple $([P]_M^N, [Q]_M^N)$ appartient déjà à $\mathcal{S}$.

Chacune des évolutions possibles affectant seulement le processus $[P]_M^N$ est traitée parmi les cinq cas. Les cas affectant seulement le processus R se traitent de façon similaire au second cas de la proposition précédente, alors que les cas affectant les deux parties, c'est-à-dire de la forme $[P]_M^N \parallel R \xrightarrow{\tau} [P]_M^N \parallel R'$ ou $[P]_M^N \parallel R \xrightarrow{\tau} [a.0 \parallel P]_M^N \parallel R'$ ne cause pas plus de problème puisqu'ils se traitent comme les cas 4 et 5 ci-haut. On déduit alors que $\mathcal{S}$ est une bissimulation qui contient $([P]_M^N,\ [Q]_M^N)$, ce qui complète la démonstration. □

3.6.4 Retour sur l'exemple de la section 3.4

Nous avons mentionné à la section 3.4 que les processus $[Hub]_M^N$ et $[Com]^N$ étaient équivalents au sens de la bissimulation définie dans cette section. Il nous est maintenant

possible de démontrer cette affirmation.

Il suffit de construire une relation de bissimulation faible, laquelle devra contenir le couple $([Hub]^N_M, [Com]^N)$, pour que le résultat suive immédiatement. Pour construire cette bissimulation, il suffit de traiter les différentes évolutions du processus $[Hub]^N_M$ et d'associer ces évolutions à celles équivalentes du processus $[Com]^N$. Si l'on considère uniquement les évolutions engendrées par un envoi provenant de H_1 (autrement dit, si l'on considère que $\mathrm{I}(N) = \{int_1\}$ au lieu de $\{int_1, int_2, int_3\}$), nous trouvons la relation de bissimulation suivante :

$$
\begin{aligned}
S = \; & \{([Hub \parallel R_1]^N_M \parallel R_2,\ [Com \parallel R_1]^N \parallel R_2) : R_1 \in \mathcal{P}_{int_1}, R_2 \in \mathcal{P}_{int_5 \cup int_6}\} \\
\cup \; & \{([int_5(@_1, @).int_6(@_1, @).Hub \parallel R_1]^N_M \parallel R_2,\ [Com \parallel R_1]^N \parallel R_2) : \\
& R_1 \in \mathcal{P}_{int_1}, R_2 \in \mathcal{P}_{int_5 \cup int_6}, @ \neq @_2, @_3\} \\
\cup \; & \{([int_6(@_1, @).Hub \parallel R_1]^N_M \parallel R_2,\ [Com \parallel R_1]^N \parallel R_2) : \\
& R_1 \in \mathcal{P}_{int_1}, R_2 \in \mathcal{P}_{int_5 \cup int_6}, @ \neq @_2, @_3\} \\
\cup \; & \{([int_5(@_1, @_2).int_6(@_1, @_2).Hub \parallel R_1]^N_M \parallel R_2,\ [int_5(@_1, @_2).Com \parallel R_1]^N \parallel R_2) : \\
& R_1 \in \mathcal{P}_{int_1}, R_2 \in \mathcal{P}_{int_5 \cup int_6}\} \\
\cup \; & \{([int_6(@_1, @_2).Hub \parallel R_1]^N_M \parallel R_2,\ [Com \parallel R_1]^N \parallel R_2) : \\
& R_1 \in \mathcal{P}_{int_1}, R_2 \in \mathcal{P}_{int_5 \cup int_6}\} \\
\cup \; & \{([int_5(@_1, @_3).int_6(@_1, @_3).Hub \parallel R_1]^N_M \parallel R_2,\ [int_6(@_1, @_3).Com \parallel R_1]^N \parallel R_2) : \\
& R_1 \in \mathcal{P}_{int_1}, R_2 \in \mathcal{P}_{int_5 \cup int_6}\} \\
\cup \; & \{([int_6(@_1, @_3).Hub \parallel R_1]^N_M \parallel R_2,\ [int_6(@_1, @_3.Com \parallel R_1]^N \parallel R_2) : \\
& R_1 \in \mathcal{P}_{int_1}, R_2 \in \mathcal{P}_{int_5 \cup int_6}\}
\end{aligned}
$$

L'ensemble $\mathcal{P}_{int_1}$ contient tous les processus de la forme $int_1(\mathbf{a_1}, \mathbf{b_1}).0 \parallel int_1(\mathbf{a_2}, \mathbf{b_2}).0 \parallel \ldots \parallel int_1(\mathbf{a_n}, \mathbf{b_n}).0$ où les $\mathbf{a_i}$ ainsi que les $\mathbf{b_i}$ correspondent à des adresses IP. Alors que l'ensemble $\mathcal{P}_{int_5 \cup int_6}$, de façon similaire, contient tous les processus de la forme $int_5(\mathbf{a_1}, \mathbf{b_1}).0 \parallel int_5(\mathbf{a_2}, \mathbf{b_2}).0 \parallel \ldots \parallel int_5(\mathbf{a_n}, \mathbf{b_n}).0 \parallel int_6(\mathbf{c_1}, \mathbf{d_1}).0 \parallel int_6(\mathbf{c_2}, \mathbf{d_2}).0 \parallel \ldots \parallel int_6(\mathbf{c_m}, \mathbf{d_m}).0$. De manière simplifiée, un processus appartient à $\mathcal{P}_{int_1}$ s'il est constitué d'actions pouvant entrer dans la composante $[X]^{\{int_1\uparrow, int_4\downarrow, int_5\downarrow, int_6\downarrow\}}$, donc d'actions passant par le canal int_1, alors que l'ensemble $\mathcal{P}_{int_5 \cup int_6}$ est constitué d'actions pouvant sortir de cette composante, donc passant par l'un des canaux int_4, int_5 ou int_6.

Dans ce cas simplifié, il n'est pas difficile de se convaincre que S est bien une bissimulation. Dans le cas général, c'est-à-dire avec $\mathrm{I}(N) = \{int_1, int_2, int_3\}$, il s'agit d'ajouter

les évolutions concernant l'envoi de paquets provenant de H_2 ou H_3, les processus R_1 et R_2 devront alors appartenir respectivement à $\mathcal{P}_{int_1 \cup int_2 \cup int_3}$ et $\mathcal{P}_{int_4 \cup int_5 \cup int_6}$.

3.7 Quelques résultats d'équivalence de processus

Nous énonçons et démontrons maintenant quelques propositions concernant l'équivalence entre processus. Dans cette section, nous utilisons la notation $C(N)$ pour désigner l'ensemble des canaux associés à la composante N. En d'autres termes, $C(N)$ est l'ensemble des canaux d'entrée et des canaux de sortie de N, cette notation est donc un raccourci pour noter $I(N) \cup O(N)$. Une conséquence du premier de ces résultats est que l'on peut affaiblir le moniteur d'un processus si ce moniteur contrôle des canaux qui ne sont ni des canaux d'entrée, ni des canaux de sortie, ce qui signifie que le processus muni du moniteur affaibli sera équivalent au processus initial.

Proposition 3.7.1 (Affaiblissement)**.**

$$[P]_M^N \approx [P]_{M'}^N$$

où M' est le moniteur donné par $f_{M'}(a) = \begin{cases} T & si\ f_M(a) = T\ ou\ a \notin C(N) \\ F & sinon \end{cases}$

Démonstration. Les seules règles pouvant être affectées sont les règles concernant le comportement des moniteurs face aux entrées et aux sorties d'action, soient les règles $M3$ à $M6$. Regardons la règle $M3$: la condition d'emploi de la règle stipule que l'on doit avoir $a \in \mathrm{O}(N)$ et $f_M(a) = V$, il faut donc démontrer que l'on a

$$a \in \mathrm{O}(N) \wedge f_M(a) = T \iff a \in \mathrm{O}(N) \wedge f_{M'(a)} = T.$$

Ce qui se fait comme suit :

$$\begin{array}{rcl} a \in \mathrm{O}(N) \wedge f'_M(a) = T & \iff & a \in \mathrm{O}(N) \wedge (f_M(a) = T \vee a \notin \mathrm{C}(N)) \\ & \iff & (a \in \mathrm{O}(N) \wedge f_M(a) = T) \vee (a \in \mathrm{O}(n) \wedge a \notin \mathrm{C}(N)) \\ & \iff & a \in \mathrm{O}(N) \wedge f_M(a) = T. \end{array}$$

À noter que l'affirmation $a \in \mathrm{O}(n) \wedge a \notin \mathrm{C}(N)$ est toujours fausse puisque $\mathrm{O}(N) \subset \mathrm{C}(N)$. Les règles $M4$ à $M6$ se traitent de la même façon. □

Proposition 3.7.2 (Distribution sur $\|$)**.** *Si les noms N, N_1 et N_2 respectent les conditions suivantes :*

– $I(N_1) \cap O(N_2) \cap \{a : f_M(a) = F\} = O(N_1) \cap I(N_2) \cap \{a : f_M(a) = F\} = \emptyset$

– $I(N') \cap O(N') \cap \{a : f_M(a) = F\} = \emptyset \quad \forall N' \in \{N, N_1, N_2\}$

alors nous avons l'équivalence

$$[[P]^{N_1} \parallel [Q]^{N_2}]^N_M \approx [[P]^{N_1}_M \parallel [Q]^{N_2}_M]^N.$$

Cette proposition affirme que sous certaines conditions, un moniteur peut se distribuer sur différentes composantes, ou de façon inverse, plusieurs moniteurs peuvent être regroupés en un seul moniteur surveillant une composante centrale. Analysons les conditions exigées sur les canaux avant de procéder à la démonstration.

La première condition énonce simplement que les deux composantes N_1 et N_2 ne doivent pas avoir la possibilité de faire des communications contrôlées par le moniteur M. Par exemple, une action a pouvant sortir de N_1 et entrer dans N_2 ne pourrait plus le faire, suite à la distribution du moniteur, si celui-ci l'en empêchait.

La seconde condition précise que les actions pouvant entrer et sortir d'une composante ne doivent pas être contrôlées par le moniteur. Cette condition sert à empêcher des situations pouvant être engendrées par des processus tels que $a.\overline{a}.0$. En effet, supposons que $a \in \mathrm{O}(N_1) \cap \mathrm{I}(N_1)$ et $f_M(a) = F$, on a dans ces conditions la trace suivante :

$$\begin{array}{rcl} [a.\overline{a}.0]^{N_1} & \xrightarrow{\tau} & [\overline{a}.0]^{N_1} \parallel a.0 \\ & \xrightarrow{\tau} & [\overline{a}.0 \parallel a.0]^{N_1} \\ & \xrightarrow{c(a)} & [0]^{N_1} \end{array}$$

alors qu'aucune trace ne pourra mener le processus $[a.\overline{a}.0]^{N_1}_M$ à exécuter une communication. Procédons maintenant à la démonstration de la proposition.

Démonstration. Nous allons démontrer que l'ensemble $\mathcal{S}$ suivant

$$\begin{array}{rcl} \mathcal{S} & = & \Big\{ \Big(\big[[P]^{N_1} \parallel [Q]^{N_2} \parallel R_1 \big]^N_M \parallel S,\ \big[[P]^{N_1}_M \parallel [Q]^{N_2} \parallel R_2 \big]^N \parallel S \Big) : \\ & & S \in \mathcal{P}_{\mathcal{A}} \text{ et } R_1, R_2 \in \mathcal{P}_{\mathcal{A}} \text{ satisfont la condition } (*) \Big\} \end{array}$$

est une bissimulation. Ceci complétera la démonstration puisque, clairement, le couple $([[P]^{N_1} \parallel [Q]^{N_2}]^N_M,\ [[P]^{N_1}_M \parallel [Q]^{N_2}_M]^N)$ appartient à $\mathcal{S}$. La condition $(*)$ s'écrit ainsi :

1. – R_1 est formé d'actions a telles que $f_M(a) = T$ et $a \in \mathrm{I}(N)$ ou telles que $a \in \mathrm{O}(N_1) \cup \mathrm{O}(N_2)$.
   – R_2 est formé d'actions a telles que $a \in \mathrm{I}(N)$ ou telles que $f_M(a) = T$ et $a \in \mathrm{O}(N_1) \cup \mathrm{O}(N_2)$.

2. – R_1 contient toutes les actions a de R_2 qui appartiennent à $\mathrm{O}(N_1) \cup \mathrm{O}(N_2)$ ou qui appartiennent à $\mathrm{I}(N)$ et qui respectent $f_M(a) = T$.
   – R_2 contient toutes les actions a de R_1 qui appartiennent à $\mathrm{I}(N)$ ou qui appartiennent à $\mathrm{O}(N_1) \cup \mathrm{O}(N_2)$ et qui respectent $f_M(a) = T$.

De façon simplifiée, cette condition précise que les processus $R1$ et $R2$ doivent être le plus semblable possible, tout en étant uniquement formés par des actions pouvant théoriquement (en fonction des moniteurs ainsi que des contrôles locaux d'entrée et de sortie) provenir de l'un des processus P ou Q, ou bien d'un processus externe.

On procède alors par cas pour démontrer que $\mathcal{S}$ est bien une bissimulation. On ne s'intéresse qu'aux cas concernant les règles $M3$ à $M6$, les autres cas ne posant clairement pas de problème, puisqu'ils se traitent de façon identique à des cas rencontrés dans des propositions précédentes.

Premier cas : S évolue.

Le processus $[[P]^{N_1} \parallel [Q]^{N_2} \parallel R_1]_M^N \parallel S$ peut évoluer par l'une des règles $M5$ ou $M6$ où une action d'envoi émise par S tente de traverse le moniteur M, deux choix sont possibles : soit l'action est permise, alors l'évolution se fait par la règle $M5$, soit l'action n'est pas permise et l'évolution se fait par la règle $M6$.

Si $[[P]^{N_1} \parallel [Q]^{N_2} \parallel R_1]_M^N \parallel S \xrightarrow{\tau} [[P]^{N_1} \parallel [Q]^{N_2} \parallel R_1 \parallel a.0]_M^N \parallel S'$ avec $S \xrightarrow{a} S'$ où l'évolution est permise par la règle $M5$, on aura donc $f_M(a) = T$ et $a \in \mathrm{I}(N)$. On aura également $[[P]_M^{N_1} \parallel [Q]_M^{N_2} \parallel R_2]^N \parallel S \xrightarrow{\tau} [[P]_M^{N_1} \parallel [Q]_M^{N_2} \parallel R_2 \parallel a.0]^N \parallel S'$, puisque la règle $M5$ s'applique toujours ici. Il reste alors à montrer que $a.0 \parallel R_1$ et $a.0 \parallel R_2$ satisfont $(*)$. Puisque $a \in \mathrm{I}(N)$ et $f_M(a) = T$, le point 1 ne cause pas de problème et comme l'action est ajoutée aux deux processus R_1 et R_2, le point 2 est satisfait car les processus originaux le satisfont.

Si $[[P]^{N_1} \parallel [Q]^{N_2} \parallel R_1]_M^N \parallel S \xrightarrow{\tau} [[P]^{N_1} \parallel [Q]^{N_2} \parallel R_1]_M^N \parallel S'$ avec $S \xrightarrow{a} S'$ où l'évolution est permise par la règle $M6$, et donc $f_M(a) = F$ et $a \in \mathrm{I}(N)$. On aura par conséquent $[[P]_M^{N_1} \parallel [Q]_M^{N_2} \parallel R_2]^N \parallel S \xrightarrow{\tau} [[P]_M^{N_1} \parallel [Q]_M^{N_2} \parallel R_2 \parallel a.0]^N \parallel S'$ où l'évolution est cette fois permise par la règle $M5$, le moniteur qui bloquait l'action ayant été enlevé. Il faut alors montrer que R_1 et $a.0 \parallel R_2$ satisfont $(*)$. Le processus R_1 satisfait déjà le point 1, et pour $a.0 \parallel R_2$ il n'y a pas de problème puisque $a \in \mathrm{I}(N)$. Puisque R_1 est inchangé et $f_M(a) = F$, le point 2 est également respecté.

Second cas : R_1 évolue.

Une évolution de R_1 impliquant l'une des règle $M3$ à $M6$ peut soit correspondre à une tentative d'émettre une action d'envoi vers le processus externe S, soit correspondre à une tentative d'émettre une action d'envoi vers l'un des processus P ou Q.

Si $[[P]^{N_1} \parallel [Q]^{N_2} \parallel R_1]^N_M \parallel S \xrightarrow{\tau} [[P]^{N_1} \parallel [Q]^{N_2} \parallel R'_1]^N_M \parallel a.0 \parallel S$ où l'évolution est permise par la règle $M5$ avec $R_1 \xrightarrow{a} R'_1$, de la même façon que si $[[P]^{N_1} \parallel [Q]^{N_2} \parallel R_1]^N_M \parallel S \xrightarrow{\tau} [[P]^{N_1} \parallel [Q]^{N_2} \parallel R'_1]^N_M \parallel S$ où l'action émise par R_1 est bloquée par le moniteur M (par la règle de sortie bloquée $M6$), on traite ces cas comme ceux précédents.

Si $[[P]^{N_1} \parallel [Q]^{N_2} \parallel R_1]^N_M \xrightarrow{\tau} [[P \parallel a.0]^{N_1} \parallel [Q]^{N_2} \parallel R'_1]^N_M$ où l'évolution est permise par la règle $M3$ avec $R_1 \xrightarrow{a} R'_1$, alors on sait que $a \in \mathrm{I}(N_1)$. Alors, si le couple $([[P]^{N_1} \parallel [Q]^{N_2} \parallel R_1]^N_M,\ [[P]^{N_1}_M \parallel [Q]^{N_2}_M \parallel R_2]^N)$ appartient à $\mathcal{S}$, il faut avant tout montrer que a doit aussi être une action de R_2.

Puisque a est une action de R_1, par le point 1 de la condition $(*)$, soit $f_M(a) = T$ et $a \in \mathrm{I}(N)$, soit $a \in \mathrm{O}(N_1) \cup \mathrm{O}(N_2)$ et donc $a \in \mathrm{O}(N_2)$, puisque l'on sait que $a \in \mathrm{I}(N_1)$ et que par hypothèse $\mathrm{I}(N_1) \cap \mathrm{O}(N_1) = \emptyset$. Si $a \in \mathrm{I}(N)$, alors a est aussi une action de R_2 par le point 2 de $(*)$, et si $a \in \mathrm{O}(N_2)$, alors, puisque $a \in \mathrm{I}(N_1)$, on sait par hypothèse que $f_M(a) = T$ (en effet, par hypothèse $\mathrm{I}(N_1) \cap \mathrm{O}(N_2) \cap \{a : f_M(a) = F\} = \emptyset$). Donc, encore par le point 2 de $(*)$, a doit être une action de R_2.

Le même raisonnement qui nous a conduit au fait que a est une action de R_2 nous permet également de déduire que $[P]^{N_1}_M \parallel a.0 \xrightarrow{\tau} [P \parallel a.0]^{N_1}_M$. En effet, puisque a est une action de R_1, soit $a \in \mathrm{I}(N)$ et $f_M(a) = T$, dans ce cas le résultat suit immédiatement, soit $a \in \mathrm{O}(N_1) \cup \mathrm{O}(N_2)$ et alors $a \in \mathrm{O}(N_2)$ et donc $f_M(a) = T$. Finalement, on obtient que

$$[[P]^{N_1}_M \parallel [Q]^{N_2}_M \parallel R_2]^N \xrightarrow{\tau} [[P \parallel a.0]^{N_1}_M \parallel [Q]^{N_2}_M \parallel R'_2]^N$$

Si l'évolution est de la même forme, mais concerne la composante de Q plutôt que celle de P, le raisonnement est exactement le même.

Troisième cas : P évolue.

Dans ce cas, P tente d'émettre une action d'envoi vers l'extérieur de sa composante. Puisque la composante $[P]^{N_1}$ ne possède pas de moniteur, la sortie d'une action est permise par la règle $M3$ si et seulement si cette action appartient à l'ensemble $\mathrm{O}(N_1)$.

Si $[[P]^{N_1} \parallel [Q]^{N_2} \parallel R_1]^N_M \xrightarrow{\tau} [[P']^{N_1} \parallel [Q]^{N_2} \parallel a.0 \parallel R_1]^N_M$, alors $a \in \mathrm{O}(N_1)$. Si $f_M(a) = T$, on aura également, par la règle $M3$, $[[P]^{N_1}_M \parallel [Q]^{N_2}_M \parallel R_2]^N \xrightarrow{\tau} [[P']^{N_1}_M \parallel [Q]^{N_2}_M \parallel a.0 \parallel R_2]^N$.

Maintenant, si $f_M(a) = F$, alors, par la règle $M4$

$$[[P]_M^{N_1} \parallel [Q]_M^{N_2} \parallel R_2]^N \xrightarrow{\tau} [[P']_M^{N_1} \parallel [Q]_M^{N_2} \parallel R_2]^N$$

Il faut maintenant montrer que dans ce cas, les processus $a.0 \parallel R_1$ et R_2 respectent la condition $(*)$. La point 1 de la condition $(*)$ tient clairement pour $a.0 \parallel R_1$ et R_2 puisque $a \in \mathrm{O}(N_1)$ et que le processus R_2 est inchangé. La condition 2 est également respectée puisque $f_M(a) = F$.

Autres cas :

Bien sûr, si l'évolution se fait par le processus Q, le traitement se fait de façon identique au cas pour P. Il est également possible que l'évolution soit de la forme $\xrightarrow{\widetilde{a}}$, mais ce cas est tout à fait similaire au cas où l'action est émise par S.

Nous avons fait la démonstration dans un sens, l'autre sens de la preuve se fait de façon identique, nous abordons, seulement comme exemple, le cas suivant, qui semble être le plus complexe :

$$[[P]_M^{N_1} \parallel [Q]_M^{N_2} \parallel R_2]^N \xrightarrow{\tau} [[P']_M^{N_1} \parallel [Q]_M^{N_2} \parallel R_2]^N$$

où l'évolution est permise par la règle $M4$ avec $P \xrightarrow{a} P'$.

Dans ce cas, l'utilisation de $M4$ implique que $a \in \mathrm{O}(N_1)$ et $f_M(a) = F$, alors la règle $M3$ permet l'évolution $[[P]^{N_1} \parallel [Q]^{N_2} \parallel R_1]_M^N \xrightarrow{\tau} [[P']^{N_1} \parallel [Q]^{N_2} \parallel a.0 \parallel R_1]_M^N$, puisqu'il n'y a plus de moniteur sur la composante de P. Il faut donc montrer que $a.0 \parallel R_1$ et R_2 satisfont les points 1 et 2 de la condition $(*)$. C'est clair pour le point 1, puisque R_2 est inchangé et que $a \in \mathrm{O}(N_1)$, et le point 2 est également satisfait puisque $f_M(a) = F$.

Nous avons donc démontrer que

$$\begin{aligned} \mathcal{S} \;=\; \Big\{ \Big(\big[[P]^{N_1} \parallel [Q]^{N_2} \parallel R_1\big]_M^N \parallel S, \big[[P]_M^{N_1} \parallel [Q]_M^{N_2} \parallel R_2\big]^N \parallel S \Big) : \\ S \in \mathcal{P}_{\mathcal{A}} \text{ et } R_1, R_2 \in \mathcal{P}_{\mathcal{A}} \text{ satisfont la condition } (*) \Big\} \end{aligned}$$

est bien une bissimulation, ce qui démontre la proposition. □

Ce résultat peut facilement être généralisé au résultat suivant :

Corollaire 1 (Distribution sur $\parallel$)**.** *Si les noms N, N_i pour $i = 1, 2, \ldots, n$ respectent les conditions suivantes :*

- $I(N_i) \cap O(N_j) \cap \{a : f_M(a) = F\} = O(N_i) \cap I(N_j) \cap \{a : f_M(a) = F\} = \emptyset \quad \forall 1 \leq i, j \leq n$
- $I(N) \cap O(N) \cap \{a : f_M(a) = F\} = \emptyset$

Alors nous avons l'équivalence suivante :

$$[[P_1]^{N_1} \parallel [P_2]^{N_2} \parallel \cdots \parallel [P_n]^{N_n}]^N_M \approx [[P_1]^{N_1}_M \parallel [P_2]^{N_2}_M \parallel \cdots \parallel [P_n]^{N_n}_M]^N. \quad \square$$

Proposition 3.7.3 (Sélection)**.** *Soit N un nom et soit M_{a1}, M_{a2}, M_{b1} et M_{b2} des moniteurs tels que*

$$f_{M_{a1}}(a) \wedge f_{M_{b1}}(a) = f_{M_{a2}}(a) \wedge f_{M_{b2}}(a) \quad \textit{pour toute action } a$$

et tels que pour toute action a, si $f_M(a) = T$ pour $M = M_{a1}, M_{a2}, M_{b1}$ ou M_{b2}, alors on a que $a \notin C(N)$.

Alors

$$[[P]^{N_1}_{M_{a1}} \parallel [Q]^{N_2}_{M_{b1}}]^N \approx [[P]^{N_1}_{M_{a2}} \parallel [Q]^{N_2}_{M_{b2}}]^N.$$

Cette proposition affirme entre autres que, sous certaines conditions, si un moniteur contrôle les communications entre deux processus (par exemple, si $M_{b1} = \emptyset$ dans la proposition), alors il est possible de séparer ce moniteur en deux sous-moniteurs et d'appliquer ces sous-moniteurs aux deux processus à surveiller. Par exemple, si un processus envoi des données sur un canal, alors qu'un autre en reçoit, il est équivalent de surveiller celui qui reçoit ou celui qui envoi, ce qui est conforme à notre étude des réseaux informatiques. En effet, si deux machines sont connectées entre elles, il est possible de contrôler les communications d'une machine à l'autre en surveillant seulement une des deux machines, ou en partageant la surveillance sur les deux machines.

Démonstration. Cette démonstration se fait de façon semblable aux démonstrations précédentes, nous l'omettrons donc pour passer directement à des exemples d'applications des propositions.

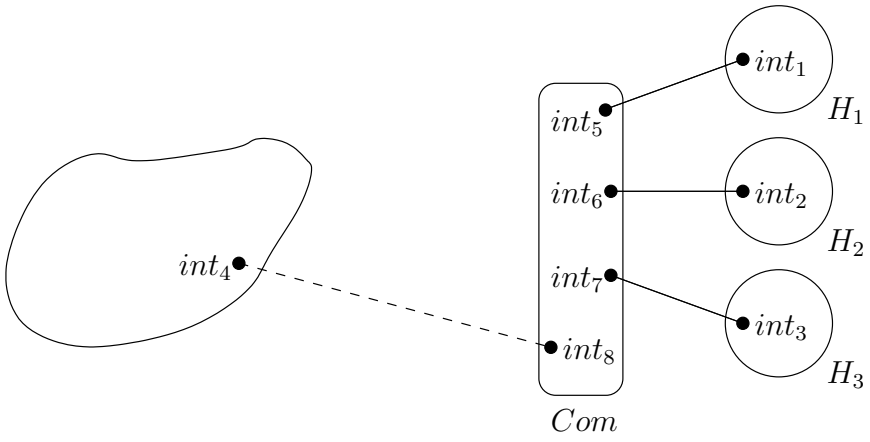
3.7.1 Exemples d'application des propositions

Application des propositions 3.7.1 et 3.7.2

Voici un premier exemple pour illustrer l'utilisation des propositions d'affaiblissement des moniteurs et de distribution sur l'opérateur de communication. Nous allons modéliser un petit réseau formé de trois hôtes et d'un commutateur, ce réseau pourra

communiquer avec l'extérieur, mais uniquement par l'intermédiaire du commutateur ; la figure 3.2 donne une représentation du réseau. Nous imaginerons alors un moniteur surveillant les communications du réseau avec le monde extérieur (qui pourrait représenter par exemple le réseau Internet). Les propositions 3.7.1 et 3.7.2 nous permettrons de déplacer le moniteur surveillant tout le réseau de sorte à ce qu'il surveille uniquement le commutateur, soit le seul sous-processus du réseau capable de communiquer avec l'extérieur.

FIG. 3.2 – Réseau de 3 hôtes relié à un autre réseau



$$
\begin{aligned}
H_i &\stackrel{\text{def}}{=} int_i(@_s, @_d, m).H_i + \overline{int}_{i+4}(@_s, @_d, m).H_i \\
Com &\stackrel{\text{def}}{=} \sum_{i=1}^{4} \overline{int}_i(@_s, @_d \neq @_s, m).int_{p(@_d)}(@_s, @_d, m).Com \\
&\quad + \sum_{i=1}^{4} \overline{int}_i(@_s, @_s, m).Com \\
R &\stackrel{\text{def}}{=} [[Com]^C \parallel [H_1|H_2|H_3]^H]^R
\end{aligned}
$$

où la fonction $p(@)$ qui définit le comportement du commutateur est donnée par

$$
p(@_x) = \begin{cases} x+4 & \text{si } x \in \{1,2,3\} \\ 8 & \text{sinon} \end{cases}
$$

alors que les ensembles d'entrées et de sorties des composantes sont :

$$
\begin{array}{llll}
\mathrm{I}(H) = \{int_5, int_6, int_7\} & & \mathrm{O}(H) = \{int_1, int_2, int_3\} \\
\mathrm{I}(Com) = \{int_1, int_2, int_3, int_4\} & & \mathrm{O}(Com) = \{int_5, int_6, int_7, int_8\} \\
\mathrm{I}(R) = \{int_4\} & & \mathrm{O}(R) = \{int_8\}
\end{array}
$$

Considérons maintenant que l'on munit le réseau R d'un moniteur M qui surveille les communications sur les canaux du réseau reliés avec l'extérieur, soient les canaux int_4 et

int_8. Les propositions 3.7.2 et 3.7.1 nous permettent de déplacer ce moniteur pour qu'il contrôle uniquement le commutateur, en somme nous avons les équivalences suivantes :

$$\begin{aligned} R &\stackrel{\text{def}}{=} [[Com]^C \parallel [H_1|H_2|H_3]^H]^R_M \\ &\approx \langle \text{ Proposition 3.7.2 } \rangle \\ &\quad [[Com]^C_M \parallel [H_1|H_2|H_3]^H_M]^R \\ &\approx \langle \text{ Proposition 3.7.1 } \rangle \\ &\quad [[Com]^C_M \parallel [H_1|H_2|H_3]^H]^R \end{aligned}$$

Application de la proposition 3.7.3

Considérons deux nouveaux réseaux H_1 et H_2 interconnectés et ne disposant d'aucune possibilité de communiquer avec l'extérieur. En notant i_1 l'interface de H_1 qui envoie vers H_2 et i_2 l'interface de H_2 qui envoie vers H_1, nous pouvons modéliser ce réseau ainsi :

$$H_1 \stackrel{\text{def}}{=} i_1(x).H_1 + \bar{i}_2(x).H_1$$

$$H_2 \stackrel{\text{def}}{=} i_2(x).H_2 + \bar{i}_1(x).H_2$$

$$R \stackrel{\text{def}}{=} [[H_1]^{N1} \parallel [H_2]^{N2}]^N$$

où

$$\begin{array}{lcll} \mathrm{I}(N1) & = & \{i_2\} & \mathrm{O}(N1) = \{i_1\} \\ \mathrm{I}(N2) & = & \{i_1\} & \mathrm{O}(N2) = \{i_2\} \\ \mathrm{I}(N) & = & \emptyset & \mathrm{O}(N) = \emptyset \end{array}$$

Soit maintenant M un moniteur contrôlant certaines communications entre $N1$ et $N2$, et donc contrôlant les canaux i_1 et i_2. Ce moniteur pourrait par exemple couper les communications correspondant aux transferts de certains types de paquet IP. Sans connaître la définition exacte du moniteur M, la proposition 3.7.3 nous permet d'établir les équivalences suivantes :

$$\begin{aligned} [[H_1]^{N1}_M \parallel [H_2]^{N2}_M]^N &\approx [[H_1]^{N1}_M \parallel [H_2]^{N2}]^N \\ &\approx [[H_1]^{N1} \parallel [H_2]^{N2}_M]^N. \end{aligned}$$

C'est-à-dire que le résultat sera le même si le moniteur contrôle le processus H_1, le processus H_2 ou bien les deux processus simultanément.

3.8 Les pare-feu externes

Pour empêcher une composante $[P]_M^N$ de communiquer certaines informations avec d'autres composantes, il suffit de configurer le moniteur M de sorte à ce qu'il bloque ces informations. Par exemple, si la composante correspond à un ordinateur, le moniteur pourrait représenter un pare-feu logiciel installé dans l'ordinateur. Pour contrôler le trafic entre deux machines, ou plus généralement entre deux réseaux, il est également possible d'ajouter au système une machine qui aura pour seul but d'effectuer un contrôle sur les paquets circulant sur le réseau. On parle alors d'ajout de pare-feu matériel, ou pare-feu externe.

Il est possible de représenter des pare-feu externes avec notre algèbre, et la façon de le faire est calquée sur la réalité matérielle. Supposons par exemple que nous disposions de deux machines reliées entre elles, et que pour une certaine raison (*e.g.* pour ne pas consommer trop de ressources système), nous voulions ajouter une troisième machine pour faire le contrôle des paquets passant d'une machine à l'autre. Il faudrait alors déconnecter les fils reliant les deux machines pour les connecter au pare-feu ajouté. C'est précisément ce que nous ferons avec nos processus. Voyons le fonctionnement des pare-feu externes avec le petit exemple de la figure 3.3.

FIG. 3.3 – Réseau formé de deux machines

Ce petit réseau peut être modélisé comme suit :

$$\begin{array}{lcl} H_1 & \stackrel{\text{def}}{=} & i_1(x).H_1 + \overline{i_2}(x).H_1 \\ H_2 & \stackrel{\text{def}}{=} & i_2(x).H_2 + \overline{i_1}(x).H_2 \\ R & \stackrel{\text{def}}{=} & [H_1 \parallel H_2]^{\emptyset}. \end{array}$$

Vu la simplicité du système, nous n'avons pas encapsulé chacun des hôtes dans une

composante, mais nous avons plutôt placé tout le réseau dans une composante simple $[\,]^{\emptyset}$ pour mettre l'emphase sur le fait que les deux hôtes n'ont aucun moyen de communiquer avec l'extérieur. On se rappelle cependant que les composantes ne font aucun contrôle sur les communications internes, les deux processus H_1 et H_2 sont donc libres de faire des échanges sur les canaux i_1 et i_2.

Le pare-feu externe devra recevoir des messages venant de l'un des processus H_1 ou H_2 et les retransmettre, après vérification, à l'autre processus. Ce pare-feu pourra être modélisé ainsi :

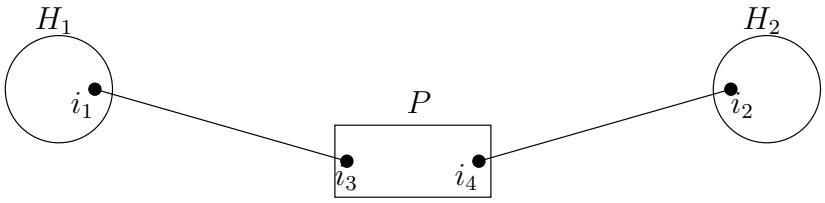
$$P \stackrel{\text{def}}{=} [\overline{i_1}(x).i_4(x).P + \overline{i_2}(x).i_3(x).P]^{\{i_1\uparrow,i_2\uparrow,i_3\downarrow,i_4\downarrow\}}.$$

Maintenant, pour illustrer que l'on coupe la connexion entre les hôtes pour la rétablir sur le pare-feu, il faut modifier H_1 et H_2 pour qu'ils écoutent P, on obtient alors les processus H_1' et H_2' ainsi que le nouveau réseau R' suivants :

$$\begin{array}{lcl} H_1' & \stackrel{\text{def}}{=} & i_1(x).H_1' + \overline{i_4}(x).H_1' \\ H_2' & \stackrel{\text{def}}{=} & i_2(x).H_2' + \overline{i_3}(x).H_2' \\ R' & \stackrel{\text{def}}{=} & [H_1' \parallel P \parallel H_2']^{\emptyset}. \end{array}$$

Ce nouveau réseau est illustré à la figure 3.4.

FIG. 3.4 – Le réseau R pourvu d'un pare-feu externe



3.8.1 Renforcement de politique de sécurité

En plaçant les deux hôtes à l'intérieur d'une composante comme suit :

$$H \stackrel{\text{def}}{=} [H_1' \parallel H_2']^{\{i_1\downarrow,i_2\downarrow,i_3\uparrow,i_4\uparrow\}},$$

nous obtenons la modélisation suivante du réseau :

$$R'' \stackrel{\text{def}}{=} [H \parallel P]^{\emptyset}.$$

Considérons maintenant un moniteur M surveillant les actions des processus H'_1 et H'_2, c'est-à-dire, un moniteur contrôlant les canaux i_1 et i_2, par lesquels ces processus peuvent émettre, ainsi que les canaux i_3 et i_4 par lesquels ils peuvent recevoir. Nous pouvons contrôler les communications du processus H en lui appliquant le moniteur M, ce qui donne

$$R_M \stackrel{\text{def}}{=} [H_M \parallel P]^{\emptyset}$$

où $H_M \stackrel{\text{def}}{=} [H]_M^{\{i_1\downarrow, i_2\downarrow, i_3\uparrow, i_4\uparrow\}}$.

Puisque le réseau R_M ne possède pas de possibilité de communiquer avec l'extérieur, la proposition 3.7.3 indique que le réseau R_M est bissimilaire au réseau R'_M obtenu en déplaçant le moniteur M vers le pare-feu P comme suit :

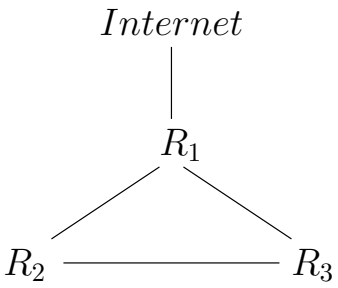
$$R'_M \stackrel{\text{def}}{=} [H \parallel P_M]^{\emptyset}$$

où $P_M \stackrel{\text{def}}{=} [\overline{i_1}(x).i_4(x).P + \overline{i_2}(x).i_3(x).P]_M^{\{i_1\uparrow, i_2\uparrow, i_3\downarrow, i_4\downarrow\}}$. Ainsi, la tâche de surveiller les émissions et les réceptions d'action des deux hôtes est entièrement réalisée par le pare-feu, et cela, avec le même résultat qu'une surveillance directe des hôtes.

3.9 Étude de cas

Nous présentons maintenant, de façon informelle, une petite étude de cas qui illustre l'implantation d'une politique de sécurité avec des contraintes sur un petit réseau.

FIG. 3.5 – Trois réseaux reliés à Internet



La figure 3.5 illustre trois sous-réseaux reliés entre eux et dont l'un d'eux, R_1, est relié à Internet. Supposons que l'on souhaite interdire tout échange entre deux réseaux de

paquet IP utilisant le protocole UDP, avec en plus la contrainte que seul le sous-réseaux R_2 peut être muni d'un moniteur.

Il faut donc recourir à l'implantation de pare-feu externe pour faire respecter la politique de sécurité. L'ordre de priorité pour l'ajout de composantes de sécurité sera le suivant :

1. Moniteur sur R_2 ;
2. Pare-feu externe.

Nous procédons en trois étapes pour implanter la politique de sécurité, ces trois étapes sont :

1. Modéliser le réseau ;
2. Créer un moniteur pour les composantes pouvant en être munies (ici seulement R_2) ;
3. Créer des pare-feu externes pour contrôler les liens restants (nous aurons ici besoin de deux pare-feu).

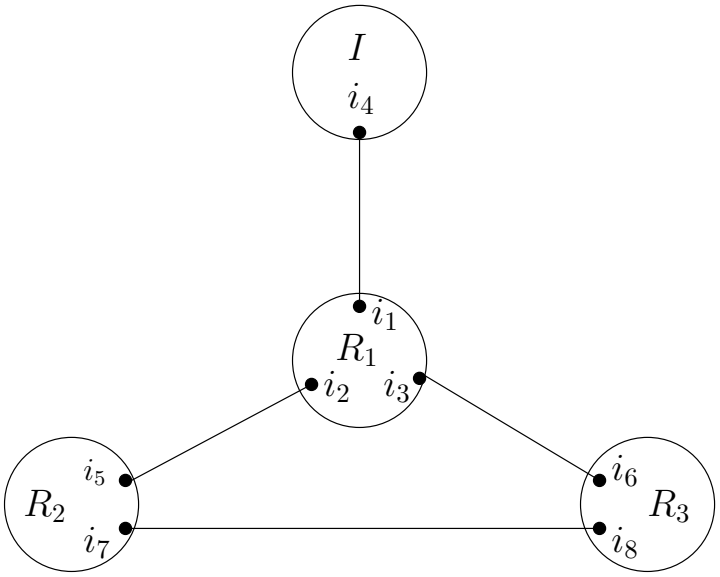
3.9.1 Modélisation du réseau

Dans ce problème, nous n'avons pas à considérer des détails tels que la provenance et la destination du message (les adresses source et destination), nous nous intéressons uniquement au protocole utilisé dans les paquets IP transigeant entre les réseaux. Nous pouvons donc considérer un paquet comme étant uniquement identifié par le protocole qu'il utilise (TCP, IP, ICMP, ...). De plus, nous n'avons pas à savoir, par exemple, si un paquet provenant d'*Internet* est passé par R_3 avant d'arriver à R_2. Alors, nous allons considérer les sous-réseaux (ainsi qu'Internet) comme des processus capable d'envoyer des paquets par leurs interfaces et capable de recevoir des paquets par le biais des interfaces auxquelles ils sont reliés.

La première étape consiste à assigner des interfaces aux différentes composantes, ce que l'on voit à la figure 3.6.

Nous obtenons alors les modélisations suivantes pour les composantes du réseau :

FIG. 3.6 – Numérotation des interfaces



$$
\begin{aligned}
I &\stackrel{\text{def}}{=} i_4(p).I + \overline{i}_1(p).I \\
R_1 &\stackrel{\text{def}}{=} \sum_{j=1}^{3} i_j(p).R_1 + \sum_{j=4}^{6} \overline{i}_j(p).R_1 \\
R_2 &\stackrel{\text{def}}{=} i_5(p).R_2 + i_7(p).R_2 + \overline{i}_2(p).R_2 + \overline{i}_8(p).R_2 \\
R_3 &\stackrel{\text{def}}{=} i_6(p).R_3 + i_8(p).R_3 + \overline{i}_3(p).R_3 + \overline{i}_7(p).R_3
\end{aligned}
$$

Par exemple, la modélisation du processus I indique que ce processus peut envoyer par l'interface i_4 un paquet utilisant un protocole p, et qu'il peut recevoir de l'interface i_1, laquelle appartient au processus R_1, un paquet également identifié par son protocole. À noter qu'ici p est une variable appartenant à l'ensemble des protocoles utilisables (TCP, UDP, ICMP, ...).

Cela donne la modélisation suivante pour le réseau :

$$R \stackrel{\text{def}}{=} [[I]^I \parallel [R_1]^{R_1} \parallel [R_2]^{R_2} \parallel [R_3]^{R_3}]^{\emptyset}$$

avec

$$
\begin{array}{llll}
\mathrm{I}(I) = \{i_1\} & & \mathrm{O}(I) = \{i_4\} \\
\mathrm{I}(R_1) = \{i_4, i_5, i_6\} & & \mathrm{O}(R_1) = \{i_1, i_2, i_3\} \\
\mathrm{I}(R_2) = \{i_2, i_8\} & & \mathrm{O}(R_2) = \{i_5, i_7\} \\
\mathrm{I}(R_3) = \{i_3, i_7\} & & \mathrm{O}(R_3) = \{i_6, i_8\}
\end{array}
$$

3.9.2 Création d'un moniteur pour R_2.

Selon les priorités établies précédemment, la politique de sécurité (interdisant le transfert de paquet utilisant le protocole UDP) doit être instaurée en privilégiant l'ajout d'un moniteur au sous-réseau R_2, ce que nous faisons ici.

Le sous-réseaux R_2 possède les interfaces i_5 et i_7 et il est relié aux interfaces i_2 et i_8. Ce sont ces interfaces que le moniteur de R_2 devra surveiller. Ce moniteur, que l'on identifiera par M_{R_2} sera représenté par la fonction $f_{M_{R_2}}$ définie ainsi :

$$f_{M_{R_2}}(x(p)) = \begin{cases} \text{F} & \text{si } x \in \{i_2, i_5, i_7, i_8\} \text{ et } p = \text{UDP} \\ \text{T} & \text{sinon} \end{cases}$$

Nous ne pouvons pas appliquer ce moniteur à chacune des composantes du réseau, puisqu'il est spécifié que seul le réseau R_2 peut être muni d'un moniteur. Cependant, les propositions d'affaiblissement (3.7.1) et de sélection (3.7.3) permettent de montrer qu'il est équivalent d'appliquer le moniteur à toutes les composantes, ou seulement à la composante du processus R_2. À noter par contre que ce n'est pas directement la proposition 3.7.3 qui s'applique ici, mais une version généralisés au cas où plus de deux processus sont présents. Donc, en appliquant le moniteur M_{R_2}, nous obtenons le réseau suivant :

$$R' \stackrel{\text{def}}{=} [[I]^I \parallel [R_1]^{R_1} \parallel [R_2]^{R_2}_{M_{R_2}} \parallel [R_3]^{R_3}]^{\emptyset}.$$

Dans ce réseau intermédiaire, les communications entre R_1 et R_2, ainsi que celles entre R_2 et R_3, sont contrôlées. Cependant, aucun contrôle n'est fait sur les communications entre I et R_1 et entre R_1 et R_3, de plus, il n'est pas possible de faire ce contrôle en modifiant le moniteur placé sur R_2. Selon nos contraintes, nous ne pouvons monitorer ni R_1, ni R_3, nous devons donc créer des pare-feu externes qui serviront en quelque sorte de douanes pour accéder à ces composantes.

3.9.3 Construction des pare-feu externes

Nous devons créer deux pare-feu externes, qui seront ajoutés au réseau pour instaurer la politique de sécurité, à la manière de ce qui est fait dans la section 3.8. L'un de ces pare-feu, P_1, aura pour tâche de contrôler le trafic entre Internet et le sous-réseau R_1,

alors que le second pare-feu, P_2, contrôlera le trafic entre les deux sous-réseaux R_1 et R_3.

Le pare-feu P_1 doit recevoir les paquets provenant de R_1 et destinés à I, il doit donc être relié à l'interface i_1, il effectue ensuite un contrôle sur ces données puis les retransmet par une interface, nommons-la i_{10}. Il faudra alors modifier le processus I pour qu'il soit relié à cette interface plutôt qu'à i_1. Le pare-feu doit également pouvoir faire le traitement inverse, soit recevoir des paquets de I, les contrôler, puis les retransmettre à R_1 ; il faudra donc également modifier le processus R_1 pour qu'il écoute le processus P_1 plutôt que le processus I. L'interface de P_1 reliée à R_1 sera notée i_9. Le pare-feu P_1 doit donc être comme suit :

$$P_1 \stackrel{\text{def}}{=} [\overline{i}_1(p).i_{10}(p).P_1 + \overline{i}_4(p).i_9(p).P_1]^{P_1}_{M_{P_1}}.$$

On obtiendra un pare-feu similaire pour contrôler le trafic entre R_1 et R_3 :

$$P_2 \stackrel{\text{def}}{=} [\overline{i}_3(p).i_{12}(p).P_2 + \overline{i}_6(p).i_{11}(p).P_2]^{P_2}_{M_{P_2}}.$$

On doit maintenant définir les fonctions $f_{M_{P_1}}$ et $f_{M_{P_2}}$ qui définissent le comportement des moniteurs M_{P_1} et M_{P_2} ; ces fonctions correspondent à la configuration des pare-feu. En fait, ces fonctions seront semblables à celle qui définit le moniteur sur R_2, on a alors :

$$f_{M_{P_1}}(x(p)) = \begin{cases} \text{F} & \text{si } x \in \{i_1, i_4\} \text{ et } p = \text{UDP} \\ \text{T} & \text{sinon} \end{cases}$$

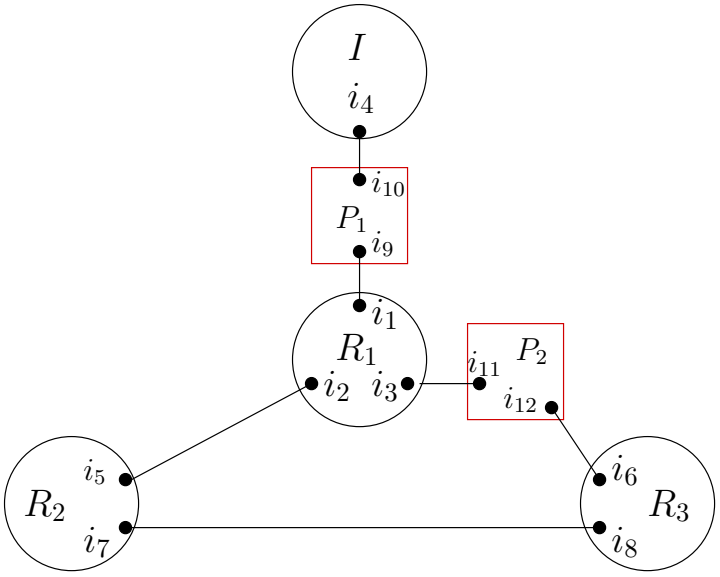
et

$$f_{M_{P_2}}(x(p)) = \begin{cases} \text{F} & \text{si } x \in \{i_3, i_6\} \text{ et } p = \text{UDP} \\ \text{T} & \text{sinon} \end{cases}$$

Il reste finalement à modifier les processus I, R_1 et R_3 pour refléter les changements de connexion entre les composantes.

$$\begin{aligned} I' &\stackrel{\text{def}}{=} i_4(p).I' + \overline{i}_{10}(p).I' \\ R'_1 &\stackrel{\text{def}}{=} \sum_{j=1}^{3} i_j(p).R'_1 + \overline{i}_5(p).R'_1 + \overline{i}_9(p).R'_1 + \overline{i}_{11}(p).R'_1 \\ R'_3 &\stackrel{\text{def}}{=} i_6(p).R'_3 + i_8(p).R'_3 + \overline{i}_7(p).R'_3 + \overline{i}_{12}(p).R'_3 \end{aligned}$$

FIG. 3.7 – Nouveau réseau avec composantes de sécurité



Le réseau final est alors

$$R'' \stackrel{\text{def}}{=} [[I']^{I'} \parallel [R_1']^{R_1'} \parallel [R_2]^{R_2}_{M_{R_2}} \parallel [R_3']^{R_3'} \parallel P_1 \parallel P_2]^{\emptyset}$$

avec

$$\begin{array}{lcllcl} \mathrm{I}(I) & = & \{i_{10}\} & \mathrm{O}(I) & = & \{i_4\} \\ \mathrm{I}(R_1) & = & \{i_5, i_9, i_{11}\} & \mathrm{O}(R_1) & = & \{i_1, i_2, i_3\} \\ \mathrm{I}(R_2) & = & \{i_2, i_8\} & \mathrm{O}(R_2) & = & \{i_5, i_7\} \\ \mathrm{I}(R_3) & = & \{i_7, i_{12}\} & \mathrm{O}(R_3) & = & \{i_6, i_8\} \end{array}$$

Comme spécifié, aucun transfert de paquet IP utilisant le protocole UDP n'est possible entre les différentes composantes du réseau. Le contrôle du trafic est réalisé par un moniteur contrôlant directement le sous-réseau R_2 et par deux pare-feu que nous avons ajoutés au réseau.

3.9.4 Problème inverse

Nous proposons maintenant d'analyser le problème inverse. Nous partons du processus obtenu à la section précédente et nous voulons obtenir une carte du réseau semblable à celle de la figure 3.5 (avec l'ajout des composantes de sécurité).

On détermine dans une première étape qu'il y aura six composantes, soit une composante par boîte [ ]. Ensuite, pour chacune des actions d'envoi que peut exécuter un processus, l'on ajoute un point (correspondant à une interface) à la composante qu'il représente. Par exemple, le processus P_1 possède deux actions d'envoi, soient $i_{10}(p)$ et $i_9(p)$, la composante associée possédera donc deux interfaces.

Pour chacune des interfaces d'une composante, nous traçons un lien vers chaque composante pouvant recevoir un paquet de celle-ci, par exemple, le processus P_1 sera relié à tous les processus pouvant exécuter $\bar{i}_9$ et $\bar{i}_{10}$, on obtient alors le schéma représenté à la figure 3.8.

FIG. 3.8 – Représentation intermédiaire

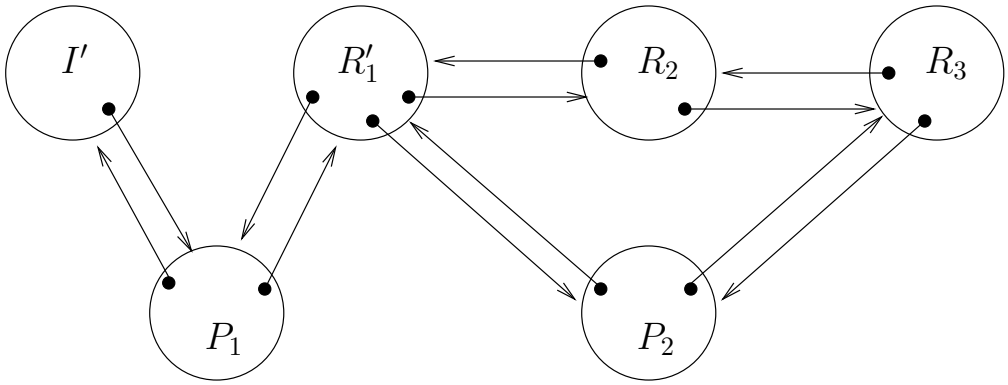
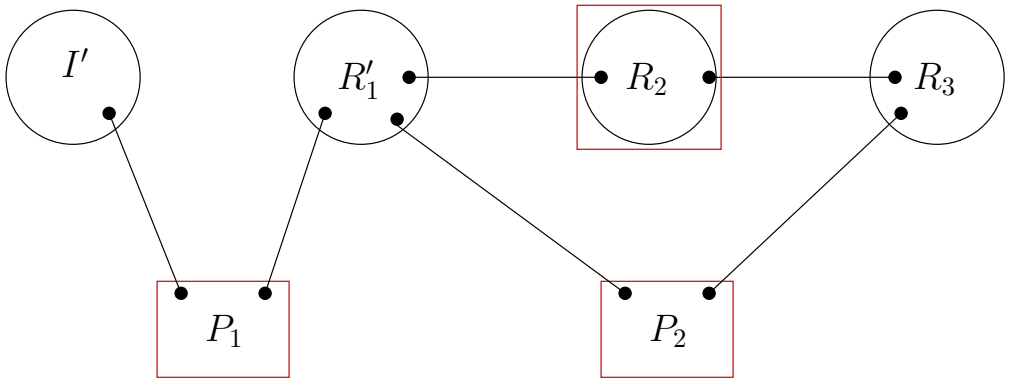


FIG. 3.9 – Représentation finale de R'



Quelques modifications au schéma de la figure 3.8 nous permettrons d'obtenir un résultat plus compréhensible. Premièrement, nous fusionnons les flèches allant et venant entre deux composantes en un seul lien passant d'une interface à l'autre. Deuxièmement,

nous marquerons différemment les processus P_1 et P_2 des autres processus, puisqu'ils se distinguent du fait qu'ils ne peuvent pas générer de nouveau message, en effet ils ne peuvent qu'en retransmettre. Finalement, nous ajoutons une boîte entourant le processus R_2' pour marquer qu'il est monitoré. Ceci donne le représentation illustrée à la figure 3.9.

3.10 Moniteurs évolutifs

Les moniteurs décrits à la section 3.2.3 représentent des pare-feu dits *stateless firewalls*, soient des pare-feu qui bloquent certains paquets et en laissent passer d'autres en exécutant un traitement individuel des paquets, c'est-à-dire, sans considérer les paquets précédents.

Les pare-feu dits *statefull firewalls* sont ceux capables de traiter des suites logiques de paquets pouvant représenter une attaque. Il est possible en effet que plusieurs paquets donnés, considérés individuellement, ne représentent pas une attaque, mais que ces mêmes paquets, placés dans un ordre séquentiel ou temporel précis, deviennent une attaque. Un pare-feu traitant les suites logiques de paquets pourra fort probablement avoir des taux de faux-positifs et de faux-négatifs bien inférieurs à ceux d'un pare-feu classique.

Ces pare-feu permettent de faire de la prévention d'intrusions. En détection d'intrusions, on analyse des suites de paquets pour noter dans un fichier d'audit les comportements suspects pouvant correspondre à une attaque. Il reste ensuite à analyser le ficher d'audit et à décider des mesures nécessaires si l'attaque est jugée sérieuse. La prévention d'intrusions est très proche de la détection d'intrusions, cependant, au lieu de simplement noter les comportements suspects pour une analyse ultérieure (qui préférablement devrait se faire en quasi temps réel), l'on bloque les messages qui peuvent correspondre à une attaque. Par exemple, si une trace contenant a et c est considérée comme une attaque s'il n'y a pas d'action b entre a et c, alors, suite à une action a, le pare-feu bloquera toute action c jusqu'à exécution de l'action b.

Une petite modification de la sémantique opérationnelle de l'opérateur de surveillance peut nous permettre de représenter des moniteurs évolutifs, qui représentent des *statefull firewalls*. Nous présentons cette nouvelle sémantique au tableau 3.8, un court exposé informel suivra pour expliquer une méthode pour représenter les moniteurs.

Dans cette sémantique, les moniteurs peuvent évoluer parallèlement au processus. Ainsi,

TAB. 3.8 – Sémantique opérationnelle de l'opérateur de restriction ([ ])

$(M1)$	$\dfrac{P \xrightarrow{c(a)} P'}{[P]^N_M \xrightarrow{c(a)} [P']^N_M}$	Communication
$(M2)$	$\dfrac{P \xrightarrow{\tau} P'}{[P]^N_M \xrightarrow{\tau} [P']^N_M}$	Action silencieuse
$(M3)$	$\dfrac{P \xrightarrow{a} P' \quad M \xrightarrow{a} M'}{[P]^N_M \xrightarrow{\tau} a.0 \parallel [P']^N_{M'}}\ a \in S(N),\ f_M(a) = V$	Sortie
$(M4)$	$\dfrac{P \xrightarrow{a} P' \quad M \xrightarrow{a} M'}{[P]^N_M \xrightarrow{\tau} [P']^N_{M'}}\ a \in S(N),\ f_M(a) = F$	Sortie bloquée
$(M5)$	$\dfrac{P \xrightarrow{a} P' \quad M \xrightarrow{a} M'}{P \parallel [Q]^N_M \xrightarrow{\tau} P' \parallel [a \| Q]^N_{M'}}\ a \in E(N),\ f_M(a) = V$	Entrée
$(M6)$	$\dfrac{P \xrightarrow{a} P' \quad M \xrightarrow{a} M'}{P \parallel [Q]^N_M \xrightarrow{\tau} P' \parallel [Q]^N_{M'}}\ a \in E(N),\ f_M(a) = F$	Entrée bloquée

les règles $M1$ et $M2$ sont identiques à celles de la sémantique des moniteurs fixes, puisque les moniteurs ne contrôlent pas les communications internes. Par contre, les autres règles comportent des différences avec la première sémantique, différences dues à l'évolution des moniteurs. Par exemple, pour la règle $M5$, si le moniteur M, en observant l'action a, devient le moniteur M', ce que l'on note $M \xrightarrow{a} M'$, alors, suite à l'entrée de l'action a, le processus $[P]^N_M$ devient le processus $[a.0 \parallel P]^N_{M'}$. Ceci correspond à la règle pour moniteur fixe, sauf que suite à l'entrée de l'action a, le moniteur M est remplacé par le moniteur M'. En somme, le moniteur M évolue pour prendre note du fait que l'action a a été observée.

Il existe maintenant différentes façons de représenter ces moniteurs évolutifs, nous en présentons une basée sur la théorie des automates.

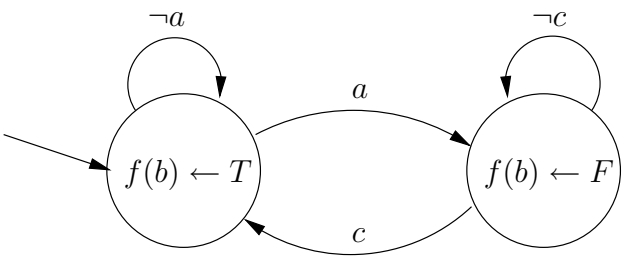
3.10.1 Représentation des moniteurs par des automates

Les moniteurs évolutifs peuvent être représentés par des automates de la manière suivante :

Une fonction de pare-feu initiale est définie, par exemple $f(x) \equiv T$ si toutes les actions sont *a priori* permises. Des arcs étiquetés par des actions permettent de passer d'un état à l'autre de l'automate. Puis, dans chacun des états, on précise la nouvelle fonction de pare-feu définissant le moniteur représenté par l'automate.

Passons à quelques exemples pour mieux comprendre le fonctionnement exact des automates. L'automate suivant permet de faire respecter la politique voulant qu'une action b ne puisse pas suivre une action a s'il n'y a pas d'action c entre les deux.

FIG. 3.10 – Automate représentant un moniteur



La fonction de pare-feu initiale est la fonction qui n'interdit aucune action, soit $f(x) \equiv T$, et l'automate reste dans l'état initial tant et aussi longtemps qu'une action a n'est pas exécutée. Si l'action a est exécutée, l'automate change d'état et la fonction de pare-feu est modifiée et devient alors

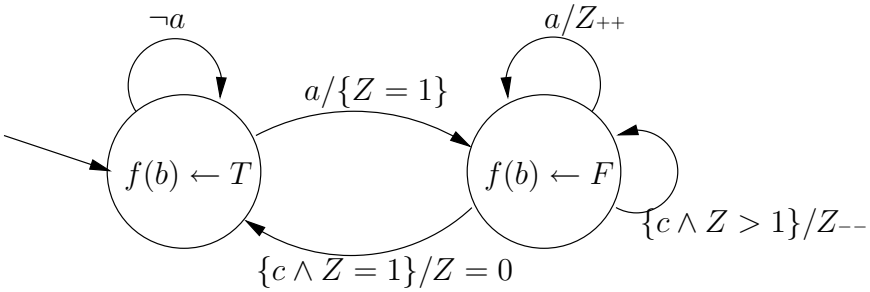
$$f(x) = \begin{cases} T & \text{si } x \neq b \\ F & \text{si } x = b \end{cases}$$

Ainsi $f(b) \leftarrow F$ signifie que la fonction f est modifiée de sorte à conserver toutes ses valeurs sauf $f(b)$ qui devient F. Tant et aussi longtemps que l'action c n'est pas exécutée, l'automate reste dans ce second état qui interdit l'exécution de l'action b. Si l'action c est exécutée, l'automate retombe dans l'état initial et l'action b est à nouveau permise.

Voyons maintenant un second automate légèrement plus complexe qui reprend l'idée du premier exemple en restreignant davantage la politique. Cette fois l'automate fait respecter la politique voulant que pour chaque action a exécutée, au moins une action c doit suivre pour que l'action b redevienne possible. Cela permet alors la suite $a.a.c.a.c.c.b$

mais interdit la suite *a.a.c.b* qui était permise par le premier automate. On doit donc ajouter une variable (Z) pour compter les actions a ainsi que les actions c. L'automate de la figure 3.11 fait respecter cette politique.

FIG. 3.11 – Automate représentant un moniteur muni d'un compteur



Nous arrêtons ici notre exposé informel sur les moniteurs évolutifs. Bien sûr, des résultats semblables à ceux sur les moniteurs fixes peuvent être développés. Ces résultats seraient encore plus intéressants puisque ces moniteurs permettent beaucoup plus de possibilités. Ceci pourrait donc faire l'objet de travaux futurs.

3.11 Conclusion

Nous avons présenté, dans ce chapitre, une nouvelle algèbre de processus qui est basée sur CCS et qui est adaptée à la modélisation de processus surveillés par des moniteurs. La syntaxe de l'algèbre inclut plusieurs des opérateurs de CCS, dont les opérateurs de préfixage et de choix, elle possède également un opérateur de communication, qui permet l'entrelacement d'actions et la création d'actions de communication, ainsi qu'un opérateur de surveillance qui permet la définition des moniteurs, et qui permet également de forcer les communications à la manière de l'opérateur de restriction de CCS. Le comportement des moniteurs de notre algèbre est basé sur le comportement des pare-feu fixes (*stateless firewalls*), c'est-à-dire que les moniteurs contrôlent les entrées et les sorties d'actions en détruisant les envois indésirables.

Suite à la définition de la syntaxe et de la sémantique opérationnelle de notre algèbre, nous avons défini une relation de bissimulation, inspirée de la relation de bissimulation faible de CCS, qui permet de comparer les processus en fonction des actions qu'ils peuvent exécutés et en fonction de leur interaction avec d'autres processus. Nous avons ensuite démontré quelques résultats concernant la congruence de cette relation pour les

opérateurs de préfixage, de communication et de surveillance, et concernant la distribution et la sélection des moniteurs dans un processus.

Nous avons terminé le chapitre en présentant deux exposés informels. Le premier de ces exposés discutait du renforcement d'une politique de sécurité dans un réseau en présence de contraintes concernant les processus pouvant être munis d'un moniteur. Alors que le second exposé informel présentait une extension de l'algèbre permettant de représenter des moniteurs représentant des *statefull firewalls*.

Conclusion et travaux futurs

Dans ce travail, nous avons présenté une nouvelle algèbre de processus. Cette algèbre, qui se base sur CCS, est essentiellement destinée à la sécurisation formelle de réseau informatique. Avant de nous attaquer à la définition de l'algèbre, nous avons exploré les approches développées en détection d'intrusions. En effet, ce domaine de recherche a fourni plusieurs outils permettant de sécuriser les réseaux informatiques, outils que l'on regroupe sous l'appellation de systèmes de détection d'intrusions. Nous avons donc exploré les deux principales approches dans le domaine : l'approche comportementale et l'approche par scénarios. La première de ces approches se base sur le fait que chaque utilisateur possède un profil qui lui est propre ; un agent tentant de pénétrer le système possédera fort probablement un comportement non compatible avec celui de l'utilisateur régulier, donnant ainsi une piste pour détecter une intrusion. La seconde approche, dite par scénarios, se base sur le fait qu'une attaque possède une signature qui peut être détectée sur une trace d'audit, cette approche nécessite donc la création d'une banque d'attaques.

Suite à l'étude des approches globales de détection d'intrusions, nous avons décidé de nous orienter vers un domaine très lié : la prévention d'intrusions. On peut voir la prévention d'intrusions comme étant une détection d'intrusions en temps réel dans laquelle le système, au lieu de donner une alerte avertissant d'une intrusion potentielle, réagit de sorte à empêcher l'attaquant de nuire, par exemple, on coupant la communication.

La seconde étape du travail a consisté en l'étude des langages destinés à la modélisation de systèmes communicants. Nous avons choisi de porter la majeure partie de nos efforts pour cette étape à l'étude des algèbres de processus. Plus précisément, nous nous sommes attardés aux algèbres inspirées de CCS, soient le π-calcul et le calcul *ambient* (en plus bien sûr de CCS).

Nous avons alors entrepris la partie principale du travail, qui consistait en l'élaboration d'une algèbre de processus destinée à la sécurisation de réseaux informatiques. Pour ce faire, nous avons choisi de nous baser sur CCS, que nous avons muni d'un nouvel

opérateur, que nous avons nommé opérateur de surveillance. Cet opérateur permet de définir le comportement d'un processus muni d'un moniteur qui contrôle ses communications. Le moniteur a le pouvoir d'absorber certains messages voulant pénétrer dans le processus ou voulant en sortir ; ce comportement des moniteurs a été inspiré du comportement des pare-feu dans un réseau, qui détruisent certains paquets pour les empêcher d'entrer dans un sous-réseau ou d'en sortir. Une première version de l'algèbre présentait des moniteurs fixes, c'est-à-dire représentant des pare-feu dits, dans la langue anglaise *stateless firewalls*, soient des pare-feu qui traitent chaque paquet individuellement, sans considération de la trace complète.

Après avoir défini notre algèbre de processus, c'est-à-dire après avoir défini une syntaxe et une sémantique opérationnelle, nous avons défini une relation d'équivalence entre les processus : la relation de bissimulation faible (qui est inspirée de la relation du même nom en CCS). Nous avons alors pu obtenir des résultats, que nous avons énoncés sous forme de propositions, concernant certaines équivalences de processus. En parallèle à cela, nous avons fourni quelques exemples de modélisations de réseaux simples se concluant par une étude de cas non formelle d'une implantation d'une politique de sécurité dans un réseau donné.

Pour obtenir une méthode entièrement automatisée pour générer des réseaux sécurisés par ajout de moniteurs à partir d'un réseau non sécurisé et d'une politique de sécurité, il nous faut maintenant développer (où choisir si celui-ci existe déjà) un langage formel propre à l'écriture des politiques de sécurité. Il restera alors à automatiser la configuration des moniteurs faisant respecter une politique de sécurité préalablement définie.

En plus de ces deux points cruciaux, une analyse plus rigoureuse doit être faite sur l'algèbre pour s'assurer qu'elle nous permet de modéliser convenablement, c'est-à-dire avec le niveau d'abstraction désiré, un réseau informatique possédant un nombre raisonnable de machines. En effet, les exemples donnés dans ce mémoire possèdent très peu de machines, contrairement à un réseau informatique standard. Il faut donc étudier la question à savoir s'il est possible de modéliser à l'aide de notre algèbre un réseau informatique de taille plus imposante, avec un degré de réalisme satisfaisant. Une étude détaillée devrait également être faite pour trouver une modélisation conforme pour les différentes pièces pouvant se trouver sur un réseau.

Toujours concernant notre algèbre de processus, le problème des messages perdus, qui n'a pas été aborder dans ce document, devra être analysé pour bien en comprendre ses conséquences et trouver comment minimiser son impact sur la vérification de propriété. En effet, l'opérateur de surveillance, qui rend asynchrone la communication entre des

sous-processus séparés par un moniteur, permet à certains messages de se perdre, et ainsi, il permet d'augmenter la taille d'un processus, et cette augmentation peut être sans limite. Prenons par exemple le simple processus suivant :

$$[P]^{\{a\downarrow\}}$$
$$P \stackrel{\text{def}}{=} a.P.$$

Ce processus peut évoluer par des actions silencieuses faisant sortir des actions a sans que ces actions ne soient exécutées :

$$\begin{array}{lcl} [P]^{\{a\downarrow\}} & \xrightarrow{\tau} & [P]^{\{a\downarrow\}} \parallel a.0 \\ & \xrightarrow{\tau} & [P]^{\{a\downarrow\}} \parallel a.0 \parallel a.0 \\ & \xrightarrow{\tau} & [P]^{\{a\downarrow\}} \parallel a.0 \parallel a.0 \parallel a.0 \\ & \xrightarrow{\tau} & \ldots \\ & \ldots & \end{array}$$

À chaque exécution, la taille du processus augmente et celle-ci peut donc devenir infinie. Cela peut créer des problèmes d'espace mémoire durant une simulation. Il faut donc, si possible, trouver un moyen de limiter cette progression sans nuire à l'expressivité de l'algèbre.

Finalement, il serait intéressant de développer un outil semblable au *Concurrency Workbench* (voir [16]), qui permet d'étudier des processus modélisés en CCS. On peut également envisager d'aller plus loin encore en développant une application graphique servant à la modélisation de réseaux et à la configuration de pare-feu. Cette application serait formée d'un éditeur graphique nous permettant de représenter un réseau informatique sous la forme d'un schéma, puis un module de traduction générerait une modélisation du réseau en suivant le schéma. Il ne resterait plus qu'à spécifier une politique de sécurité pour obtenir une configuration des pare-feu du réseau.

Bibliographie

[1] Adi, K., A. El-Kabbal et L. Pene, « Distributed firewall verification with mobile ambients », *Prooceedings of the workshop on practice and theory of access control technologies wptact'05*, Kamel Adi, L. L. e. M. M., éditeur, 2005.

[2] Alford, M. W., J. P. Ansart, G. Hommel, L. Lamport, B. Liskov, G. P. Mullery et F. B. Schneider, *Distributed systems : methods and tools for specification. An advanced course*, Springer-Verlag New York, Inc., 1985.

[3] Alpern, B. et F. B. Schneider, « Recognizing Safety and Liveness », *Distributed Computing*, 2(3), p. 117–126, 1987.

[4] Altisen, K., A. Clodic, F. Maraninchi et É. Rutten, « Using controller-synthesis techniques to build property-enforcing layers. », *Esop*, p. 174–188, 2003.

[5] Anderson, J., *Computer Security Threat Monitoring and Surveillance*, rapport technique, James P. Anderson Company, Fort Washington, Pennsylvania, 1980.

[6] Axelsson, S., *Research in Intrusion-Detection Systems : A survey*, rapport technique 98–17, Department of Computer Engineering, Chalmers University of Technology, Goteborg, Suède, décembre 1998.

[7] Bauer, L., J. Ligatti et D. Walker, *More enforceable security policies*, In Foundations of Computer Security, Copenhague, Danemark, Juillet 2002.

[8] Bergstra, J. A. et J. W. Klop, « Algebra of Communicating Processes with Abstraction », *Theoretical Computer Science*, 37(1), p. 77–121, mai 1985.

[9] Bergstra, J. A., *Handbook of process algebra,* Bergstra, Ponse et Smolka éditeurs, Elsevier Science Inc., New York, NY, USA, 2001.

[10] Browne, P., « The Security Audit », *Checking For Computer Security Center Sefl-Audits*, p. 173–184, 1979.

[11] Cannady, J., « Artificial neural networks for misuse detection », *Proceedings of the 1998 national information systems security conference (nissc'98) october 5-8 1998. arlington, va.*, p. 443–456, 1998.

[12] Cardelli, L. et A. D. Gordon, « Mobile ambients », *Foundations of software science and computation structures : First international conference, FOSSACS '98*, Springer-Verlag, Berlin Germany, 1998.

[13] Chen, L., S. Anderson et F. Moller, *A Timed Calculus of Communicating Systems*, rapport technique ECS-LFCS-90-127, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, 1990.

[14] Cisco System, *Cisco ids*, Disponible sur http ://www.cisco.com/en/US/products/sw/secursw/ps2113/index.html.

[15] Clarke, E. M. et E. A. Emerson, « Design and synthesis of synchronization skeletons using branching-time temporal logic. », *Logic of programs*, p. 52–71, 1981.

[16] Cleaveland, R., J. Parrow et B. Steffen, « The concurrency workbench : A semantics-based tool for the verification of concurrent systems », *ACM Transactions on Programming Languages and Systems*, 15(1), p. 36–72, January 1993.

[17] Couture, M., *Détection d'intrusions et analyse passive de réseaux*, mémoire de maîtrise, Département d'informatique et de génie logiciel, Université Laval, Québec, 2005.

[18] Crosbie, M., B. Dole, T. Ellis, I. Krsul et E. Spafford, *IDIOT—users guide*, rapport technique TR-96-050, The COAST Project, Dept. of Computer Science, Purdue University, West Lafayette, IN, USA, Septembre 1996.

[19] DEBAR, H., M. BECKER et D. SIBONI, « A Neural Network Component for an Intrusion Detection System », *Sp '92 : Proceedings of the 1992 ieee symposium on security and privacy*, p. 240–250, IEEE Computer Society, Mai 1992.

[20] Debar, H., *Application des réseaux de neurones à la détection d'intrusions sur les systèmes informatiques*, thèse de doctorat, Université Paris 6, Juin 1993.

[21] Denning, D., « An Intrusion-Detection Model », *IEEE Transaction on Sotfware Engineering*, 13(2), p. 220–232, Février 1987.

[22] Eckmann, S., G. Vigna et R. Kemmerer, *STATL : An Attack Language for State-based Intrusion Detection*, Dept. of Computer Science, University of California, Santa Barbara., 2000.

[23] Fox, K. L., R. R. Henning, J. H. Reed et R. P. Simonian, *A Neural Network Approach Towards Intrusion Detection*, rapport technique, Harris Corporation, 1990.

[24] Hoare, C., *Communication Sequential Processes (CSP)*, Prentice Hall International, Englewood Cliffs (NJ), USA, 1985.

[25] Ilgun, K., « USTAT : A Real-Time Intrusion Detection System for UNIX », *Proceedings of the 1993 IEEE symposium on research in security and privacy*, p. 16–28, Oakland, CA, 1993.

[26] Internet Security Systems, *Realsecure*, Disponible sur http ://www.iss.net/products_services/intrusion_detection.php.

[27] K. Iglun, R. K. et P. Porras, « State Transition Analysis : A Rule-based Intrusion Detection System », *IEEE Transaction on Software Engineering*, 21(3), 1995.

[28] Kumar, S. et E. Spafford, *An Application of Pattern Matching in Intrusion Detection*, rapport technique 94-013, Purdue University, Department of Computer Sciences, 1994.

[29] Kumar, S. et E. Spafford, « A Software Architecture to Support Misuse Intrusion Detection », *Proceedings of the 18th national information security conference*, p. 194–204, Baltimore, MD, Octobre 1995.

[30] Kumar, S., *Classification and Detection of Computer Intrusions*, thèse de doctorat, Purdue University, Purdue, IN, 1995.

[31] Lacasse, A., M. Mejri et B. Ktari, « Formal implementation of network security policies », *Second annual conference on privacy security and trust*, p. 161–166, Wu Centre, University of New Brunswick, Fredericton, Privacy, security and trust 2004 conference, octobre 2004.

[32] Lamport, L., « Proving the Correctness of Multiprocess Programs », *IEEE Trans. Software Eng.*, 3(2), p. 125–143, 1977.

[33] Lichtenstein, O. et A. Pnueli, « Checking that finite state concurrent programs satisfy their linear specification », *Popl '85 : Proceedings of the 12th acm sigact-sigplan symposium on principles of programming languages*, p. 97–107, New York, NY, USA, ACM Press, 1985.

[34] Manna, Z. et A. Pnueli, *Verification of concurrent programs, Part I : The temporal framework*, rapport technique CS-TR-81-836, Stanford University, Stanford, CA, USA, 1981.

[35] Maraninchi, F. et Y. Rémond, « Argos : an automaton-based synchronous language », *Computer Languages* No 27*, Elsevier pub*, p. 61–92, 2001.

[36] Milner, R., J. Parrow et D. Walker, « A calculus of mobile processes, i », *Inf. Comput.*, 100(1), p. 1–40, 1992a.

[37] Milner, R., J. Parrow et D. Walker, « A calculus of mobile processes, ii », *Inf. Comput.*, 100(1), p. 41–77, 1992b.

[38] Milner, R., *A Calculus of Communicating Systems*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.

[39] Milner, R., « Lectures on a calculus for communicating systems », *Seminar on concurrency, Carnegie-Mellon University*, p. 197–220, London, UK, Springer-Verlag, 1985.

[40] Milner, R., *The polyadic pi-calculus : a tutorial*, *Logic and algebra of specification*, Bauer, F. L., W. Brauer et H. Schwichtenberg, éditeurs, p. 203–246, Springer-Verlag, 1993.

[41] Milner, R., *Communication and Concurrency*, Prentice Hall International (UK) Ltd., Hertfordshire, UK, 1995.

[42] Mé, L., Z. Marrakchi, C. Michel, H. Debar et F. Cuppens, « La détection d'intrusion : les outils doivent coopérer », *Revue de l'Électricité et de l'Électronique*, (5), p. 50–55, Mai 2001.

[43] Mé, L. et V. Alanou, « Détection d'intrusion dans un système informatique : méthodes et outils », *TSI*, 96(4), p. 429–450, 1996.

[44] Mé, L. et C. Michel, *Intrusion detection : A bibliography*, rapport technique SSIR-2001-01, Supélec., Rennes, France, Septembre 2001.

[45] Mé, L., *Audit de sécurité par algorithmes génétiques*, thèse de doctorat, Université de Rennes 1, juillet 1994, Numéro d'ordre 1069.

[46] Mé, L., *Un complément à l'approche formelle : la détection d'intrusions*, Journée CIDR97, Rennes, Octobre 1997.

[47] Nielson, F., H. R. Nielson, R. R. Hansen et J. G. Jensen, « Validating firewalls in mobile ambients », *Concur '99 : Proceedings of the 10th international conference on concurrency theory*, p. 463–477, London, UK, Springer-Verlag, 1999.

[48] Parrow, J., *An introduction to the pi-calculus,* Dans *Handbook of process algebra,* Bergstra, Ponse et Smolka éditeurs, p. 479–543, Elsevier, 2001.

[49] Ramadge, P. et W. Wonham, « On the supremal controllable sublanguage of a given language », *SIAM J. Control and Optimization*, 25(3), 1987.

[50] Schneider, F. B., « Enforceable security policies », *Information and System Security*, 3(1), p. 30–50, 2000.

[51] Stirling, C., « Modal and temporal logics for processes », *Proceedings of the VIII Banff Higher order workshop conference on logics for concurrency : structure versus automata*, p. 149–237, Secaucus, NJ, USA, Springer-Verlag New York, Inc., 1996.

[52] Teller, D., P. Zimmer et D. Hirschkoff, « Using ambients to control resources », *In proceedings of concur'02, lncs 2421*, Springer, 2002.

[53] Vigna, G., R. A. Kemmerer et P. Blix, « Designing a web of highly-configurable intrusion detection sensors », *Raid '00 : Proceedings of the 4th international symposium on recent advances in intrusion detection*, p. 69–84, Springer-Verlag, 2001.

[54] Vigna, G. et R. A. Kemmerer, « NetSTAT : A Network-Based Intrusion Detection Approach », *Acsac '98 : Proceedings of the 14th annual computer security applications conference*, p. 25–, Washington, DC, USA, IEEE Computer Society, 1998.