Une étude des jeux distribués

Julien Bernet, David Janin

LaBRI, CNRS UMR 5800
Laboratoire Bordelais de Recherche en Informatique

Rapport de recherche RR-1455-12, daté du 31 aout 2011

Une étude des jeux distribués

par

Julien Bernet et David Janin,
LaBRI, IPB, Université de Bordeaux

Résumé

Dans cette article, nous étudions quelques propriétés des jeux distribués : un modèle orienté sémantique qui vise à capturer nombre de formalismes de spécification de problèmes de synthèse de programmes distribués.
Nous examinons en particulier les interconnections étroites qui existent entre les automate d'arbres alternants et les jeux distribués. En utilisant notamment le théorème de simulation de Muller et Schupp et une notion de composition séquentielle d'automates d'arbres, nous obtenons des outils conceptuels simples et élegants permettant de résoudre, sous certainnes hypothèses de linéarité, les jeux distribués.
Parce que les jeux distribués permettent aussi bien la spécification de comportement synchrone ou asynchrone, nous pouvons aussi énoncer et prouver une intuition assez commune en conception de systèmes distribués : dans le cas d'une capacité mémoire bornée, tout système distribué asynchrone se réduit à un système synchrone. Formellement, nous démontrons que tout jeu distribué asynchrone peut être transformé en un jeu distribué synchrone de même taille qui lui est équivalent au sens où les deux jeux admettent, essentiellement, les mêmes stratégies distribuées à mémoires finies.

In this paper we survey some properties of discrete distributed games : a semantical model that aims at capturing many formalisms for specifying distributed program synthesis problems.
We describe in particular the tight connection that exist between alternating tree automata theory. Using Muller and Schupp's simulation theorem and a notion of sequential composition of tree automata, we provide simple and elegant tools to solve than can solve, under adequate linearity hypothesis, distributed games.
Because distributed games enable the specification of both synchronous and asynchronous behaviors, we also state and prove what use to be a common intuition in distributed system design : when memory capacity is bounded, asychronous system are essentially synchronous. More formally, we prove that any asynchronous distributed game can be transformed into an equivalent synchronous games in the sense that they essentially both share the same finite state distributed strategies.

Table des matières

Introduction

Les jeux distribués dont il est question dans cette article, sont issus du travail de thèse du premier auteur [1] sous la direction du second [7]. Leur définition et leur étude, présentées dans une série de conférences [15, 2, 3, 8, 4], visent a offrir un modèle orienté sémantique, indépendant de toute architecture, pour l'étude et l'analyse des problèmes de spécification et de conception de systèmes distribués.

Des systèmes distribués

Plus précisément, nous considérons ici des systèmes constitués de plusieurs composants, largement autonomes, qui communiquent et interagissent afin d'atteindre, ensemble, un certain comportement globale. Ces systèmes sont dit ouverts au sens où, placés dans un certain environnement d'execution, ils sont soumis à des sollicitations externes qui influencent en permanence leurs comportements.

De tels systèmes sont maintenant omniprésent dans notre quotidien. Par exemple, les véhicules modernes sont aujourd'hui équipés de nombreux capteurs qui collectent à chaque instant de nombreuses données de provenances et de natures variés. Des contrôleurs informatisés, associés aux organes du véhicule tels que son moteur, ses freins, sa direction, son navigateur, ..., ont pour tâche, à chaque instant, de calculer à partir des données reçues par tout ou partie des capteurs, le comportement souhaité du composant qu'ils contrôlent.

Bien que le comportement de chaque composant soit souvent relativement simple, le comportement résultant du système global peut être particulièrement complexe. Quelques dépendances et contraintes de cohérences entre des composants distincts peuvent induire, à partir de comportements locaux pourtant acceptables, des comportements résultants erronés. Que doit être, par exemple, le comportement d'un véhicule soumis tout à la fois à une action de freinage sur ses roues et à une accélération du moteur ? Ce type d'erreurs, qui porte sur le comportement résultant de l'interaction entre plusieurs composants, est non seulement difficile à identifier mais peut aussi se révéler particulièrement difficile à corriger ; toute modification locale du comportement de l'un des composants du système peut avoir des répercutions drastiques sur le comportement global du système.

Et quelques hypothèses simplificatrices

Le modèle proposé ici, vise donc à comprendre et à identifier un peu mieux les difficultés intrinsèques liées à la conception de systèmes distribués. Mais ce modèle ne saurait prétendre à être universelle. Il est beaucoup plus abstrait que les systèmes distribués réels évoqués ci-dessus. De nombreux concepts, pourtant essentiels dans la mise au point de ces systèmes tel que, par exemple, la temporisation des exécutions, sont tout simplement absent de notre modèle.

Plus précisément, notre modèle est défini en suivant certaines *hypothèses simplificatrices*, présentées ci-dessous. Parfois, elles pourront paraître arbitraires ou excessives aux lecteurs experts en conception de systèmes distribués. Mais ces hypothèses auront le mérite d'être explicites. En particulier, notre objectif n'est pas de défendre la pertinence de ces hypothèses. Ce serait une discussion qui sortirait complètement du cadre de cette article. Au contraire, elles accompagnent, en l'état, le modèle des jeux distribués présentés ici.

La première hypothèse porte sur le temps inhérent aux systèmes modélisés. Nous le supposons discret, divergent et implicite. Plus précisément, nous supposons dans ce modèle que le temps s'écoule de façon discrète, chaque instant suivant l'autre après un délais minimum mais inconnu qui garanti qu'une infinité d'évènements conduit à un écoulement du temps infini.

La seconde hypothèse porte sur les données manipulées par ces systèmes. Nous les supposons, dans la plupart des cas, en nombre et à valeur fini. En particulier, la mesure de l'écoulement du temps ne peut pas constituer, du moins en toute généralité, une donnée manipulée par le système.

La troisième hypothèse, qui s'appuie sur les deux premières, est qu'a chaque instant, les systèmes considérés sont caractérisés par des états globaux courants - en nombre fini - et que l'évolution de ces systèmes peut être modélisé par une succession de transitions "instantanées" faisant passer le système d'un état global à un autre. En théorie de la concurrence, on se placera donc dans le cadre d'une sémantique par "interleaving" bien distincte de la sémantique "true concurrency", plus riche, qui pourrait être envisagée.

La quatrième hypothèse consiste à privilégier dans notre approche la sémantique (comportementale) des systèmes sur la syntaxe (ou la structuration) des données manipulés. Par exemple, qu'importe de savoir si les données d'un système sont structurés en 64 registres de 1 bit ou bien en un seul registre de 64 bit. Dans notre approche, sémantique, un tel système sera, dans tout les cas, modélisé par un état global de mémoire pouvant prendre 2^{64} valeurs.

Par ailleurs, dans notre approche, les architectures de communication décrivant la façon dont les sites des systèmes modélisés resteront implicites. Qu'importe de savoir où est localisé telle ou telle valeur, seule son influence (partielle) sur le comportement du système importe. Ce parti pris nous permet en particulier de modéliser des architectures dynamiques. Par exemple, on peut modéliser un ensemble fini d'agents mobiles qui se déplacent et dont l'architecture de communication est liée, à chaque instant, à la distance entre les agents mobiles.

Pour finir et par contre, nous modélisons explicitement, pour chaque système distribué, un nombre fini de flots d'exécution de processus locaux qui contribuent et produisent ensemble, le comportement globale du système.

Ainsi, le comportement global des systèmes modélisés pourra être vu comme une succession (éventuellement infinie) d'alternances entre (1) une action globale, incontrôlable, de l'environnement d'exécution, résultant de l'état global du système (et des contraintes associées) définit par l'ensemble des états de contrôle de chaque processus, et (2) un ensemble d'actions locales (au plus une par flux d'execution), contrôlables et indépendantes, définissant les réponses locales de

chacun des composants actifs à la sollicitation globale de l'environnement.

Des jeux pour la spécification de programmes

Notre modèle s'appuie sur la notion de jeu formel [6] qui, en informatique, permet a priori de modéliser toutes les interactions possibles entre les programes à concevoir et l'environnement dans lequel ils évoluent.

Ainsi, afin de traiter le cas des systèmes distribués, nous sommes amenés à définir des jeux à plus de deux joueurs où s'affrontent un environnement et une équipe de processus.

Dans notre approche, les coups des Processus sont définis localement : chaque processus dispose de sa propre arène et il y joue comme il le ferait dans un jeu à deux joueurs. En particulier, à partir de toute position locale à son arène, l'ensemble des coups qu'il peut jouer est indépendant de la position globale du système.

Au contraire, les coups du joueur Environnement, qui représentent aussi bien les interactions avec l'extérieur - interaction avec l'utilisateur, par exemple - que les règles auxquelles le système est soumis sont globaux mais peuvent être contraint. Par exemple, pour modéliser un canal de communication fiable, l'environnement peut être contraint de recopier la valeur produite, en entrée du canal par le processus émetteur afin de la rendre disponible, en sortie du canal, par le processus récepteur.

Ainsi, dans le modèle proposé : spécifier un système distribué à concevoir revient donc à décrire, à travers un jeu distribué, l'ensemble des comportement locaux possibles des processus et l'ensembles des conséquences globales de ces comportments locaux. Concevoir un système distribué qui répond à cette spécification revient alors à choisir parmi les comportements locaux possibles des stratégies locales assurant, ensemble, le bon fonctionnement global du système.

Contexte de recherche et travaux connexes

Notre modèle s'inspire largement des travaux de Reif et al. [18, 19] dans lesquels la notion d'information partielle est utilisée pour définir un modèle de calculabilité avec informations distribuées.

Dans le contexte des systèmes finis et des spécifications comportementales linéaires, l'étude du problème de la synthèse de programmes sur architecture distribuées est formalisé par Pnueli et Rosner [20, 21]. Leurs travaux sont étendus à des spécifications arborescentes par Kupferman et Vardi ă[9]. Dans ces approches, la décidabilité des architectures linéaires est chaque fois démontré, le cas général étant indécidable. Dans le contexte légèrement différent de la théorie du contrôle, on peut aussi citer aussi les travaux de Wonham et al [11] ou encore les travaux de Vincent [23].

C'est dans les travaux de thèse de Madhusudan [12] qu'on peut trouver une synthèse de la plupart des fondement du domaine. Ce dernier obtient en particulier [13] une extension de la classe des architecture décidables en s'intéressant au

cas de spécifications comportementales composées uniquement de spécifications locales à chaque site.

La théorie des automates utilisés ici est, quant à elle, une variation de la théorie des automates alternants de Muller et Schupp [16, 17] qui offre un outillage conceptuel puissant autours de la théorie des automates d'arbres.

Structure de l'article

Nous présentons, dans une première partie, le modèle des *jeux distribués* et la notion, correspondante, de *stratégie distribuée* qui en découle.

Quelques propriétés élémentaires de ces jeux sont exposées. La modélisation implicite des communications par la restriction des coups possibles de l'environnement est illustré à travers le codage, en terme de jeux distribués, du problème de la synthèse de programmes sur une architecture pipeline.

Dans une seconde partie, les liens étroit qui relient cette définition et la théorie des automates d'arbres alternants de Muller et Schupp sont présenté. Ils offrent un outillage simple qui nous permet, sous certaines hypothèses de linéarité des jeux, de fournir un algorithme de résolution des jeux distribués : un algorithme permettant de décider et de construire s'il en existe les stratégies distribuées finies gagnantes du jeu fourni en entrée.

Les constructions présentées dans la littérature qui sous tendent les algorithmes de résolution de problèmes de synthèses distribués [20, 21, 11, 9] ou de résolution des jeux distribués [15], qui pourraient apparaître comme autant de constructions spécifiques, s'unifient alors dans notre approche à travers les notions, somme toute classique en théorie des automates, de composition séquentielle d'automate et de simulation d'automate d'arbres alternants par des automates d'arbres non alternants [16].

La décidabilité du problème de la synthèse de programmes sur l'architecture pipeline est, en particulier, une conséquence de la linéarité des jeux distribués qui le modélisent.

Le modèle des jeux distribués permet, tout à la fois, la modélisation de comportement locaux synchrones ou asynchrones. Il permet donc d'explorer formellement quelques propriétés de la spécification de système distribués dans des contextes synchrones ou asynchrones qui, en l'absence de modèle formel, reste du domaine de l'intuition. Ainsi, le modèle proposée nous permet en effet d'énoncer et de prouver l'intuition suivante : dans le cas de capacité mémoire globalement bornée, tout système distribué asynchrone se comporte, pour l'essentiel, comme un système synchrone.

Plus précisément, nous démontrons dans cet article que, si on se restreint aux stratégies distribuées à mémoire fini, tout jeu distribué asynchrone peut être normalisé, de façon simple, en un jeu distribué synchrone, équivalent au jeu initial au sens où les stratégies distribuées gagnantes qu'ils induisent sont, pour l'essentiel, les mêmes.

Un bémol cependant, la normalisation synchrone proposé appliqué à des jeux linéaires asynchrones peut en toute généralité briser la linéarité des jeux dont il

sont issues. Une procédure de linéarisation a posteriori est donc proposé pour remédier à cela.

Nous concluons alors notre étude en évoquant quelques pistes de recherche qui mériterait sans doute d'être poursuivie et qui montre, en un sens, quelques limites du modèle.

1 Jeux distribués

Nous définissons dans cette section la notion de jeux distribués et la notion de stratégies distribuées qui l'accompagne. Nous illustrons ensuite cette définition par la construction de jeux distribués à partir des dites *pipelines* qui nous servira de fil rouge tout au long de notre présentation.

1.1 Notations

On utilise dans cette article des notations largement standards en théorie des mots. Nous précisons cependant les notations utilisées afin d'éviter toute ambiguïté.

Pour tout alphabet A, on note A^* (resp. A^ω) l'ensemble de tout les mots finis sur l'alphabet A. On utilise aussi les notations $A^\infty = A^* \cup A^\omega$ et $A^? = \{\epsilon\} + A$.

Etant donné $L \subseteq A^*$, on note aussi L^+ (resp. L^ω lorsque $\epsilon \notin L$) l'ensemble des mots finis (respectivement infinis) construit en concaténant un nombre fini (resp. un nombre infini) de mot de L.

Pour tout mot fini $w = a_1 \dots a_n$, on note $|w| = n$ la longueur du mot w. Pour tout mot infini w, on note $\inf(w) = \{a \in A \mid w \in (A^*.a)^\omega\}$ l'ensemble des lettres de A qui apparaissent infiniment dans le mots w.

Etant donné deux ensembles A et X, pour tout mot $w \in A^*$, on note aussi $\pi_X(w)$ la projection du mot w sur l'alphabet X c'est à dire le mot obtenu à partir de w en supprimant toute les lettres qui n'appartiennent pas à X.

Etant donné n ensembles $A_1, \dots, A_n$, étant donné $A = A_1 \times \dots \times A_n$, étant donné un ensemble d'indices $I = \{i_1, \dots, i_k\} \subseteq \{1, \dots, n\}$ avec $i_1 < i_2 < \dots < i_{k-1} < i_k$, on note $A[I]$ l'ensemble $A[I] = A_{i_1} \times \dots A_{i_k}$. Pour tout $x \in A$ de la forme $x = (a_1, \dots, a_n) \in A$, on note aussi $x[I]$ le k-uplet définit par $x[I] = (x_{i_1}, \dots, x_{i_k}) \in A[I]$. Enfin, pour tout ensemble $P \subseteq A$, on note $P[I]$ l'ensemble définit par $P[I] = \{x[I] \in A[I] |\ x \in P\}$.

Dans le cas ou l'ensemble d'indices I est de la forme $I = \{i, i+1, \dots, j\}$ avec $1 \leq i \leq j \leq n$ on pourra simplifier ces notations en $A[i,j]$, $x[i,j]$ et $P[i,j]$ respectivement (allant jusqu'à noter $A[i]$, $x[i]$ et $P[i]$ lorsque $i = j$).

Ces notations s'étendent naturellement aux mots sur l'alphabet A de la façon suivante. Pour tout mot $w = a_1.a_2.\dots \in A^\infty$, pour tout $I \subseteq \{1, \dots, n\}$, on note $w[I] = a_1[I].a_2[I].a_3[I]\dots$. Ces notations s'étendent aussi au relations. Pour toute relation $R \subseteq A \times A$, on note $R[I]$ la relation sur $A[I]$ définit par $R[I] = \{(x[I], y[I]) \in A[I] \times A[I] |\ (x,y) \in R\}$.

Remarquons, en particulier, que la notation $w[i,j]$ ne désignera pas comme cela est parfois le cas le sous-mot de w constitué de la séquence des lettres situées entre la $i$ème à la $j$ème lettre de w. Il désignera ici le mot obtenu à partir de w en projetant chacune de ses lettres sur l'alphabet $A[i,j]$.

1.2 Arènes distribuées

Une *arène distribuée* à n processus est une arène de jeux construite à partir du produit de n arènes simples qui définit les coups possibles d'un joueur Environnement jouant seul contre n joueurs Processus numérotés de 1 à n.

Définition 1.2.1 (Arènes de jeux distribués)
Etant donné n arènes simples de la forme $G_i = \langle P_i, E_i, T_{P,i}, T_{E,i}\rangle$ pour $i \in [1,n]$, c'est à a dire définis, pour chaque indice i, par un ensemble P_i de positions du joueur processus i, un ensemble E_i de position du joueur Environnement, un ensemble $T_{P,i} \subseteq P_i \times E_i$ de coups possibles pour le joueur Processus et un ensemble $T_{E,i} \subseteq E_i \times P_i$ de coups possibles pour le joueur Environnement, une arène de jeux distribuée *construite à partir des arènes de jeux locales $\{G_i\}_{i\in[1,n]}$ est définit comme étant toute arène de jeu de la forme $G = \langle P, E, T_P, T_E\rangle$ satisfaisant les propriétés suivantes :*

1. Positions de l'environnement : $E = \prod_{i\in[1,n]} E_i$,
2. Positions des processus : $P = \prod_{i\in[1,n]}(E_i \cup P_i) - \prod_{i\in[1,n]} E_i$,
3. Coups des processus *: T_P est l'ensemble de toute les paires $(p,e) \in (P \times E)$ telle que, pour tout $i \in [1,n]$:*
   - *soit $p[i] \in P_i$ et $(p[i], e[i]) \in T_{P,i}$ (le processus i est* actif dans la position *p),*
   - *soit $p[i] \in E_i$ et $p[i] = e[i]$ (le processus i est* inactif dans la position *p),*
4. Coups de l'environnement *: T_E est un* **sous-ensemble** *de l'ensemble de toute les paires $(e,p) \in (E \times P)$ telle que, pour tout $i \in [1,n]$:*
   - *soit $p[i] \in P_i$ et $(e[i], p[i]) \in T_{P,i}$ (l'environnement active le processus i),*
   - *soit $p[i] \in E_i$ et $p[i] = e[i]$ (l'environnement garde le processus i inactif).*

Remarque 1.2.2 La définition des arènes distribuées, pas forcément simple en première lecture, appel quelques commentaires.

On remarque par exemple que les coups du joueur Environnement sont définit globalement. Au contraire, les coups de chaque joueur Processus, définit localement sur chaque arène, nous permettent de modéliser la connaissance partielle qu'à chaque processus de la position globale.

A chaque instant, le joueur Environnement a connaissance de la position globale du jeu. Au contraire, à chaque, dans une position globale p, le processus i ne connait *apriori* que sa propre position $p[i]$.

En particulier, un coup global de l'équipe des processus est constitué de toute combinaison de coups locaux des processus actifs. Sans préjuger de la qualité de la position globale du joueur Environnement qui sera ainsi atteinte - une telle

position pourra par exemple être perdante pour l'équipe des processus - toute combinaison de choix locaux est possible.

Remarquons cependant que la définition d'une arène distribuée nous invite aussi à restreindre les coups possibles du joueur Environnement. Certaines combinaison de coups localement définit pour l'environnement peuvent ne pas être autorisé. On comprend alors qu'on peut forcer l'environnement à transmettre de l'information d'une arène à l'autre. Dans ce cas, le transfert d'information est modélisé implicitement, au niveau sémantique, sans aucune référence a priori à une quelconque architecture de communication.

C'est cette dernière caractéristique qui permet de modéliser, en particulier, les problèmes de synthèse distribués à la Pnueli et Rosner [21, 10], ou bien les problèmes de synthèse de contrôleur à la Wonham [11]. Mais nous pourrions tout aussi bien modéliser des réseaux de processus avec des architectures de communications changeante. Le modèle proposé est *générique*.

Lorsque toute les combinaisons possibles de coups locaux de l'environnement sont autorisés, on obtient une arène maximale en un certain sens et qui mérite d'être identifiée. Plus précisément :

Définition 1.2.3 (Produit asynchrone libre)
Dans le cas où l'ensemble T_E des coups du joueur Environment de l'arène distribuée G est maximal, l'arène est appelé le produit asynchrone libre *des arènes $\{G_i\}_{i\in[1,n]}$ et elle est notée $G_1 \otimes G_2 \otimes \cdots \otimes G_n$.*

Remarque 1.2.4 Toute arène, simple ou distribuée, peut être vue comme un graphe bipartie orientée dont les sommets sont les positions de l'environnement et du ou des processus, et dont les arcs sont les coups possibles entres ces positions. On peut parler de plongement d'une arène dans une autre au sens des plongements de graphes qui préservent en outre la partition des sommets.

Un constat : toute arène distribuée G définit comme ci-dessus peut être plongée dans le produit asynchrone libre $G_1 \otimes G_2 \otimes \cdots \otimes G_n$.

Plus généralement, les arènes distribuées étant construite à partir d'un produit d'arènes locales, on peut définir une opération inverse de projection d'une arène distribuée. L'arène distribuée initiale se plongeant alors dans le produit de ses projections.

Formellement :

Définition 1.2.5 (Projection)
Pour tout sous-ensemble d'indices $I \subseteq [1, \cdots, n]$, la projection $G[I]$ *de G sur ses composantes d'indice I est définit comme l'arène $G[I] = \langle P', E', T'_P, T'_E\rangle$ avec $P' = P[I] - E[I]$ (éventuellement strictement incluse dans $P[I]$!), $E' = E[I]$, $T'_P = T_P[I] \cap (P' \times E')$ et $T'_E = T_E[I] \cap (E' \times P')$.*

Remarque 1.2.6 Avec cette définition, on constate que l'arène distribuée G, initialement définit à partir des arènes $\{G_i\}_{i\in[1,n]}$, peut maintenant être définit à partir des arènes $\{G[i]\}_{i\in[1,n]}$; pour tout $i \in [1, n]$ l'arène local $G[i]$ peut être

plongée dans l'arène local G_i et l'arène G peut être plongée dans le produit libre $G[1] \otimes G[2] \cdots \otimes G[n]$.

Plus généralement, pour tout sous-ensemble non vide $I \subset \{1, \ldots, n\}$, étant donné $\overline{I} = \{1, \ldots, n\} - I$, l'arène G distribuée sur n processus, peut, tout aussi bien, être vue comme une arène distribuée de deux processus produite à partir des arènes locales $G[I]$ et $G[\overline{I}]$. Ainsi, l'arène G peut être plongée dans le produit libre $G[I] \otimes G[\overline{I}]$ et chacune de ces arènes peut, le cas échéant, être vue comme une arène distribuée plus finement.

Autrement dit, la définition d'une arène distribuée à partir de composants locaux est, en un certain sens, associative. C'est cette propriété qui nous permet par la suite de réduire la plupart des preuves portant sur des jeux distribués à n processus à des preuves portant sur les jeux distribués à deux processus.

1.3 Jeux et stratégies distribuées

Pour définir la notion de jeux distribué, rappelons tout d'abord la définition d'un jeu simple. La distinction entre jeux distribués et jeux simples sera ensuite obtenue en restreignant les stratégies admissibles de l'équipe des processus à une certaine fammilel de stratégies : les stratégies distribuées.

Définition 1.3.1
Un jeu $G = \langle P, E, T_E, T_P, e_0, \mathcal{W} \rangle$ *est définit comme étant une arène de jeux* $\langle P, E, T_E, T_P \rangle$ *équipée d'un état initial* $e_0 \in E$ *et d'un ensemble de mot* $\mathcal{W} \subseteq (E + P)^\omega$.

Une partie*, à partir de la position initiale* $e_0 \in E$*, est définie comme un chemin (au sens de la théorie des graphes orientés) dans l'arène* G *issu de la position* e_0.

A tout instant, dans une position courante $x \in P \cup E$*, soit* $x \in E$ *et c'est au tour de l'environnement de jouer en choisissant une position* $y \in P$ *telle que* $(x, y) \in T_E$*, ou bien* $x \in P$ *et c'est alors à l'équipe des processus de jouer en choisissant une position* $y \in E$ *telle que* $(x, y) \in T_P$.

Ainsi, à chaque instant, l'avancement du jeu définit un mot $w \in E.(P.E)^*.P^?$*, issue de la position initiale* e*, qui peut être prolongé, selon le cas, par l'environnement ou le (ou l'équipe des) processus.*

Dans ce contexte, une stratégie pour l'environnement *est une fonction partielle* $\tau : E.(P.E)^* \to P$ *telle que, pour tout mot* $w.e \in dom(\tau)$ *avec* $w \in (E.P)^*$ *et* $e \in E$*, on a* $(e, \tau(w.e)) \in T_E$.

De la même façon, une stratégie pour le (ou l'équipe des) processus *est une fonction partielle* $\sigma : (E.P)^+ \to E$ *telle que, pour tout mot* $w.p \in dom(\sigma)$ *avec* $w \in E.(P.E)^*$ *et* $p \in P$*, on a* $(p, \sigma(w.p)) \in T_P$.

Une partie w *est alors dites* compatible *avec la stratégie* σ *(resp. la stratégie* τ*) lorsque, pour tout entier* n *tel que* $0 < n < |w|$*, si* $w(i) \in P$ *alors* $w(i+1) = \sigma(w(1,i))$ *(resp. si* $w(i) \in E$ *alors* $w(i+1) = \tau(w(1,i))$*) en notant* $w(i)$ *la* i*ème lettre du mot* i *et* $w(1,i)$ *le prefix de* w *de longueur* i.

On dit alors que la stratégie σ *des processus (resp. la stratégie* τ *de l'environnement) est une* stratégie gagnante*ă dans le jeu* G *à partir de la position*

initiale e_0 et avec la condition de gain $\mathcal{W}$ lorsque toute partie maximale issue de e_0 et compatible avec la stratégie σ (resp. la stratégie τ) est soit finie et termine sur une position de l'environnement (resp. des processus), soit infinie est appartient à $\mathcal{W}$ (resp. n'appartient pas à $\mathcal{W}$).

Cette définition plutôt générale ne permet pas de conclure quant à l'existence de stratégie gagnante pour l'environnement ou les processus. Le cas particulier des jeux finis est plus intéressant de ce point de vue là.

Définition 1.3.2
Le jeu G est dit fini lorsque P et E sont deux ensembles finis et lorsque $\mathcal{W}$ est un langage ω-régulier, i.e. définissable par un automate d'états finis de mots infinis.

Une stratégie de processus σ est dites strategie finie*ăou* stratégie à mémoire finie *lorsque qu'il existe un quadruplé $\mathcal{M} = \langle M, m_0, \mu : M \times (P \cup E) \to M, h : M \times P \to E \rangle$, appelé la mémoire de σ, avec M un ensemble fini d'états, $m_0 \in M$ un état initial, μ une fonction de transition et h une fonction de choix telle que, pour toute partie $w \cdot p \in dom(\sigma)$ on a $\sigma(w.p) = h(\mu^*(m_0, w), p)$ avec μ^* inductivement définit par $\mu^*(m, \epsilon) = m$ et $\mu^*(m, w \cdot x) = \mu(\mu^*(m, w), x)$ pour tout état $m \in M$, tout mot $w \in (E \cup P)^*$ et toute position $x \in (E \cup P)$).*

Le résultat suivant qui porte sur les jeux finis, fondamental, est obtenu par McNaughton [14]

Théorème 1.3.3
Dans tout jeu fini à deux joueur l'un des deux joueurs admet une stratégie gagnante à mémoire finie.

Autrement, dit, même en restreignant l'ensemble des stratégies admissibles aux stratégies à mémoires finies, les jeux finis restent déterminés.

Remarque 1.3.4 Une arène de jeux distribué est un cas particulier d'arène de jeux discrets à deux joueurs qui joue chacun leur tour. Les définitions standards de parties et de stratégies sont donc applicables. De plus, les composants locaux d'une arène sont, ni plus ni moins, que des arènes de jeux classiques.

Afin d'éviter toute ambiguïté dans la suite, nous parlerons de parties ou de stratégies globales qui se jouent dans le jeu distribué dans son ensemble, par opposition à des parties ou des stratégies locales qui se jouent dans les projections des jeux distribués.

La notion de stratégie distribuée (pour l'équipe des processus) est définit en n'autorisant pour l'équipe des processus des stratégies globales qui résultent de la combinaison de stratégies locales jouées par chaque processus, ces derniers n'yant connaissance (ou ne voyant) de la partie globale en court que leurs projections sur chaque arène locale.

Formellement :

Définition 1.3.5
La vue local *du processus* $i \in [1, n]$ *d'une partie global dans un jeux distribué* $G = \langle P, E, T_P, T_E, e_0, \mathcal{W} \rangle$ *est définit par une fonction*

$$view_i : \ (E \cdot P)^* \cdot E^? \to (E_i \cdot P_i)^* \cdot E_i^?$$

inductivement définit par $view_i(\epsilon) = \epsilon$, *pour toute position* $x \in P \cup E$, $view_i(x) = x[i]$, *et pour toute partie de la forme* $w.x.y$ *avec* x *et* $y \in P \cup E$, $view_i(w \cdot x \cdot y) = view_i(w \cdot x)$ *lorsque* $x[i] = y[i]$ *ou bien* $view_i(w \cdot x \cdot y) = view_i(w \cdot x) \cdot y[i]$ *autrement.*

Il vient alors :

Définition 1.3.6
Une stratégie globale σ *pour l'équipe des processus sur un jeu distribué* G *à* n *processus est dites* stratégie distribuée *lorsque, pour chaque indice* $i \in [1, n]$ *il existe une stratégie* $\sigma_i : (E[i].P[i])^+ \to E[i]$ *pour le processus* i *dans le jeux* $G[i]$, *appelée* stratégie locale *pour le processus* i, *telles que, pour toute partie globale de la forme* $w \cdot p \in (E \cdot P)^+$, *étant donné l'ensemble* $I \subseteq \{1, \ldots, n\}$ *des processus actifs dans la position* p, $\sigma(w \cdot p) = e$ *si et seulement si* $e[i] = \sigma_i(view_i(w) \cdot p[i])$ *lorsque* $i \in I$ *et* $e[i] = p[i]$ *dans le cas contraire.*

On convient dans ce cas de noter $\sigma_1 \otimes \sigma_2 \otimes \cdots \otimes \sigma_n$ *une telle stratégie distribuée.*

Résoudre un jeu distribué *c'est alors vérifier que soit l'environnement admet une stratégie globale gagnante, soit l'équipe des processus admet une stratégie* distribuée *gagnante.*

Remarque 1.3.7 Avec cette définition de résolution de jeux distribués, on constate que les jeux distribués ne sont pas déterminés. En effet, on peut facilement construire un jeu distribué à deux processus qui admet une stratéghie gagnante pour l'équipe des processus mais dont toute les stratégies gagnantes necessites de connaitre ensemble à chaque instant, les deux projections, distinctes, des positions des processus. Dans un tel jeu, aucune stratégié gagnante n'est distribuable.

1.4 Un exemple de jeu distribué

Une *architecture distribuée* telle que définit par Pnueli et Rosner [21] est un ensemble de *sites* I connectées ensemble par un ensemble C de *canaux (unidirectionnelle) de communication* reliant les site entre eux dans le cas de communications internes aux sites ou bien constituant, des entrées de l'environnement vers un site ou des sorties d'un site vers l'environnement. La bande passante de chaque canal $c \in C$ est caractérisé par un alphabet fini X_c qui donne l'ensemble des valeurs possibles que le canal peut transmettre en une étape élémentaire de communication.

Un exemple typique d'architecture distribuée est l'*architecture pipeline* dont les sites sont totalement ordonnée, chaque site prenant ses entrées de la sortie

du site qui le précède ou de l'environnement s'il est le premier, le dernier site produisant dans l'environnement une sortie.

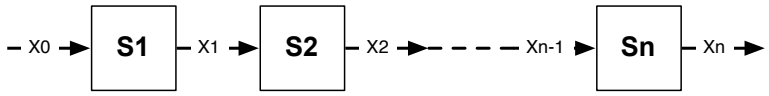

FIGURE 1 – L'architecture pipeline

Dans cette architecture, A chaque instant, chaque site $i \in [1, n]$ reçoit une valeur $x_{i-1} \in X_{i-1}$ et produit une valeur $x_i \in X_i$. Un programme pour chaque site i est donc une fonction séquentielle $f_i : X_{i-1}^* \to X_i^*$ telle que pour toute entrée $u \in X_{i-1}^*$ on a $|f_i(u)| = |u|$.

Le comportement d'une architecture ainsi programmée consiste alors, pour chaque séquence d'entrées $u \in X_0^*$, à produire la sortie $v \in X_n^*$ définie par

$$v = f_n \circ f_{n-1} \circ \cdots \circ f_1(w)$$

.

Montrons maintenant comment, pour chaque architecture pipeline $\mathbb{A}$ à n sites nous pouvons alors fabriquer une arène distribuée $G_{\mathbb{A}} = \langle P, E, T_P, T_E \rangle$ a n processus de telle sorte que les stratégies locales des processus codent les programmes de chaque sites, les parties globales joué sur le jeux distribué codant alors le comportement globale de l'architecture pipeline.

Pour faire cela, il suffit de prendre $P = X_0 \times \ldots \times X_{n-1}$; $E = X_1 \times \ldots \times X_n$, les coups locaux de chaque processus i étant définit par $T_{P,i} = X_{i-1} \times X_i$ et les coups globaux de l'environnement étant définit par $((v_1, \ldots, v_n), (v'_1, \ldots, v'_n)) \in T_E$ lorsque $v'_i = v_{i-1}$ pour tout $i \in \{1, \ldots, n\}$ and $v'_0 \in X_0$.

On vérifie que, dans cette arène, à chaque coup, chaque processus lit la valeur d'entrée qui lui est proposée et produit (sans contrainte) la sortie qu'il souhaite. Autrement dit, les stratégies locales des processus sont bien des codage des fonctions séquentielles.

Remarquons aussi que les coups de l'environnement sont restreins de tels sorte que, s'il peut effectivement choisir à chaque coup une valeur d'entrée au système (à choisir dans X_0), il est par ailleurs forcé de transmettre chaque valeur produite par le processus i avec $i < n$ au processus qui le suit, numéroté $i + 1$.

La spécification du comportement attendu peut alors être exprimée, comme dans l'approche de Kupferman et Vardi, à l'aide d'une formule de logique monadique du second ordre (ou de façon équilante un automate d'arbres [22]) définissant les (arbres de) stratégies gagnantes pour l'équippe des processus.

Ainsi, résoudre le problème de la synthèse de programmes distribués sur l'architecture pipeline reviendra bien à produire une stratégie distribuée pour l'équipe des processus dont (l'arbre de) la stratégie globale induite satisfaira cette spécification.

2 Jeux distribués linéaires et automates d'arbres

Nous présentons ici les connections étroites qui existent entre la théorie maintenant classiques des automates d'arbres et la résolutions des jeux distribués dans le cas des architectures dites linéaires. Cette partie reprend pour l'essentielle la communication des deux auteurs présentée en 2005 [2].

2.1 Arbres et automates

On présente ici une variation de la notion d'arbre et d'automates d'arbres associés qui se prettent particulièrement bien à l'analyse et au traitement des jeux distribués.

Définition 2.1.1 (Arbre)
Etant donné deux alphabets finis D et Σ, un (D, Σ)-arbre *est une fonction partielle $D^* \to \Sigma$ dont le domaine, clos par prefixe, contient au moins le mot vide ϵ. Dans la suite, les éléments de Σ sont appelés* étiquettes *et les éléments de D sont appelés* directions*.*

La définition d'automate suivante est une variation sur la definition originale de Muller et Schupp des automates alternants [17]. Avec cette nouvelle syntaxe, notre objectif est d'obtenir une définition d'automates alternants qui se comportent comme des transducteurs d'arbres. Les calculs d'un automates alternant sur un arbres étant eux-même des arbres, ils pourront être composés séquentiellement.

Définition 2.1.2 (Automate alternant et automate non déterministe)
Un automate alternant de (D, Σ)-arbre *est un tuple de la forme :*

$$\mathcal{A} = \langle Q = Q^\forall \uplus Q^\exists, D, \Sigma, q_0, \delta = \delta^\forall \cup \delta^\exists, Acc \subseteq Q^\omega \rangle$$

où Q est un ensemble fini d'états partionné en deux sous-ensembles $Q^\forall$ et $Q^\exists$ d'états respectivement appelé états universels et états existentiels, $q_0 \in Q^\exists$ est l'état initial, $\delta^\forall : Q^\forall \times D \to \mathcal{P}(Q^\exists)$ et $\delta^\exists : Q^\exists \times \Sigma \to \mathcal{P}(Q^\forall)$ sont les fonctions de transition, et $Acc \subseteq Q^\exists.(Q^\forall.Q^\exists)^\omega$, critère d'acceptance, est un langage ω-rational.

Un automate $\mathcal{A}$ est dit automate non deterministe *ou, peut être plus explicitement,* automate non alternant*, lorsque, pour tout état universel $q \in Q^\forall$, toute direction $d \in D$, on a $|\delta^\forall(q, d)| \leq 1$.*

Enfin, la taille d'un automate $\mathcal{A}$, notée $|\mathcal{A}|$, est définie[1] *comme étant la somme des cardinaux de ses ensembles d'états.*

Avec cette définition, les calculs d'automates sur les arbres sont eux-même des arbres. Plus précisément :

1. de façon plutôt sommaire, notre objet n'est pas de faire une étude précise de la complexité des outils et concepts proposés

Définition 2.1.3 (Calculs d'automates)
Un calcul *d'automate $\mathcal{A} = \langle Q, D, \Sigma, i, \delta, Acc\rangle$ sur un (D, Σ)-arbre t : $D^* \to \Sigma$ est un arbre étiqueté sur $Q^\forall$ et dont les directions sont $D \times Q^\exists$, c'est à dire un arbre ρ de la forme ρ : $(D \times Q^\exists)^* \to Q^\forall$ tel que :*

- *$\rho(\epsilon) \in \delta^\exists(q_0, t(\epsilon))$,*
- *pour tout $w \in Dom(\rho)$, si $\rho(w) = q$, alors, pour toute direction $d \in D$ telle que $w[D].d \in dom(t)$, en notant $a = t(w[D].d)$, pour tout état existentiel $q_1 \in \delta^\forall(q, d)$, il existe un état universel $q_2 \in \delta^\exists(q_1, a)$ tel que $\rho(w.(d, q_1)) = q_2$.*

Un tel calcul ρ est appelé calcul acceptant *lorsque pour toute branche infinie w du calcul ρ on a $states_\rho(w) \in Acc$, où $states_\rho(w) \in (Q^\forall.Q^\exists)^\omega$ désigne la séquence infinie d'états (universel et existentiel) rencontrés le long de w.*

On dit alors qu'un arbre t est accepté par par un automate $\mathcal{A}$ lorsqu'il existe un calcul acceptant de l'automate $\mathcal{A}$ sur l'arbre t. On note alors $L(\mathcal{A})$ l'ensemble de tous les arbres acceptés par l'automates $\mathcal{A}$. C'est le langage reconnu par l'automate $\mathcal{A}$.

Remarque 2.1.4 On peut montrer que ces automates (alternants ou non alternants), bien que définit de façon inhabituelle,ont le même pouvoir d'expression que leurs homologues plus classique tel que définit dans les travaux de Muller et Schupp [17].

En particulier :

Théorème 2.1.5
Pour tout automate alternant $\mathcal{A}$ il existe un automate non alternant $\mathcal{A}'$ tel que $L[\mathcal{A}') = L(\mathcal{A})$ avec $|\mathcal{A}'| \leq 2^{2^{|\mathcal{A}|}}$.

L'intérêt de cette définition alternative réside dans la notion suivante de composition d'automates.

Définition 2.1.6 (Composition séquentiel d'automates)
Etant donné deux automates d'arbres $\mathcal{A}_1 = \langle Q_1, D_1, \Sigma_1, q_{0,1}, \delta_1, Acc_1\rangle$ et $\mathcal{A}_2 = \langle Q_2, D_2, \Sigma_2, q_{0,2}, \delta_2, Acc_2\rangle$ tels que l'automate $\mathcal{A}_2$ soit non alternant avec $D_2 = D_1 \times Q_1^\exists$ et $\Sigma_2 = Q_1^\forall$, on définit la composition séquentiel de l'automate $\mathcal{A}_1$ suivi de l'automate $\mathcal{A}_2$ comme étant l'automate sur les arbres (D_1, Σ_1)-arbres

$$\mathcal{A}_2 \circ \mathcal{A}_1 = \langle \widetilde{Q}, D_1, \Sigma_1, \widetilde{q_0}, \widetilde{\delta}, \widetilde{Acc}\rangle$$

avec :

- $\widetilde{Q^\exists} = Q_1^\exists \times Q_2^\exists$; $\widetilde{Q^\forall} = Q_1^\forall \times Q_2^\forall$;
- $q_0 = (q_{0,1}, q_{0,2})$,
- $(q_1', q_2') \in \widetilde{\delta^\forall}((q_1, q_2), d) \Leftrightarrow \left\{ \begin{array}{l} q_1' \in \delta^\forall(q_1, d) \\ \{q_2'\} = \delta_2^\forall(q_2, (d, q_1')) \end{array} \right.$
- $(q_1', q_2') \in \widetilde{\delta^\exists}((q_1, q_2), a) \Leftrightarrow \left\{ \begin{array}{l} q_1' \in \delta^\exists(q_1, a) \\ q_2' \in \delta_2^\exists(q_2, q_1') \end{array} \right.$
- $\widetilde{Acc} = \{w \in \widetilde{Q}^\omega \mid w[1] \in Acc_1 \wedge w[2] \in Acc_2\}$

Il vient :

Théorème 2.1.7
Pour tout arbre $t : D_1^* \to \Sigma_1$, on a $t \in L(\mathcal{A}_2 \circ \mathcal{A}_1)$ si et seulement si il existe un calcul acceptant $\rho : (D_1 \times Q_1^{\exists})^* \to Q_1^{\forall}$ de l'automate $\mathcal{A}_1$ sur l'arbre t tel que $\rho \in L(\mathcal{A}_2)$.

Proof. La preuve, bien que laborieuse, ne pose pas de difficulté conceptuelle particulière. Elle est donc omise ici. On peut en trouver une version détaillée dans la thèse du premier auteur [1].

Retenons cependant que, dans cette preuve, il est crucial que l'automate $\mathcal{A}_2$ soit non alternant. Le théorème de simulation de Muller et Schupp (Théorème 2.1.5) nous assure cependant que c'est toujours possible, quoiqu'avec un nombre d'état exponentiellement plus grand. □

2.2 Jeux avec condition de gains externes

Notre objectif est de mettre en évidence et d'exploiter les rapports étroits qu'ils peut exister entre les automates d'arbres et les jeux distribués. Cette objectif passe tout d'abord par la définition de jeux distribués dont les conditions de gains serait exprimé de façon externe au jeu distribué à l'aide d'un automate qui définie directement le langage des stratégies de l'équipe des processus qui seront gagnante. On définit donc ici ce qu'on appelera des conditions de gains externes.

Définition 2.2.1 (Condition de gain externe)
Un jeu avec condition de gain externe *est un tuple*

$$G = \langle P, E, T_P, T_E, e_0, \mathcal{A} \rangle$$

où $\langle P, E, T_P, T_E \rangle$ est une arène de jeux, $e_0 \in E$ est une position initiale dans cette arène, et $\mathcal{A}$ est un automate de (P, E)-arbre. Dans un tel jeu, on dira que le joueur processus admet une stratégie gagnante lorsqu'il admet une stratégie σ dont l'arbre $t_\sigma(e_0)$ induit par les parties issues de la position e_0 et compatibles avec la stratégie σ appartient au langage $L(\mathcal{A})$ des arbres acceptés par l'automate $\mathcal{A}$.

Cette définition s'étend alors sans difficulté aux arènes distribuées, définissant ainsi une notion de jeux distribués avec condition externe.

Dans la suite, afin d'éviter toute confusion, un jeu avec condition de gain comme définie dans la section 1.3 sera appelé jeu avec condition de gain interne*.*

Nous allons maintenant démontrer que les jeux avec conditions de gains externes sont, pour l'essentiel, équivalent aux jeux avec condition interne, les conditions de gains externes pouvant être internalisées au prix de l'ajout d'un joueur processus.

Théorème 2.2.2 (Internalisation)
Pour tout jeux distribué G à n processus avec condition de gain externe $\mathcal{A}$ il existe un jeu distribué G' à $n+1$-processus avec condition de gain interne avec $G'[1,\ldots,n] = G$ et tel que l'équipe des processus admet une stratégie gagnante σ dans le jeux G si et seulement l'équipe des processus admet une stratégie gagnante de la forme $\sigma \otimes \sigma'$ dans le jeu G'.

En particulier, en se restreignant aux stratégies distribuées, l'équipe des processus admet une stratégie distribuées gagnante dans le jeux G si et seulement elle admet une stratégie distribuée gagnante dans le jeux G'.

Proof. La structure de l'énoncé nous permet de nous restreindre, sans perte de généralité, à un jeux G à deux joueurs (et donc un seul processus). Soit donc $G = \langle P, E, T_P, T_E, e_0, \mathcal{A}\rangle$ un tel jeux munie d'une condition de gain externe $\mathcal{A} = \langle Q^\forall \uplus Q^\exists, P, E, q_0, \delta = \delta^\forall \cup \delta^\exists, Acc\rangle$.

Nous définissons le jeu distribué à deux processus $G' = \langle P', E', T'_P, T'_E, e'_0, \mathcal{W}\rangle$ de la façon suivante. Les positions et les conditions de gains sont définit par :

- $P' = (E \times (Q^\exists \times E)) \cup (P \times (Q^\exists \times \{\#\}))$,
- $E' = (E \times Q^\exists) \cup (E \times Q^\forall)$,
- $e'_0 = (e_0, q_0)$,
- $\mathcal{W} = \{w \in (E'.P')^\omega \mid \pi_{Q^\forall \cup Q^\exists}(w) \in Acc\}$

et les coups sont définit (par séquences de quatre coups successifs) de la façon suivante :

1. d'une position de la forme $(e, q) \in E \times Q^\exists$ (ou bien la position initiale) : l'environnement joue (de façon deterministe) dans la position de processus $(e, (q, e)) \in E \times (Q^\exists \times E)$,
2. d'une position de la forme $(e, (q, e)) \in E \times (Q^\exists \times E)$: le processus 2 (codant l'automate) choisit localement un état universel $q' \in \delta^\exists(q, e)$ et l'autre processus reste inactif, atteignant ainsi la position d'environnement $(e, q') \in E \times Q^\forall$,
3. d'une position de la forme $(e, q') \in E \times Q^\forall$: l'environnement choisit une position $p \in T_E(e)$ et un état existentiel $q_1 \in \delta^\forall(q', p)$, atteignant ainsi la position de processus $(p, (q_1, \#)) \in P \times Q^\exists$,
4. d'une position de la forme $(p, (q_1, \#)) \in P \times Q^\exists$: le processus 1 (ou l'équipe de processus) provenant du jeu G, choisit une position d'environnement $e_1 \in T_P(p)$, le processus 2 (sur l'automate $\mathcal{A}$) reste quasiment inactif en ne faisant qu'effacer le signe $\#$, atteignant ainsi la position d'environnement $(e_1, q_1) \in E \times Q^\exists$,

ce qui couvrent bien l'ensemble des cas possibles.

Avec cette construction, dès lors qu'il existe un calcul acceptant ρ de l'automate $\mathcal{A}$ sur l'arbre $t_\sigma(e_0)$ induit par une stratégie de processus σ in G, le joueur déduit facilement de ce calcul ρ une strategy locale σ' tel que la stratégie distribuée $\sigma \otimes \sigma'$ est gagnante dans le jeu G'.

A l'inverse, étant donné une stratégie distribuée $\sigma \otimes \sigma'$ sur le jeu G', on fabrique facilement à partir de la stratégie locale σ' du processus 2 dans l'arène de G' un calcul acceptant de l'automate $\mathcal{A}$ sur (l'arbre de) la stratégie σ.

Fondamentalement, cette définition de G' ne fait que donner comme rôle au processus 2 la construction d'un calcul (n'importe lequel) de l'automate $\mathcal{A}$ sur toute stratégie σ du processus 1. Les conditions de gain internalisées dans G', qui miment les conditions d'acceptation de l'automate $\mathcal{A}$, assurent alors que la stratégie résultante dans le jeux G' est gagnante si et seulement si la stratégie initiale dans G et le calcul de $\mathcal{A}$ sur σ ainsi définies sont respectivement gagnante et acceptant. □

Remarque 2.2.3 Lorsque le jeux G est un jeu simple (à deux joueurs) avec condition de gain externe, cette internalisation de la condition de gain que être poussée plus loin (en construisant le produit de l'automate $\mathcal{A}$ par le jeu G) de telle sorte que le jeu G' résultant est un jeu simple (à deux joueurs) qui peut donc être résolu. Remarquons au passage que lorsque Acc est une condition ω-rationelle, la condition de gain $\mathcal{W}$ construite pour G' est bien aussi ω-rationelle.

2.3 Théorème d'externalisation

Dans le jeu pipeline évoqué dans la première partie, on constate que, à chaque instant, lorsque toutes les stratégies locales des processus sont connus, le processus d'indice i est capable de prédire la position courante sur toute les arènes d'indice inférieur. En effet il connait l'entrée fourni à la sous architecture composée de ces processus puisqu'il la fournit lui-même.

Cette observation suggère que, dès lors qu'un processus dans un jeu distribué admet, explicitement ou par inférence, une certaine connaissance du reste du jeu, il peut se comporter comme un automate d'arbre qui peut reconstruire le comportement globale du système. Plus précisément, en disposant en entrée des stratégies locales des processus, il peut seul, simuler le comportement du système global et, en composant cette simulation avec la condition de gain, vérifier localement que c'est bien une stratégie gagnante qui est produite par les processus.

Autrement, sous certaine condition de connaissance par un processus du reste du jeu distribué, il devrait exister une construction inverse à l'internalisation d'une condition de gain externe nous permettant, tout au contraire, d'externaliser voire même de combiner avec une condition externe déjà existante, le jeu local associé à ce processus. Ce faisant, le nombre d'arène locale misent en oeuvre pourrait être réduit et, nous pourrions, inductivement, résoudre le jeux distribué. C'est ce que ne nous proposons de démontrer dans cette partie.

La notion de leader définit ci-dessous formalise la notion intuitive de connaissance par un processus de l'ensemble du jeu. A nouveau, la quasi associativité de la notion de jeux distribués à plusieurs processus nous permet de nous restreindre au cas des jeux à deux processus.

Définition 2.3.1 (Leader)
Etant donné, un jeu distribué à deux processus $G = \langle P, E, T_P, T_E, e_0, \mathcal{A} \rangle$*, le processus* 2 *est appelé* processus leader*ălorsque, pour toute position (accessible)*

$e \in E$ de l'environnement, toute position x et $y \in P$ des processus immédiatement accessibles à partir de e, i.e. avec $(e, x) \in T_E$ et $(e, y) \in T_E$,

- *si $x[2] = y[2]$ alors $x[1] = y[1]$,*
- *si $x[2] \in E[2]$ ou $y[2] \in E[2]$ alors $x = y$.*

Remarque 2.3.2 Intuitivement, le processus 2 est une leader lorsque, dès lors qu'il connait une position d'environnement globale $e \in E$, quelque soit le coup joué par l'environnement à partir de cette position (ou bien après quelques coups jusqu'à son activation si le processus 2 est inactif dans la position e) le processus 2 peut prédire, à la lecture de sa seule position locale, la position globale atteinte.

Cette propriété locale , qui porte sur chaque coup de l'environnement, s'étend à une notion de connaissance globale de la façon suivante :

Lemme 2.3.3 Soit $G = \langle P, E, T_P, T_E, e_0 \rangle$ un jeux distribué à deux processus dans lequel le processus 2 est leader. Pour toute stratégie σ, la restriction de $view_2$ aux parties finies issue de e_0, terminant dans une position où le processus 2 est actif et consistante avec la stratégie σ est injective.

Reformulé de façon plus effective, cette observation nous conduit au résultat suivant :

Lemme 2.3.4 Pour tout jeu distribué à deux processus $G = \langle P, E, T_P, T_E, e_0, \mathcal{W} \rangle$ dans lequel le processus 2 est leader, il existe un automate $\mathcal{A}_2$ de $(E[1], P[1])$-arbres tel que, pour toutes stratégies du ou des processus σ sur G et σ_1 sur G_1, les propriétés suivantes sont équivalentes :

(1) il existe une stratégie σ_2 sur $G[2]$ telle que $\sigma = \sigma_1 \otimes \sigma_2$

(2) il existe un calcul acceptant ρ de l'automate $\mathcal{A}_2$ sur l'arbre t_{σ_1} avec, de surcroit, $\rho = t_{\sigma_2}$.

Proof. Nous donnons tout d'abord une construction de l'automate $\mathcal{A}_2$ dans le cas ou les deux processus $\mathcal{A}_1$ et $\mathcal{A}_2$ restent actifs.

L'automate $\mathcal{A}_2 = \langle Q_2, P[1], E[1], q_{0,2}, \delta_2, Acc_2 \rangle$ est définit de la façon suivante :

- $Q_2^\forall = E$; $Q_2^\exists = P[2] \cup \{q_{0,2}\}$,
- $\delta_2^\forall(q, p_1) = \{p_2 \in Q_2^\exists : \ (q, (p_1, p_2)) \in T_E\} \quad (q \in Q_2^\forall, p_1 \in P[1])$,
- $\delta_2^\exists(p_2, e_1) = \{q \in Q_2^\forall : \ q[1] = e_1 \wedge (p_2, q[2]) \in T_P[2]\} \quad (p_2 \in Q_2^\exists, e_1 \in E[1])$ avec $\delta_2^\exists(q_{0,2}, e_1) = \{e_0[2]\}$,
- $Acc_2 = Q_2^\omega$.

La correspondance énoncée ci-dessus entre les calculs de l'automates $\mathcal{A}_2$ sur les arbres de stratégie du processus 1 dans le jeu local $G[1]$ et les stratégies de l'équipe des processus dans le jeu G est une conséquence simple de notre définition et du fait que le processus 2 est un leader dans G.

La preuve ne présente aucune difficulté puisqu'il ne s'agit que de *vérifier* que pour toute stratégie du processus 1 sur la composante $G[1]$, les calculs acceptant ρ de $\mathcal{A}_2$ sur t_{σ_1} sont tout simplement les arbres des stratégies σ_2 du processus

2 sur la composante $G[2]$ telle que la stratégie distribué résultante $\sigma_1 \otimes \sigma_2$ soit gagnante dans G.

Dans le cas ou le processus 2 peut être inactivé par l'environnement, on peut vérifier, sous l'hypothèse que le joueur 2 est un leader, que le jeu peut être normaliser afin de se ramener au cas précédent.

Dans le cas ou le processus 1 peut être inactivé par l'environnement, la construction proposé peut être étendue à l'aide d'une notion de ϵ-transition (simple mais lourde à manipuler). La construction reste cependant essentiellement la même. □

Remarque 2.3.5 Le cas où les processus peuvent être inactivé peut aussi être traité par réduction à un jeu synchrone comme évoqué à la de la partie suivante.

Puisque le résultat précédent est vrai pour toute arène, même dans le cas ou $G[1]$ est une arène distribuée, il vient :

Théorème 2.3.6 (Externalisation)

Pour tout jeu distribué à $n+1$ processus de la forme $G = \langle P, E, T_P, T_E, e_0, \mathcal{A}\rangle$ avec condition de gain externe $\mathcal{A}$ et tel que le processus $n+1$ est leader G il existe un automate de $(E[1\cdots n], P[1\cdots n])$-arbre $\mathcal{A}_{n+1}$ tel que les propositions suivantes sont équivalentes :

(1) *les processus ont une stratégie distribuée gagnante dans le jeux G,*

(2) *les processus on une stratégie distribuée gagnante dans le jeux $\langle G[1\ldots n-1], e_0[1\ldots n-1], \mathcal{A}\circ\mathcal{A}_n\rangle$.*

2.4 Application aux jeux linéaires

On s'interesse ici au cas des architecture linaire dans lesquels, inductivement, chaque processus d'indice i est leader sur le jeu distribué $G[1,i]$ constitué des arènes d'indice inférieure ou égale. Appliqué de proche en proche, le théorème d'externalisation présenté ci-dessus nous permet de réduire un jeu linéaire à un jeu à deux joueur, démontrant ainsi les jeux linéaire sont déterminé et que l'existence de stratégie distribuée gagnante pour les processus est décidable.

Définition 2.4.1
Etant donné un jeu distribué à n processus de la forme $G = \langle P, E, T_P, T_E, e_0, \mathcal{W}\rangle$ on dit que le jeux G est un jeu distribué linéaire *lorsque, pour tout indice $i \in [1,n]$, pour toute position de l'environnement e et f, pour toute position des processus p and $q \in P$ telles que $(e,p) \in T_E$ et $(f,q) \in T_E$:*

Si $e[1,i] = f[1,i]$ et $p[i] = q[i] \in P[i]$ ou $p[i] \in E[i]$ ou $q[i] \in E[i]$ alors $p[1,i] = q[1,i]$.

A nouveau, cette propriété (local) de linéarité assure que, avant tout coup de l'environnement, si le processus i connait (au sens epistemique) non seulement sa propre position $e[i]$ mais aussi les positions $e[1, i-1]$ de tout les processus d'indice plus petit, alors cette propriété reste vraie après tout coup de l'environnement.

De plus, puisque les processus peuvent connaitre à l'avance les stratégies jouées par chacun d'eux, cette propriété locale garantie aussi que, globalement, à partir de la position initiale e_0 connue de tous les processus, étant fixé une stratégie distribuée $\sigma = \sigma_1 \otimes \cdots \otimes \sigma_n$, à tout instant de toute partie compatible avec la stratégie distribuée σ chaque processus actif connait (au sens epistémique) les positions de tous les processus d'indice inférieur.

Il vient :

Théorème 2.4.2
Les jeux distribuées linéaires sont décidables. De plus, si l'équipe des processus admet une stratégie distribué gagnante alors elle admet une stratégie distribuée gagnante finie.

Proof. Soit $n > 0$ le nombre de processus, et soit $G = \langle P, E, T_E, T_P, e_0, \mathcal{A} \rangle$ un jeu distribué linéaire à $n+1$ processus avec une condition externe $\mathcal{A}$ présentée sous la forme d'un automate fini non alternant. Le processus $n+1$ étant leader, par application du Théorème 2.3.6, on fabrique un jeu équivalent à n processus, avec condition de gain de la forme $\mathcal{A} \circ \mathcal{A}_n$ dans lequel le processus n est maintenant leader. On peut donc, après application du théorème de simulation sur l'automate $\mathcal{A} \circ \mathcal{A}_n$ pour obtenir un automate non alternant, réitérer le processus jusqu'à obtenir un jeu à un processus avec condition externe.

Ce dernier jeu peut alors être résolu par des techniques classiques. On peut par exemple se ramener à un jeu à deux joueurs est conclure en appliquant le Théorème 1.3.3. □

Remarque 2.4.3 A chaque itération, l'application du théorème de simulation à l'automate $\mathcal{A} \circ \mathcal{A}_n$ produit une condition externe de taille exponentiellement plus grande. La résolution d'un jeu distribué linéaire est donc d'une très mauvaise complexité. ce n'est pas surprenant puisqu'on sait que ce problème est non élémentaire [21].

Corollaire 2.4.4 On vérifie facilement que les jeux induit par les architectures pipeline décrient dans la section 1.4 son linéaires. On en conclue que le problème de la synthèse distribuée sur architectures pipelines est décidable.

3 Synchronisation des jeux distribués

Dans notre définition des jeux distribués, la possibilité est parfois offerte à l'environnement d'inactiver un processus désynchronisant ainsi l'ensemble des processus. La lecture attentive de la définition des stratégies distribuées montre

par ailleurs que les processus ne s'aperçoivent pas même pas, si l'on peut dire, de cette désynchronisation.

Les jeux résultants, qu'on pourrait qualifier d'asynchrones, sont bien adaptés à la modélisation. Par exemple, on peut vouloir modéliser la situation où deux processus communiquent au travers d'un canal de communication d'une part, et disposent d'autre part chacun d'une mémoire interne pour stocker des résultats de calcul. Il peut s'avérer pratique de mettre un processus en attente le temps qu'une communication s'effectue. Dans d'autres cas, on peut désirer modéliser des systèmes vraiment asynchrones. Dans ce cas, les processus ne doivent pas disposer d'une horloge globale ; la possibilité pour l'environnement de laisser un processus inactif permet de masquer le tic d'horloge implicite fourni par les coups successifs de l'environnement.

La présence de cet asynchronisme augmente t'il pour autant le pouvoir d'expression des jeux distribuées ? Dans cette partie, nous répondons par la négative à cette question.

Plus précisément, nous démontrons que, dès lors qu'on se restreint à des stratégies à mémoire finie, tout jeu asynchrone peut être réduit en un jeu synchrone (de même taille) qui admet, essentiellement, les mêmes stratégies distribuées gagnantes. Ce résultat constitue l'essentiel d'une communication faite en 2008 [4]. Il sont par ailleurs présenté en détails dans la thèse du premier auteur [1].

3.1 Jeux synchrones et synchronisation de jeux asynchrone

Un jeu distribué est appelé synchrone lorsque l'environnement n'a pas la possibilité d'inactiver un processus. Formellement :

Définition 3.1.1
Un jeu distribué de n processus $G = \langle P, E, T_E, T_P, e_0, \mathcal{W}\rangle$ est dit synchrone *lorsque $T_E \subseteq E \times \prod_{i\in[1,n]} P[i]$. Dans le cas contraire, il est dit* asynchrone.

Remarque 3.1.2 Dans le cas d'un jeu synchrone, l'ensemble des positions de processus accessibles depuis la position initial est de la forme $\prod_{i\in[1,n]} P[i]$. Plus encore, pour chaque partie globale $w \in (E.P)^*.E^?$, on a $view_i(w) = w[i]$, i.e. la vue local d'une partie globale n'est que sa projection locale. En effet, dans chaque position de processus, tout les processus sont actifs. Dès lors, étant données n stratégies locales $\{\sigma_i\}_{i\in[1,n]}$, la stratégie globale induite $\sigma_1 \otimes \cdots \otimes \sigma_n$ est définie pour toute partie globale w terminant dans une position des processus par :

$$\sigma_1 \otimes \cdots \otimes \sigma_n(w) = (\sigma_1(w[1]), \ldots, \sigma_n(w[n]))$$

On le voit, dans le cas des jeux synchrones, les définitions se simplifient.

Dans le cas d'un jeux asynchrone ce n'est pas vrai. La vue locale d'un processus désactivé ne change pas alors même que la partie globale progresse. On modélise ainsi le fait que le processus inactivé n'a pas même conscience si l'on peut dire de son inactivation.

On va maintenant s'attacher à montrer que tout jeu distribué peut se réduire à un jeu synchrone. On pourra ainsi bénéficier à la fois de la souplesse de modélisation fournie par les jeux asynchrones ainsi que de la facilité de manipulation des jeux synchrones.

Théorème 3.1.3
Pour tout jeu G, on peut construire un jeu synchrone $\widetilde{G}$ de même taille que G tel que les processus ont une stratégie distribuée à mémoire finie gagnante dans G si et seulement si ils en ont une dans $\widetilde{G}$.

Le reste de cette section est consacré à la preuve de ce résultat.

Prenons un jeu distribué à n Processus $G = \langle P, E, T, e_0, \mathcal{W}\rangle$. On va détailler la construction d'un jeu synchrone $\widetilde{G}$, équivalent en termes d'existence de stratégies distribuées gagnantes.

Tout d'abord, pour chaque ensemble E_i, on se munit d'un ensemble équipotent $\widehat{E_i}$ tel que $E_i \cap \widehat{E_i} = \emptyset$; pour tout élément e de E_i, on note $\widehat{e}$ son image dans $\widehat{E_i}$. Soit $\widehat{E} = \prod_{i\in\{1,\dots,n\}} \widehat{E_i}$.

On considère le jeu synchrone $\widetilde{G} = \langle \widetilde{P}, \widetilde{E}, \widetilde{T}, e_0, \widetilde{\mathcal{W}}\rangle$ dont les positions sont :

$$\widetilde{P_i} = P_i \cup \widehat{E_i} \quad ; \quad \widetilde{E_i} = E_i \quad (\text{pour tout } i \in \{1, \dots n\})$$

Pour toute position x dans $P \cup E$, on note $\widehat{x}$ la position de $\widetilde{P}$ obtenue en remplaçant dans e tous les composants appartenant à E_i par leur image dans $\widehat{E_i}$. Précisément, cela donne :

$$\widehat{x}[i] = \begin{cases} \widehat{x[i]} & \text{si } x[i] \in E_i \\ x[i] & \text{si } x[i] \in P_i \end{cases}$$

La fonction qui assigne à tout élément x de $P \cup E$ son image $\widehat{x}$ dans $\widetilde{P}$ est trivialement une bijection. Les coups de $\widetilde{G}$ sont définis comme suit :

$$\begin{aligned} \widetilde{T_i^P} &= T_i^P \cup \{(\widehat{e}, e) \mid e \in E_i\} \\ \widetilde{T^E} &= \{(e, \widehat{p}) \in \widetilde{E} \times \widetilde{P} \mid (e, p) \in T^E\} \\ &\quad \cup \{(e, \widehat{e}) \mid e \in E\} \end{aligned}$$

On définit inductivement une fonction d'effacement des coups asynchrones $\mathsf{cancel} : (\widetilde{E}\cdot\widetilde{P})^* \to (E\cdot P)^*$ comme suit : $\mathsf{cancel}(\epsilon) = \epsilon$, $\mathsf{cancel}(w\cdot e\cdot\widetilde{p}) = \mathsf{cancel}(w)\cdot e\cdot p$, et $\mathsf{cancel}(w\cdot e\cdot\widehat{e}) = \mathsf{cancel}(w)$ ((avec $p \in P$, $e \in E$). On étend cette fonction aux mots infinis en écrivant : $\mathsf{cancel}(x_0 \cdot x_1 \cdots) = \lim_{i\to\infty} \mathsf{cancel}(x_0 \cdots x_i)$ (il s'agit bien d'une suite convergente au sens de la topologie préfixe sur les mots).

La condition de gain de $\widetilde{G}$ est alors définie ainsi :

$$\widetilde{\mathcal{W}} = \mathsf{cancel}^{-1}(\mathcal{W}) \cup (\widetilde{E}\cdot\widetilde{P})^* \cdot (E\cdot\widehat{E})^\omega$$

Remarque 3.1.4 Le graphe de l'arène de G se plonge dans celui de l'arène de $\widetilde{G}$. Le graphe de l'arène de $\widetilde{G}$ n'est en l'occurrence rien de plus que le sous-graphe induit par ce plongement, auquel on rajoute une boucle sur chaque position de l'environnement, correspondant à un coup totalement asynchrone. D'autre part, la condition de gain de $\widetilde{G}$ n'est pas beaucoup plus compliquée que celle de G : parmi les conditions de gain usuelles (cf. section 1.2), seule la condition d'accessibilité n'est pas conservée par cette construction.

Commençons par montrer qu'on peut "recopier" toute stratégie distribuée gagnante dans G sur $\widetilde{G}$; il suffit pour cela de s'assurer que les stratégies locales ne tiennent pas compte des coups asynchrones qui sont joués sur leur arène.

Lemme 3.1.5 Pour toute stratégie distribuée gagnante σ sur G, il existe une stratégie distribuée $\widetilde{\sigma}$ sur $\widetilde{G}$ gagnante pour les Processus.

Proof. Tout d'abord, pour tout $i \in \{1, \ldots, n\}$, on définit une fonction d'effacement des coups localement asynchrones $\mathsf{cancel}_i : (\widetilde{E_i} \cdot \widetilde{P_i})^* \to (E_i \cdot P_i)^*$ de la façon suivante : $\mathsf{cancel}_i(\epsilon) = \epsilon$, et pour tout mot $w \in (E_i.P_i)^*$, pour toutes positions $e \in E_i$, $p \in P_i$, $\mathsf{cancel}_i(w \cdot e \cdot p) = \mathsf{cancel}_i(w) \cdot e \cdot p$, et $\mathsf{cancel}_i(w \cdot e \cdot \widehat{e}) = \mathsf{cancel}_i(w)$.

Considérons une stratégie distribuée gagnante $\sigma = \sigma_1 \otimes \cdots \otimes \sigma_n$ sur G. On définit la stratégie distribuée $\widetilde{\sigma} = \widetilde{\sigma_1} \otimes \cdots \otimes \widetilde{\sigma_n}$ définie pour tout $i \in \{1, \ldots, n\}$, pour toute partie locale $w \in \widetilde{E_i} \cdot (\widetilde{P_i} \cdot \widetilde{E_i})^*$, pour toutes positions $p \in P_i$, $e \in E_i$ par :

$$\begin{array}{rcl} \widetilde{\sigma_i}(w \cdot p) & = & \left\{ \begin{array}{ll} \sigma_i(\mathsf{cancel}_i(w \cdot p)) & \text{si } \mathsf{cancel}_i(w \cdot p) \in Dom(\sigma_i) \\ \text{indéfinie} & \text{sinon} \end{array} \right. \\ \widetilde{\sigma_i}(w \cdot \widehat{e}) & = & e \end{array}$$

Il est clair que pour tout $i \in \{1, \ldots, n\}$ le diagramme suivant commute :

$$\begin{array}{ccc} (\widetilde{E} \cdot \widetilde{P})^* & \xrightarrow{\mathsf{cancel}} & (E \cdot P)^* \\ {\scriptstyle \pi_i}\downarrow & & \downarrow{\scriptstyle \mathsf{view}_i} \\ (\widetilde{E_i} \cdot \widetilde{P_i})^* & \xrightarrow{\mathsf{cancel}_i} & (E_i \cdot P_i)^* \end{array}$$

Par suite, pour toute partie globale $w \in Dom(\widetilde{\sigma})$, et pour tout processus $i \in \{1, \ldots, n\}$ tel que $\mathsf{last}(w)[i] \in P_i$, on a :

$$\begin{array}{rl} \widetilde{\sigma_i}(w[i]) & = \sigma_i(\mathsf{cancel}_i(w[i])) \\ & = \sigma_i(\mathsf{view}_i(\mathsf{cancel}(w))) \end{array}$$

Dès lors, pour toute partie infinie w dans $\widetilde{G}$ consistante avec $\widetilde{\sigma}$, on a deux cas de figure : soit $w \in (E.P)^*.(E.\widehat{E})^\omega$, et est gagnante ; sinon, la partie $\mathsf{cancel}(w)$ est complète dans G, et consistante avec σ ; elle appartient donc à $\mathcal{W}$. On a alors immédiatement $w \in \widetilde{\mathcal{W}}$. La stratégie $\widetilde{\sigma}$ est donc bien gagnante sur $\widetilde{G}$. □

Il reste maintenant à prouver le lemme suivant :

Lemme 3.1.6 Pour toute stratégie distribuée à mémoire finie gagnante $\widetilde{\sigma}$ sur $\widetilde{G}$, il existe une stratégie distribuée σ sur G gagnante pour les Processus.

Le problème qu'on va évidemment rencontrer est qu'une stratégie locale sur $\widetilde{G_i}$ ($i \in \{1, \ldots, n\}$) a la possibilité de compter les coups localement asynchrones, et ainsi disposer d'une information supplémentaire sur le déroulement de la partie globale par rapport à une stratégie locale sur G.

La solution va consister à mettre en évidence que ce comptage est de toute façon inutile, car à tout moment l'Environnement peut choisir de jouer un coup totalement asynchrone. Un processus k n'a alors aucun intérêt à compter les coups localement asynchrones sur son arène, ne sachant pas s'il s'agit de coups totalement asynchrones ou bien si l'Environnement joue effectivement sur une autre arène.

La technique de preuve va consister à *saturer* la mémoire de toute stratégie distribuée à mémoire finie sur $\widetilde{G}$, afin de construire une stratégie distribuée sur G qui se comporte comme le ferait la stratégie sur $\widetilde{G}$ si l'environnement jouait un grand nombre de coups totalement asynchrones à chaque fois que c'est à lui de jouer.

On commence par une variation sur le lemme de l'étoile appliqué aux mémoires de stratégies.

Lemme 3.1.7 Pour toute arène $\langle P, E, T\rangle$, pour toute stratégie σ à mémoire finie $\mathcal{M} = \langle M, m_0, {\cdot}h\rangle$, il existe un entier L tel que pour tout élément de mémoire $m \in M$ et pour tout mot $v \in (P \cup E)^+$, on ait :

$$\mu^*(m, v^L) = \mu^*(m, v^{k \cdot L}) \quad \text{pour tout entier } k > 0 \tag{1}$$

Proof. Quels que soient $m \in M$ et $v \in (P \cup E)^+$, soit $m_1 \cdot m_2 \cdots$ la suite d'états de mémoire donnée par $m_i = \mu^*(m, v^i)$. L'ensemble M étant fini, il existe deux entiers i et j tels que $i < j$ et $m_i = m_j$. Soient $x_m^v = i$ et $y_m^v = j - i$. Ces deux entiers satisfont l'équation suivante :

$$\mu^*(m, v^{x_m^v}) = \mu^*(m, v^{x_m^v + k \cdot y_m^v}) \quad \text{pour tout entier } k > 0 \tag{2}$$

De plus, si le couple (x_m^v, y_m^v) satisfait l'équation 2, alors $(x_m^v + k', y_m^v)$ et $(x_m^v, k' \cdot y)$ la satisfont aussi (quel que soit l'entier k'). On procède comme indiqué sur le schéma ci-dessous, en itérant (a) jusqu'à avoir $x_m^v = k \cdot y_m^v$, puis en itérant (b) k fois :

$$m \overbrace{\xrightarrow{v} \cdots \xrightarrow{v} m_i \xrightarrow{v}}^{x_m^v + 1 \text{ fois}} m_{i+1} \underbrace{\xrightarrow{v} \cdots \xrightarrow{v} m_j \xrightarrow{v}}_{y_m^v \text{ fois}} m_{j+1} = m_{i+1} \cdots \tag{a}$$

$$m \overbrace{\xrightarrow{v} \cdots \xrightarrow{v}}^{x_m^v \text{ fois}} m_i \underbrace{\xrightarrow{v} \cdots \xrightarrow{v} m_j = m_i \xrightarrow{v} \cdots \xrightarrow{v}}_{y_m^v + y_m^v \text{ fois}} m_{2 \cdot j} = m_j \cdots \tag{b}$$

On peut par conséquent trouver un entier L_m^v qui satisfait :

$$\mu^*(m, v^{L_m^v}) = \mu^*(m, v^{k \cdot L_m^v}) \quad \text{pour tout entier } k > 0 \tag{3}$$

Soit $\sim$ la relation binaire sur les mots de $(P \cup E)^+$ définie par $v_1 \sim v_2$ si et seulement si pour tout $m \in M$, $\mu^*(m, v_1) = \mu^*(m, v_2)$. Il s'agit trivialement d'une relation d'équivalence. La définition de L_m^v est compatible avec $\sim$: si $v_1 \sim v_2$, alors $L_m^{v_1} = L_m^{v_2}$. Par conséquent, on peut définir L_m^x pour tout $x \in (P \cup E)^+/\sim$.

Remarquons à présent que l'action de chaque mot $u \in (P \cup E)^+$ sur $\langle M, m_0, \mu, h\rangle$ est caractérisée par une application $f_u : M \to M$, définie par $f_u(m) = \mu^*(m, u)$. En particulier, si $f_u = f_v$, alors on a $u \sim v$. Par conséquent, l'ensemble quotient $(P \cup E)^+/\sim$ a au plus $2^{|M|}$ éléments.

L'entier $L_m = \mathsf{ppcm}\{L_m^x \mid x \in (P \cup E)_\sim\}$ est donc bien défini. Cet entier satisfait pour tout $v \in (P \cup E)^+$:

$$\mu^*(m, v^{L_m}) = \mu^*(m, v^{k \cdot L_m}) \quad \text{pour tout entier } k > 0 \tag{4}$$

De la même façon, on définit $L = \mathsf{ppcm}\{L_m \mid m \in M\}$, qui satisfait l'équation 1 pour tous $m \in M$ et $v \in (P \cup E)^+$, ce qui conclut la preuve. □

Dans le cas où l'on considère une mémoire de stratégie distribuée, le résultat précédent peut être encore affiné.

Lemme 3.1.8 Pour toute arène distribuée à n Processus $\langle P, E, T\rangle$, pour toute stratégie distribuée à mémoire finie σ avec pour mémoires $(\mathcal{M}_i = \langle M_i, m_{0,i}, \mu_i, h_i\rangle)_{i \in \{1,\dots,n\}}$, il existe un entier L tel que pour tout entier $i \in \{1, \dots, n\}$, pour tout élément de mémoire $m \in M_i$ et pour tout mot $v \in (P_i \cup E_i)^+$, on ait :

$$\mu_i^*(m, v^L) = \mu_i^*(m, v^{k \cdot L}) \quad \text{pour tout entier } k > 0 \tag{5}$$

Proof. Pour tout entier $i \in \{1, \dots, n\}$, le lemme 3.1.7 nous permet de dire qu'il existe un entier L_i satisfaisant :

$$\mu^*(m, v^{L_i}) = \mu^*(m, v^{k \cdot L_i}) \quad \text{pour tout entier } k > 0 \tag{6}$$

L'entier $L = \mathsf{ppcm}\{L_i \mid i \in \{1, \dots, n\}\}$ satisfait alors l'équation 5. □

Supposons qu'on dispose d'une stratégie distribuée à mémoire finie $\widetilde{\sigma} = \widetilde{\sigma_1} \otimes \cdots \otimes \widetilde{\sigma_n}$ gagnante sur $\widetilde{G}$, avec les mémoires locales suivantes :

$$\widetilde{\mathcal{M}_i} = \langle \widetilde{M_i}\ ,\ \widetilde{m_{0,i}} \in \widetilde{M_i}\ ,\ \widetilde{\mu_i} \colon \widetilde{M_i} \times (\widetilde{P_i} \cup \widetilde{E_i}) \to \widetilde{M_i}\ ,\ \widetilde{h_i} \colon \widetilde{M_i} \times \widetilde{P_i} \to \widetilde{E_i}\rangle$$

(pour $(i \in \{1, \dots, n\})$).

On peut appliquer le résultat du lemme 3.1.8 pour déduire la propriété suivante : il existe un entier L tel que pour tout Processus $i \in \{1, \dots, n\}$, pour tout élément de mémoire m de $\widetilde{M_i}$, pour toute position $e \in \widetilde{E_i}$, on ait :

$$\mu_i^*(m, (e \cdot \widehat{e})^L) = \mu_i^*(m, (e \cdot \widehat{e})^{k \cdot L}) \quad \text{pour tout } k \in \mathbb{N} \tag{7}$$

On considère la stratégie distribuée à mémoire finie sur G : $\sigma = \sigma_1 \otimes \cdots \otimes \sigma_n$, avec les mémoires locales $\mathcal{M}_i = \langle M_i, m_{0,i}, \mu_i, h_i \rangle$, où $M_i = \widetilde{M_i}, m_{0,i} = \widetilde{m_{0,i}}, h_i = \widetilde{h_i}$, et :

$$\mu_i(m, e) = \widetilde{\mu_i}^*(m, e \cdot (\widehat{e} \cdot e)^{2 \cdot L - 1})$$
$$\mu_i(m, p) = \widetilde{\mu_i}(m, p)$$

On va montrer que σ est gagnante sur G. Tout d'abord, considérons la fonction $\mathsf{fill} \colon (E \cdot P)^* \to (\widetilde{E} \cdot \widetilde{P})^*$ définie par :

$$\mathsf{fill}(e_0 \cdot p_0 \cdot e_1 \cdot p_1 \cdots p_n) = e_0 \cdot (\widehat{e_0} \cdot e_0)^{2 \cdot L - 1} \cdot \widehat{p_0} \cdot e_1 \cdot (\widehat{e_1} \cdot e_1)^{2 \cdot L - 1} \cdot \widehat{p_1} \cdots \widehat{p_n}$$

On étend fill aux mots infinis de la même façon que cancel.

Remarque 3.1.9 La fonction fill est clairement d'un fonction partielle des parties de G and les parties de $\widetilde{G}$. Il est en outre aisé de voir que $\mathsf{cancel} \circ \mathsf{fill}$ est la fonction identité, et donc que si $\mathsf{fill}(w) \in \widetilde{Acc}$, alors $w \in Acc$.

Le lemme qui suit constitue le cœur de la preuve, en ce sens qu'il démontre que σ se comporte sur G comme le ferait $\widetilde{\sigma}$ sur $\widetilde{G}$ si l'environnement faisait $2 \cdot L - 1$ coups totalement asynchrones chaque fois que c'est à lui de jouer.

Lemme 3.1.10 Pour toute partie infinie w dans G consistante avec σ, la partie $\mathsf{fill}(w)$ est consistante avec $\widetilde{\sigma}$.

Proof. Soit $w = e_0 \cdot p_0 \cdot e_1 \cdot p_1 \cdots$. On note w_i le préfixe $e_0 \cdots e_i$ de w (pour tout entier $i \geq 0$)

Il suffit alors de montrer que pour tout entier $i \geq 0$, et pour chaque processus k actif après w_i (c'est-à-dire $p_i[k] \in P_k$), on a : $\sigma_k(\mathsf{view}_k(w_i)) = \widetilde{\sigma_k}(\mathsf{fill}(w_i)[k])$.

Étant donné que σ et $\widetilde{\sigma}$ ne diffèrent que sur les fonctions de mise à jour des mémoires locale, il suffit de prouver que les mémoires locales de σ et de $\widetilde{\sigma}$ coïncident pour tout préfixe w_i et son image $\mathsf{fill}(w_i)$ dans $\widetilde{G}$; plus précisément, on veut vérifier l'égalité suivante pour tout entier $i \geq 0$, et pour tout $k \in \{1, \ldots, n\}$ tel que $p_i[k] \in P_k$:

$$\mu_k^*(m_{0,k}, \mathsf{view}_k(w_i)) = \widetilde{\mu_k}^*(m_{0,k}, \mathsf{fill}(w_i)[k]) \tag{8}$$

On procède par récurrence sur la longueur des préfixes w_i de w. Pour la base de la récurrence, pour tout k on a :

$$\begin{aligned}
\mu_k^*(m_{0,k}, \mathsf{view}_k(w_1)) &= \mu_k(m_{0,k}, e_0[k]) \\
&= \widetilde{\mu_k}^*(m_{0,k}, e_0[k] \cdot (\widehat{e_0[k]} \cdot e_0[k])^{2 \cdot L - 1}) && \text{définition de } \mu_k \\
&= \widetilde{\mu_k}^*(m_{0,k}, \mathsf{fill}(e_0)[k]) && \text{définition de } \mathsf{fill}
\end{aligned}$$

Par la suite, si i est un entier positif, supposons que l'égalité 8 est vérifiée pour tous les préfixes w_j de w_i, avec $0 \leq j < i$, et pour tout $k \in \{1, \ldots, n\}$ tel que $p_j[k] \in P_k$.

Pour tout $k \in \{1, \ldots, n\}$ tel que $p_i[k] \in P_k$, soit j le plus grand entier strictement inférieur à i tel que $p_j[k] \in P_k$ (entre p_j et p_i, l'environnement ne joue que des coups asynchrones sur l'arène du processus k). Cette situation est résumée par le schéma suivant, en posant $p = p_j[k]$ et $e = e_{j+1}[k]$:

	w_j								
				w_i					
	e_0	$\cdots$	e_j	p_j	e_{j+1}	$\cdots$	e_i	p_i	$\cdots$
1	$e_0[1]$	$\cdots$				$\cdots$			$\cdots$
$\vdots$									
k	$e_0[k]$	$\cdots$	$e_j[k]$	p	e	$e \cdots e$	e	$p_i[k]$	$\cdots$
$\vdots$									
n	$e_0[n]$	$\cdots$				$\cdots$			$\cdots$

Comme on le voit, on a $w_i[k] = w_j[k] \cdot p \cdot e^{2(i-j)-1}$. En outre, on peut décomposer $\mathsf{fill}(w_i)$ comme suit :

$$\mathsf{fill}(w_i) = \mathsf{fill}(w_j) \cdot \widehat{p_j} \cdot e_{j+1} \cdot (\widehat{e_{j+1}} \cdot e_{j+1})^{2L-1} \cdot \widehat{p_{j+1}} \cdots e_i \cdot (\widehat{e_i} \cdot e_i)^{2L-1}$$

De plus, en utilisant le fait que $p_\ell[k] = e_\ell[k] = e_i[k] = e$ (avec $j < \ell < i$), on obtient la décomposition suivante pour $\mathsf{fill}(w_i)[k]$:

$$\mathsf{fill}(w_i)[k] = \mathsf{fill}(w_j)[k] \cdot p \cdot e \cdot (\widehat{e} \cdot e)^{2L-1} \cdot \overbrace{\widehat{e} \cdot e \cdot (\widehat{e} \cdot e)^{2L-1} \cdots \widehat{e} \cdot e \cdot (\widehat{e} \cdot e)^{2L-1}}^{i-j-1 \text{ fois}}$$

On peut manipuler un peu cette expression :

$$\begin{aligned} \mathsf{fill}(w_i)[k] &= \mathsf{fill}(w_j)[k] \cdot p \cdot e \cdot (\widehat{e} \cdot e)^{2L-1} \cdot \ (\widehat{e} \cdot e \cdot (\widehat{e} \cdot e)^{2L-1})^{i-j-1} \\ &= \mathsf{fill}(w_j)[k] \cdot p \cdot e \cdot (\widehat{e} \cdot e)^{2L-1} \cdot \ (\widehat{e} \cdot e)^{2(i-j-1)L} \\ &= \mathsf{fill}(w_j)[k] \cdot p \cdot e \cdot (\widehat{e} \cdot e)^{L-1} \cdot \ (\widehat{e} \cdot e)^{(2i-2j-1)L} \end{aligned} \tag{9}$$

Soit $m = \mu_k^*(m_{0,k}, \mathsf{view}_k(w_j))$. Par hypothèse d'induction, $m = \widetilde{\mu_k}^*(m_{0,k}, \mathsf{fill}(w_j)[k])$.

Dès lors, on peut démontrer l'hypothèse d'induction pour w_i ; on posant $m' = \mu_k^*(m_{0,k}, \mathsf{view}_k(w_i))$, on obtient :

$$\begin{aligned} m' &= \mu_k^*(m, p \cdot e) \\ &= \widetilde{\mu_k}^*(m, p \cdot e \cdot (\widehat{e} \cdot e)^{2L-1}) && \text{(définition de } \mu_k) \\ &= \widetilde{\mu_k}^*(m, p \cdot e \cdot (\widehat{e} \cdot e)^{L-1} \cdot (\widehat{e} \cdot e)^{L}) \\ &= \widetilde{\mu_k}^*(m, p \cdot e \cdot (\widehat{e} \cdot e)^{L-1} \cdot (e \cdot \widehat{e})^{(2i-2j-1)L}) && \text{(équation 7)} \\ &= \widetilde{\mu_k}^*(m_{0,k}, \mathsf{fill}(w_j)[k] \cdot p \cdot e \\ &\qquad \cdot (\widehat{e} \cdot e)^{L-1} \cdot (e \cdot \widehat{e})^{(2i-2j-1)L}) && \text{(hyp. d'induction)} \\ &= \widetilde{\mu_k}^*(m_{0,k}, \mathsf{fill}(w_i)[k]) && \text{(équation 9)} \end{aligned}$$

La partie infinie $\mathsf{fill}(w)$ dans $\widetilde{G}$ est donc bien consistante avec $\widetilde{\sigma}$. □

Sachant que $\widetilde{\sigma}$ est gagnante, et en vertu de la remarque 3.1.9, cela suffit pour prouver que σ est gagnante, ce qui démontre le lemme 3.1.6.

Remarque 3.1.11 La stratégie σ n'est pas plus complexe que $\widetilde{\sigma}$. De fait, elle utilise une mémoire de même taille.

3.2 Synchronisation des jeux linéaires

Comme nous l'avons déjà évoqué, le problème de la synthèse distribué est, en général, indécidable [18, 20]. L'existence d'une stratégie distribué gagnante pour les processus dans les jeux distribués et donc tout aussi indécidable. Cette indécidabilité est même quasi immédiate puisqu'elle est déjà vrai dans le cas de jeux distribué a deux processus avec condition d'accessibilité ($\mathcal{W} = \emptyset$) ou de sureté ($\mathcal{W} = (E + P)^\omega$) [8]. Cependant, dès lors que les jeux distribués satisfont la condition de linéarité le problème de la résolution des jeux distribués devient décidable.

A la lumière de cette classe de jeux décidable, la réduction des jeux asynchrones en jeux synchrones que nous proposons ci-dessus n'est pas satisfaisante. En introduisant un non determinisme global de l'environnement cette réduction détruit en particulier toute propriété de linéarité : alors que le flot d'information peut être linéaire dans un jeux G asynchrone il n'a aucune raison de le rester dans le jeux synchrone équivalent $\widetilde{G}$.

Nous proposons dans cette section une modification de notre construction qui permet de préserver cette linéarité.

Plus généralement, les cas de jeux décidables, comme le cas des jeux linéaires, s'appuient sur la capacité des processus d'inférer, à la volée, une connaissance des positions de jeux dans quelques autres arènes locales. La construction que nous proposons ici consiste, après avoir synchronisé un jeux asynchrone, a recréer par une opération de normalisation les relations de connaissance entre processus qui existait avant l'application de la synchronisation. Présentée dans le cas des jeux linéaires, cette construction se généralise sans peine à tout schéma de connaissance implicite qui sera alors explicité. Les algorithmes de résolution des jeux qui s'appuieraient sur cette relation de connaissance pourront donc toujours être appliqué après synchronisation et normalisation.

Définition 3.2.1
Etant donné un jeu distribué à n-processus $G = \langle P, E, T_P, T_E, e_0, \mathcal{W}\rangle$*, on définit le jeu* $lin(G) = \langle P', E', T'_P, T'_E, e'_0, \mathcal{W}'\rangle$*, appelé la* linéarisation *du jeu* G*, comme étant le jeu définit à partir du jeu* G *de la façon suivante :*

1. *pour tout* $i \in [1, n]$ *:*
   (a) $P'_i = P[1, i] - E[1, i] + \{\bot_i\}$,
   (b) $E'_i = E[1, i]$,
   (c) $T'_{P,i} = T_P[1, i] \cap (P'_i \times E'_i)$,
2. *et pour tout* $e \in E' = \prod_{i\in[1,n]} E'_i$ *:*
   (a) *soit la position* e *est* coherent w.r.t. game G *au sens où, pour tout* $i \in [1, n]$*, on a* $e[i] = (e[n])[1, i]$*, alors nous posons* $(e, p) \in T'_E$ *pour toute position* $p \in P'$ *telle que* $\forall i \in [1, n], (e[i], p[i]) \in T_E[1, i]$,

(b) soit la position e est incohérente et nous posons $(e, \perp) \in T'_E$ *avec* $\perp = (\perp_1, \cdots, \perp_n)$.

3. $e'_0 = (e_0[1], e_0[1,2], \cdots, e_0[1,n-1], e_0[1,n])$,
4. *et* $\mathcal{W}' = \{w \in (P' + E')^\omega : w[n] \in \mathcal{W}\}$.

Remarque 3.2.2 Dans le jeu $lin(G)$, chaque fois que les processus atteignent une position incohérente, l'environnement ne peut plus que jouer dans la position $\perp$ où les processus perdent. Il apparait donc que les positions pertinentes du jeux, c'est à dire les positions où l'équipe des processus jouera, ne pourront être que des positions cohérentes c'est à dire toute position $x \in E' + P'$ telle que, étant donné $y = x[n] \in P + E$, on aura $x = (y[1], y[1,2], \cdots, y[1,n-1], y[1,n])$.

Autrement dit, dans toute position cohérente x du jeu $lin(G)$, pour tout indice $i \in [1,n]$, le processus i connait explicitement la position $x[j]$ de tout processus j tel que $1 \leq j \leq i$. Ainsi, $lin(G)$ est non seulement un jeu linéaire, mais cette linéarité est explicite.

Formellement, les stratégies distribuées dans les jeux G et $lin(G)$ correspondent les unes aux autres de la façon suivante.

Lemme 3.2.3 Pour toute stratégie distribuée gagnante $\sigma_1 \otimes \cdots \otimes \sigma_n$ dans le jeu G, la stratégie distribuée $\sigma'_1 \otimes \cdots \otimes \sigma'_n$ dans le jeu $lin(G)$ définie, pour tout $i \in [1,n]$, par $\sigma'_i = \sigma_1 \otimes \cdots \otimes \sigma_i$, est une stratégie distribuée gagnante dans le jeu $lin(G)$.

En général, il n'y pas de construction inverse. En effet, les jeux de la forme $lin(G)$ sont linéaires. L'existence de stratégie distribuée gagnante y est donc décidable ce qui n'est pas le cas en général. Cependant, lorsque le jeux G est linéaire, la construction inverse est possible.

Lemme 3.2.4 Pour tout jeux distribué linéaire G est linéaire, pour toute stratégie distribuée gagnante $\sigma'_1 \otimes \cdots \otimes \sigma'_n$ dans le jeu $lin(G)$ il existe une stratégie distribuée gagnante $\sigma_1 \otimes \cdots \otimes \sigma_n$ dans le jeu G telle que, pour tout $i \in [1,n]$, on ait $\sigma'_i = \sigma_1 \otimes \cdots \otimes \sigma_i$.

Proof. Observons tout d'abord que, parce que la stratégie $\sigma'_1 \otimes \cdots \otimes \sigma'_n$ est gagnante dans le jeu $lin(G)$ elle ne passe que sur des positions cohérentes. Sans perte de généralité, on peut supposer que les stratégies locales suivies par les processus sont elles-même cohérente. Plus précisément, on peut supposer que, pour tout indices i et j tels que $1 \leq i < j \leq n$, pour toute partie globale finie w dans le jeu $lin(G)$, nous avons $\sigma'_i(view_i(w)) = \sigma'_j(view_j(w))[1,i]$

Cela étant dit, la propriété énoncée vient immédiatement de l'étude des jeux linéaires présentés précédemment (voir section). La stratégie distribué $\sigma_1 \otimes \cdots \otimes \sigma_n$ peut être inductivement définie de la façon suivante.

Premièrement, la stratégie σ_1 est juste définie comme étant la stratégie σ'_1. En effet, à l'exception de la position $\perp_1$ qui n'est de toute façon pas atteinte, les jeux locaux $lin(G)[1]$ et $G[1]$ sont essentiellement les mêmes.

Ensuite, pour tout $i \in [2, n]$, la stratégie σ_i est inductivement définit à partir de la stratégie $\sigma'_{i-1} = \sigma_1 \otimes \cdots \otimes \sigma_{i-1}$ et de la stratégie σ'_i en *mimant*, à partir de la connaissance initiale de la position e_0, de la connaissance de stratégie σ'_{i-1} et de la partie localement jouée w_i dans le jeu $G[i]$, la partie w (unique par linéarité) qui est jouée sur la projection $G[1, i]$ du jeu G et qui satisfait $w_i = view_i(w)$. A partir de cette observation, nous prenons $\sigma_i(w_i) = \sigma'_i(w)$. La linéarité nous assure qu'un tel mime peut toujours être effectué. □

Il vient :

Théorème 3.2.5
Pour tout jeu distribué linaire à n processus G asynchrone, le jeu $lin(\widetilde{G})$ est linéaire et synchrone, et il est de plus équivalent au jeu G au sens ou les processus admettent une stratégie distribuée gagnante dans le jeux G si et seulement si ils admettent une stratégie distribuée gagnante dans le jeu $lin(\widetilde{G})$.

Proof. Tout d'abord, dans le sens direct, toute stratégie distribuée gagnante $\sigma_1 \otimes \cdots \otimes \sigma_n$ induit, par composition avec la fonction cancel, une stratégie distribuée gagnante dans le jeux $\widetilde{G}$ qui, a son tour, en appliquant le Lemme 3.2.3, induit à son tour une stratégie distribuée gagnante dans le jeu $lin(\widetilde{G})$.

Réciproquement, en supposant qu'il existe une stratégie distribuée gagnante et à état finie dans le jeu $\widetilde{\sigma}$ dans le jeu $lin(\widetilde{G})$, il apparait que, en utilisant des arguments de saturation semblable à ceux utilisée dans le Théorème 3.1.3 (bien que la construction ne soit pas, à strictement parler, identique), on peut fabriquer une stratégie distribuée gagnante σ' dans le jeu $lin(G)$. Cette stratégie peut alors être traduite, par le Lemme 3.2.4 en une stratégie σ pour le jeu initial G. □

4 Conclusions

Nous avons donc présenté ici la définition des jeux distribués en énonçant et en démontrant quelques uns des théorèmes fondamentaux de la théorie qu'ils induisent.

Il s'agit tout d'abord des interconnections étroites qui existent entre la théorie des automate d'arbres alternants - incluant notamment le théorème de simulation de tout automate alternant par un automate non alternant et la notion de composition d'automate qui devient possible - et les jeux distribués, ces premiers permettant de résoudre ces second dans le cas d'architecture linéaire.

Il s'agit ensuite de cette réduction des jeux asynchrones aux jeux synchrones. Il est connu des concepteurs de systèmes asynchrones que les différences réelles avec les systèmes synchrones viennent, par exemple dans un système de communication par files d'attente de messages, du caractère non borné de ces files

d'attente. Autrement dit, en l'absence de mémoire non bornée, l'étude des systèmes asynchrone pourrait intuitivement se ramener à l'étude des systèmes synchrones. Notre approche nous permet d'énoncer et de prouver formellement cette intuition.

Bien entendu, les jeux distribués présenté ici ne sont pas universels.

Par exemple, le cas du problème de la synthèse distribués avec spécifications locales [13] ne peut être couvert par notre approche. L'architecture manipulée, avec une entrée additionnelle sur le dernier élément du pipeline, est indécidable dans le cas de spécification externe. Notre technique d'externalisation ne semble donc pas applicable.

Les travaux de synthèses distribuées autours des formalismes de traces [5], ne se prettent pas non plus à une réduction à des jeux distribués. La structure des traces de Mazurkiewicz, qui code en un sens la "true concurrency", ne semble pas pouvoir se réduire à des jeux distribués qui codent, implicitement, une sémantique d"'interleaving'.

Aujourd'hui, depuis les travaux évoqués ci-dessus, de nombreuse autres pistes ont été ouvertes. Elles repoussent encore plus loin les frontière de l'indécidabilité mais leur évocation sort du périmètre de cet article.

Retenons cependant que si les jeux distribués pourraient permettre, via la notion de composition parallèle d'arènes locales, de représenter une certaine structuration des donnés à la manière des algèbres de processus, ils héritent cependant d'une propriété particulièrement désastreuse de ces approches : la composition parallèle de composants interagissants n'est pas compositionnelle.

Plus précisément, quelles que soient les propriétés des composants mis en oeuvre, les propriétés globales résultant du système obtenu par composition parallèle des composants restent particulièrement imprédictibles. Autrement dit, l'interopérabilité des composants semblent devenir l'un des problèmes les plus difficiles à résoudre lors du développement de systèmes distribués.

Il manque peut-être et encore à concevoir et à intégrer dans les formalismes de spécification de problèmes de synthèses distribués des outils de représentation qui permettent tout à la fois la spécification fonctionnelle des composants tout en autorisant, de façon conservative, la composition spatiale et temporelle de ces mêmes spécifications.

Références

[1] J. Bernet. *Jeux distribués*. PhD thesis, LaBRI, University of Bordeaux I, November 2006.

[2] J. Bernet and D. Janin. Tree automata and discrete distributed games. In Maciej Liśkiewicz and Rüdiger Reischuk, editors, *Fundamentals of Computation Theory*, volume 3623 of *LNCS*, pages 540–551, Lübeck, August 2005. Springer.

[3] J. Bernet and D. Janin. On distributed program specification and synthesis in architectures with cycles. In *26th IFIP W.G 6.1 Int. Conf. on Formal*

Techniques for Networked and Distributed Systems (FORTE), volume 4229 of *LNCS*, pages 175–190. Springer, 2006.

[4] J. Bernet and D. Janin. From asynchronous to synchronous specifications for distributed program synthesis. In *Int. conf. on Current Thrends in Theo. and Prac. Comp. Science (SOFSEM)*, volume 4910 of *LNCS*, pages 162–173. Springer, 2008.

[5] P. Gastin, B. Lerman, and M. Zeitoun. Distributed games with causal memory are decidable for series-parallel systems. In P. K. Pandya and J. Radhakrishnan, editors, *In Proc. 24th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'04)*, volume 3328 of *LNCS*, pages 275–286. Springer, 2004.

[6] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics and Infinite Games*, volume 2500 of *LNCS*. Springer, 2002.

[7] D. Janin. *A contribution to formal methods : games, logic and automata*. Habilitation à diriger les recherches, LaBRI, University of Bordeaux 1, December 2005.

[8] D. Janin. On the (high) undecidability of distributed synthesis problems. In *Int. conf. on Current Thrends in Theo. and Prac. Comp. Science (SOFSEM)*, volume 4362 of *LNCS*, pages 320–329. Springer, 2007.

[9] O. Kupferman and M. Y. Vardi. Church's problem revisited. *Bulletin of Symbolic Logic*, 5 :245–263, 1999.

[10] O. Kupferman and M. Y. Vardi. Synthesizing distributed systems. In *In Proc. IEEE Symposium on Logic in Computer Science (LICS'01)*, pages 389–398, 2001.

[11] F. Lin and M. Wonham. Decentralized control and coordination of discrete event systems with partial observation. *IEEE Transactions on automatic control*, 33(12) :1330–1337, 1990.

[12] P. Madhusudan. *Control and synthesis of open reactive systems*. PhD thesis, University of Madras, 2001.

[13] P. Madhusudan and P.S. Thiagarajan. Distributed controller synthesis for local specifications. In *28th International Colloquium on Automata, Languages and Programming (ICALP'01)*, volume 2076 of *LNCS*, pages 396–407. Springer, 2001.

[14] R. McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65 :149–184, 1993.

[15] S. Mohalik and I. Walukiewicz. Distributed games. In P. K. Pandya and J. Radhakrishnan, editors, *In Proc. 23th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'03)*, volume 2914 of *LNCS*, pages 338–351. Springer, 2003.

[16] David E. Muller and Paul E. Schupp. Alternating automata on infinite trees. *Theoret. Comput. Sci.*, 54(2-3) :267–276, 1987.

[17] D.E. Muller and P.E. Schupp. Simulating alternating tree automata by non-deterministic automata : New results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theoret. Comput. Sci.*, 141 :69–107, 1995.

[18] G.L. Peterson and J.H. Reif. Multiple-person alternation. In *20th Annual IEEE Symposium on Foundations of Computer Science (FOCS'79)*, pages 348–363, october 1979.

[19] G.L. Peterson, J.H. Reif, and S. Azhar. Decision algorithms for multiplayer non-cooperative games of incomplete information. *Computers and Mathematics with Applications*, 43 :179–206, january 2002.

[20] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *In Proc. 16th ACM Symposium on Principles of Programming Languages*, pages 179–190, 1989.

[21] A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *In Proc. 31th IEEE Symposium on Foundations of Computer Science (FOCS'90)*, pages 746–757, 1990.

[22] M.O. Rabin. Decidability of second order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141 :1–35, 1969.

[23] A. Vincent. Synthèse de contrôleurs et stratégies gagnantes dans les jeux de parité. In Hermès, editor, *Modélisation des Systèmes Réactifs*, pages 87–98, 2001.