Tree automata and discrete distributed games

Julien Bernet, David Janin

Tree Automata and Discrete Distributed Games

Julien Bernet and David Janin *

LaBRI, Université de Bordeaux I
351, cours de la Libération
33 405 Talence cedex FRANCE
{bernet|janin}@labri.fr

Abstract. Distributed games, as defined in [6], is a recent multiplayer extension of discrete two player infinite games. The main motivation for their introduction is that they provide an abstract framework for distributed synthesis problems, in which most known decidable cases can be encoded and solved uniformly.
In the present paper, we show that this unifying approach allows as well a better understanding of the role played by classical results from tree automata theory (as opposed to adhoc automata constructions) in distributed synthesis problems. More precisely, we use alternating tree automata composition, and simulation of an alternating automaton by a non-deterministic one, as two central tools for giving a simple proof of known decidable cases.

Introduction

Distributed games, as defined in [6], is a recent multiplayer extension of discrete two player infinite games. The main motivation for their introduction is that they provide an abstract framework for distributed synthesis problems, in which most known decidable cases [1, 3–5, 10] can be encoded and solved uniformly.

In the present paper, we show that this unifying approach allows as well a better understanding of the role played by classical results from tree automata theory in distributed synthesis problems.

More precisely, in the above mentioned works, many decision algorithms rely (more or less implicitly) on automata constructions that are not explicitly related to classical automata theory.

For instance, in [3], the main construction given by the authors to solve the pipeline synthesis problem "sounds" like the sequential composition of two tree-automata. Similarly, one of the main construction (glue operation) defined in [6] "sounds" like Muller and Schupp simulation of an alternating automaton by a non deterministic one [7].

The purpose of this paper is to validate this intuition, by explicitly defining the encountered automata (or their inputs) when they are missing in these works, and to apply known constructions in order to reprove these synthesis results.

* This work is partially supported by the European Commission Research and Training Network "Games and Automata for Synthesis and Validation" (RTN GAMES)

This way, it is expected that it will contribute to the foundation of a common ground into which methods and approaches can be encoded and compared one with the other.

The technical relevance of our reformulation work is illustrated by the encoding and solving of the pipeline case [3].

Other Related Works

Peterson and Reif ([8], extended in [9]) initiated the research on multiplayer games of incomplete information, considering finite games, and introducing the notion of *hierarchical games*: these games satisfy the property that one can linearly order the set of players such that $p_1 \leq p_2$ if and only if "$p2$ knows more than p_1", or equivalently "p_2 knows the state of p_1". They prove that these games are solvable, by iteratively removing the incomplete information associated with each player.

Subsequent results on distributed synthesis (such as [10], [3]) essentially used the same ideas and techniques, except in the fact that they consider infinite plays and/or branching time specifications.

The common technique is to cut out the last player from the game (i.e. the one that knows the state of all the other), modifying in the process the specification so that it reflects all moves that can be taken by this player, then do the same with the last but one, etc. ... until a "simple" 2-player game is left to solve.

Our paper rely on the same principle, making the automata constructions explicit.

Organization of the Paper

In the first section, after reviewing some of the notations used in this paper, we fix the definitions of trees, tree automata and infinite two player games. Muller and Schupp non determinization theorem is stated, and a notion of sequential composition of tree automata is also defined and analyzed.

Distributed games and distributed strategies are presented in the second section. These games are played by a team of process players versus a single environment opponent. Each process player only gets incomplete information about the position of the other processes. The existence of a winning distributed strategy in a distributed game is shown to be undecidable, even for simple winning conditions such as safety and reachability.

In the third section, we first show that using an (external) tree-automaton in order to define winning strategies in a (distributed) game is essentially equivalent to adding an additional process player (internalizing the automaton) into the game. Then, conversely, we show that when a process player has enough knowledge to deduce the positions of the other processes, then its local arena can be externalized as a tree automaton reading the strategies of the remaining processes, in which non-deterministic choices correspond to the moves of the process; this automaton can be composed with any existing external winning condition.

Under sufficient conditions, one can apply repeatedly this construction in order to reduce the number of process players and thus to solve the distributed game.

The long-term goal of this approach is to gain benefit from the high level of abstraction provided by game theory and, altogether, gain benefit from well-known constructions of automata theory (as it as been developed from Rabin's seminal result [11]), to help having a better understanding of the fundamental obstacles to the synthesis of distributed systems.

1 Trees, Automata and Games

For any alphabet A, let A^* and A^ω be the set of all finite and infinite words with letters from A. Let $A^\infty = A^* \cup A^\omega$, and $A^? = \{\epsilon\} + A$. Standard notations on words and languages of words are used. In particular, given a language $L \subseteq A^*$, we use the notations L^+ and (when the empty word $\epsilon \notin L$) L^ω that stand, respectively, for the set of words built by concatenating finitely many and infinitely many finite words of L. For any finite word $w = a_1 \dots a_n$, let $|w| = n$ be the length of w. For any infinite word w, let $\inf(w) = \{a \in A \mid w \in (A^*.a)^\omega\}$ be the set of letters that occur infinitely often in w.

For any two sets A and X, for any word $w \in A^*$, define $\pi_X(w)$ (the projection of w over X) as the word obtained by deleting any letter that is not in X from the word representation of w.

Given n numbered sets $A_1, \dots, A_n$, given $A = A_1 \times \dots \times A_n$, given any set of indices $I = \{i_1, \dots, i_k\} \subseteq \{1, \dots, n\}$ with $i_1 < \dots < i_k$, we write $A[I]$ for the set $A[I] = A_{i_1} \times \dots A_{i_k}$, for any $x = (a_1, \dots, a_n) \in A$, we write $x[I]$ for the elements $x[I] = (x_{i_1}, \dots, x_{i_k}) \in A[I]$, and, for any $P \subseteq A$, we write $P[I]$ for the set $P[I] = \{x[I] \in A[I] : \ x \in P\}$.

In case $I = \{i, i+1, \dots, j\}$ (where $1 \leq i \leq j \leq n$), these notations simplify to $A[i \dots j]$, $x[i \dots j]$ and $P[i \dots j]$ respectively (and even simplify to $A[i]$, $x[i]$ and $P[i]$ when $i = j$). These notations also extend to words as follows: for any word $w = a_1.a_2. \dots \in A^\infty$, for any $I \subseteq \{1, \dots, n\}$, $w[I] = a_1[I].a_2[I].a_3[I] \dots$, and to relations: for any relation $R \subseteq A \times A$, we write $R[I]$ the relation on $A[I]$ defined by $R[I] = \{(x[I], y[I]) \in A[I] \times A[I] : \ (x, y) \in R\}$.

Given two finite alphabets D and Σ, a *Σ-labeled D-tree* (also called *D,Σ-tree*) is a partial function $D^* \to \Sigma$ whose domain is closed under prefix operation. In the sequel, elements of Σ are called *labels* and elements of D are called *directions*.

For any tree $t : \ D^* \to \Sigma$, the function $flat_t : \ Dom(t) \to \Sigma.(D.\Sigma)^*$ is defined by: $flat_t(\epsilon) = t(\epsilon)$ and, for any $w \in D^*$ and $d \in D$ such that $w.d \in Dom(t)$, $flat_t(w.d) = flat_t(w).d.t(w.d)$. Observe that $Dom(t)$ and $flat_t(Dom(t))$ ordered by the prefix ordering are isomorphic, and, as a consequence, $flat_t(Dom(t))$ uniquely determines tree t.

The following definition is a variation on Muller and Schupp's original definition of alternating automaton [7]. Our goal is to have a tree-transducer like automaton definition, even for alternating automaton.

Definition 1 (Alternating tree automaton). *A finite* (D, Σ)-alternating tree automaton *is a tuple:*

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

where Q is a finite set of states, $q_0 \in Q^{\exists}$ is the initial state, $\delta^{\forall} : Q^{\forall} \times D \to \mathcal{P}(Q^{\exists})$ and $\delta^{\exists} : Q^{\exists} \times \Sigma \to \mathcal{P}(Q^{\forall})$ are the transition functions, and the ω-rational language Acc is the infinitary acceptance criterion.

Automaton $\mathcal{A}$ is a *non deterministic automaton* (also called non alternating) when $|\delta^{\forall}(q, d)| \leq 1$ (for any $q \in Q^{\forall}$, $d \in D$).

Definition 2 (Runs). *A* run *of an automaton $\mathcal{A} = \langle Q, D, \Sigma, i, \delta, Acc \rangle$ over a Σ-labeled D-tree $t : D^* \to \Sigma$ is a $Q^{\forall}$-labeled $(D \times Q^{\exists})$ tree $\rho : (D \times Q^{\exists})^* \to Q^{\forall}$ such that:*

- *$\rho(\epsilon) \in \delta^{\exists}(q_0, t(\epsilon))$,*
- *for all $w \in Dom(\rho)$, if $\rho(w) = q$, then for any direction $d \in D$ such that $a = t(w[1].d)$ is defined, and for any existential state $q_1 \in \delta^{\forall}(q, d)$, there exists a universal state $q_2 \in \delta^{\exists}(q_1, a)$ such that $\rho(w.(d, q_1)) = q_2$.*

For any infinite branch w of a run ρ of $\mathcal{A}$ over t, $states_{\rho}(w)$ is the sequence of (universal and existential) states encountered along w. A tree t is accepted *by $\mathcal{A}$ if and only if there exists a run ρ of $\mathcal{A}$ over t such that for any infinite branch w in ρ: $states_{\rho}(w) \in Acc$. Denote by $L(\mathcal{A})$ the language of all trees that are accepted by $\mathcal{A}$. The* size *of an automaton $\mathcal{A}$ is denoted by $|\mathcal{A}|$.*

Observe that these tree automata (both alternating and non alternating), if slightly unusual, have the same expressive power as their standard counterpart, as in [7]. In particular:

Theorem 1 (Simulation [7]). *Any alternating tree automaton $\mathcal{A}$ is equivalent to a non deterministic tree automaton $\mathcal{A}'$, with $|\mathcal{A}'| \leq 2^{2^{|\mathcal{A}|}}$ (with Muller acceptance condition).*

Since the runs of an automaton on trees are themselves trees, automata act as tree transducers and can be sequentially combined.

Definition 3 (Automata Composition). *Given two tree automata $\mathcal{A}_1 = \langle Q_1, D_1, \Sigma_1, q_{0,1}, \delta_1, Acc_1 \rangle$ and $\mathcal{A}_2 = \langle Q_2, D_2, \Sigma_2, q_{0,2}, \delta_2, Acc_2 \rangle$, such that automaton $\mathcal{A}_2$ is non deterministic with $D_2 = D_1 \times Q_1^{\exists}$ and $\Sigma_2 = Q_1^{\forall}$, we define the composition of $\mathcal{A}_1$ followed by $\mathcal{A}_2$ to be the automaton*

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

defined as follows:

- *$\widetilde{Q^{\exists}} = Q_1^{\exists} \times Q_2^{\exists}$; $\widetilde{Q^{\forall}} = Q_1^{\forall} \times Q_2^{\forall}$;*
- *$q_0 = (q_{0,1}, q_{0,2})$,*

- $(q_1', q_2') \in \widetilde{\delta^\forall}((q_1, q_2), d) \Leftrightarrow \begin{cases} q_1' \in \delta^\forall(q_1, d) \\ \{q_2'\} = \delta_2^\forall(q_2, (d, q_1')) \end{cases}$
- $(q_1', q_2') \in \widetilde{\delta^\exists}((q_1, q_2), a) \Leftrightarrow \begin{cases} q_1' \in \delta^\exists(q_1, a) \\ q_2' \in \delta_2^\exists(q_2, q_1') \end{cases}$
- $\widetilde{Acc} = \{w \in \widetilde{Q}^\omega \mid w[1] \in Acc_1 \wedge w[2] \in Acc_2\}$

Theorem 2. *For any tree $t : D_1^* \to \Sigma_1$, $t \in L(\mathcal{A}_2 \circ \mathcal{A}_1)$ if and only if there exists an accepting run $\rho : (D_1 \times Q_1^\exists)^* \to Q_1^\forall$ of $\mathcal{A}_1$ over t such that $\rho \in L(\mathcal{A}_2)$.*

The proof, although tedious, is not complicated, and is therefore omitted here. Observe that it is crucial that $\mathcal{A}_2$ is non-alternating ; nevertheless, by applying Theorem 1, one can always assume that is is the case.

Definition 4 (Simple (or Two Player) Games). *A* simple arena *is a quadruple $G = \langle P, E, T_P, T_E \rangle$, where P is a finite set of Process positions, E is a finite set of Environment positions, $T_P \subseteq P \times E$ is the set of Process moves, $T_E \subseteq E \times P$ is the set of Environment moves. A* simple game *$G = \langle P, E, T_P, T_E, e_0, \mathcal{W} \rangle$ is built upon a simple arena $\langle P, E, T_P, T_E \rangle$ by equipping it with an initial position $e_0 \in E$ and a regular winning condition $\mathcal{W} \subseteq (P + E)^\omega$.*

As particular cases of winning condition, a *reachability condition* is a winning condition of the form $\mathcal{W} = (P + E)^*.X.(P + E)^\omega$ for some set of positions $X \subseteq P + E$ to be reached for Process to win, and a *safety condition* is a winning condition of the form $\mathcal{W} = ((P + E) - X))^\omega$ for some set of positions $X \subseteq P + E$ to be avoided for Process to win.

A *play* $w \in (P + E)^*$ in a simple game is any non-empty path in the arena beginning on e_0. A play w is winning for Process when either it is finite and ends in an Environment position, or it is infinite and belongs to $\mathcal{W}$. Otherwise, it is winning for Environment.

A *strategy* for Process is a partial function $\sigma : (E.P)^+ \to E$ such that for any $w.p \in Dom(\sigma)$, for any position $e \in \sigma(w.p)$, then $(p, e) \in T_P$, and for any successor p' of e, $w.p.e.p' \in Dom(\sigma)$. A play $w = e_0.x_1 \ldots$ is *consistent* with strategy σ when, for any $i \in \mathbb{N}$, if $\sigma(e_0 \ldots . x_i)$ and x_{i+1} are both defined then they are equal. A strategy σ is a *winning strategy* for Process when any maximal play (w.r.t. the prefix ordering) consistent with σ is winning for Process.

Given a strategy σ in some game G, the *strategy tree* $t_\sigma : P^* \to E$ of σ in G is defined inductively by $t_\sigma(\epsilon) = e_0$, and $t_\sigma(u.x) = \sigma(flat_{t_\sigma}(u).x)$.

Theorem 3 ([2]). *On finite two-player games with regular winning condition, either Process or Environment has a winning strategy, which can be computed effectively.*

2 Distributed Games

Definition 5 (Distributed Arena). *A* distributed arena *is a free asynchronous product where the possible Environment moves may have been restricted.*

More precisely, given two arenas $G_1 = \langle P_1, E_1, T_{P,1}, T_{E,1}\rangle$ *and* $G_2 = \langle P_2, E_2, T_{P,2}, T_{E,2}\rangle$*, a (two-process) distributed arena built upon the arenas* G_1 *and* G_2 *is any simple arena* $G = \langle P, E, T_P, T_E\rangle$ *of the form*

- Environment positions : $E = E_1 \times E_2$,
- Processes positions : $P = (E_1 \cup P_1) \times (E_2 \cup P_2) - (E_1 \times E_2)$,
- Processes moves *:* T_P *is the set of all pairs* $(p, e) \in (P \times E)$ *such that, for* $i = 1$ *and* $i = 2$ *:*
  - **either** $p[i] \in P_i$ *and* $(p[i], e[i]) \in T_{P,i}$ *(Process* i *is* active in p*),*
  - **or** $p[i] \in E_i$ *and* $p[i] = e[i]$ *(Process* i *is* inactive in p*),*
- *and* Environment moves *:* T_E *is* **some subset** *of the set of all pairs* $(e, p) \in (E \times P)$ *such that, for* $i = 1$ *and* $i = 2$ *:*
  - **either** $p[i] \in P_i$ *and* $(e[i], p[i]) \in T_{P,i}$ *(Environment activates Process* i*),*
  - **or** $p[i] \in E_i$ *and* $p[i] = e[i]$ *(Environment keeps Process* i *inactive).*

When the set T_E *of Environment moves is maximal, we call such an arena the* free asynchronous product *of arenas* G_1 *and* G_2 *and it is denoted by* $G_1 \otimes G_2$*.*

These definitions extend to n*-process distributed arena.*

Since a distributed arena is built upon n simple arenas, we need a definition to speak about its local components:

Definition 6 (Projection of distributed arena). *Given a distributed arena* $G = \langle P, E, T_P, T_E\rangle$*, with* $E = E_1 \times \ldots \times E_n$ *and* $P = ((P_1 \cup E_1) \times \ldots \times (P_n \cup E_n)) - E$*, given a non empty set* $I \subseteq \{1, \ldots, n\}$*, define the* canonical projection $G[I]$ *of* G *on* I *as the arena* $G[I] = \langle P', E', T'_P, T'_E\rangle$ *given by:* $P' = P[I] - E[I]$ *(possibly smaller than* $P[I]$ *!),* $E' = E[I]$*,* $T'_P = T_P[I] \cap (P[I] \times E[I])$*, and* $T'_E = T_E[I] \cap (E[I] \times P[I])$*.*

Remark. Observe that a n-process distributed arena G as above can always be seen as a distributed arena built upon the games $G[1]$, ..., $G[n]$. Moreover, in the same way Cartesian product of sets is (up to isomorphism) associative, given an arbitrary non empty set $I \subset \{1, \ldots, n\}$, given $\overline{I} = \{1, \ldots, n\} - I$, the n-process distributed arena G can, as well, be seen as a distributed arena built upon the two (distributed) arenas $G[I]$ and $G[\overline{I}]$.

*Example 1 (**The Pipeline : Beginning**).* A *distributed architecture* (as defined in [10], [3]) is a set of *sites* linked together by some *communication channels*. Each site can host a *program*, which is essentially a sequential function[1] mapping a sequence of inputs to a sequence of outputs. As a typical example, in a *pipeline architecture*, the sites are linearly ordered from left to right, each site taking its input from the site on its right, and writing its output to the site on its left.

To be more precise, suppose each communication channel x_i can carry values that range over some set X_i. The site s_i receives its input from the channel x_i, and writes its outputs to the channel x_{i-1}; thus, a program for the site s_i is a

[1] recall that a sequential function is a function $f : A^* \to B^*$ that is realized by a word transducer with input alphabet A and output alphabet B.

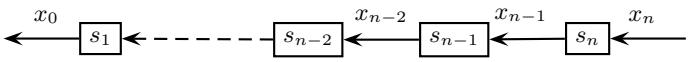

Fig. 1. A pipeline architecture

sequential function $f_i : X_i^* \to X_{i-1}^*$. The environment writes input to the system on channel x_n, and the system's output is read on channel x_0.

For any pipeline architecture $\mathbb{A}$, we can build a distributed arena $G_{\mathbb{A}} = \langle P, E, T_P, T_E \rangle$ where each process plays the role of a program: on its local arena, the environment's moves correspond to the possible inputs for this site, and the process moves correspond to the possible outputs:

- $P = X_1 \times \ldots \times X_n$; $E = X_0 \times \ldots \times X_{n-1}$.
- $((v_1, \ldots, v_n), (v'_1, \ldots, v'_n)) \in T_E$ iff $v'_i = v_{i+1}$ for each $i \in \{1, \ldots, n-1\}$ and $v'_n \in X_n$.

Observe that by restricting the Environment moves, we ensure that the environment carries correctly the values along the channels.

Definition 7 (Distributed Games). *A n-process* distributed game G *is a tuple*

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

where $\langle P, E, T_P, T_E \rangle$ is a n-process distributed arena, $e_0 \in E$ is the initial (Environment) position, and $\mathcal{W} \subseteq (E.P)^\omega$ is the (regular) winning infinitary condition.

A distributed game is a particular case of simple game. It follows that previous notions of plays and strategies are still defined. However, in order to avoid confusion with what may happen in the local arena a distributed game is build upon, we shall speak now of a *global play* and a *global strategy*.

The *local view* Process i has of a global play in a distributed game G is given by the map $view_i : \ (E.P)^*.E^? \to (E_i.P_i)^*.E_i^?$ defined in the following way:

- $view_i(\epsilon) = \epsilon$
- $view_i(x) = x[i]$
- $view_i(w.x.y) = \begin{cases} view_i(w.x) \text{ if } x[i] = y[i] \\ view_i(w.x).y[i] \text{ otherwise.} \end{cases}$

A play $w \in (E.P)^+$ is said to be active for Process i when w ends in a position $p \in P$ such that $p[i] \in P[i]$.

Definition 8 (Local and distributed Strategies). *Given a n-tuple of local strategies $(\sigma_i : \ (E[i].P[i])^+ \to E[i])_{i \in \{1,\ldots,n\}}$, the induced global strategy*

$$\sigma_1 \otimes \ldots \otimes \sigma_n : \ (E.P)^+ \to E$$

is defined as follows: for any play of the form $w.p \in (E.P)^+$, given the set $I \subseteq \{1, \ldots, n\}$ of active processes in the global Processes position p (i.e. $I = \{i \in \{1, \ldots, n\} : \ p[i] \in P_i\}$), define $\sigma(w.p) = e$ by:

– $e[i] = \sigma_i(view_i(w))$ *for* $i \in I$
– $e[i] = p[i]$ *for* $i \in \{1, \ldots, n\} - I$

(provided everything is well-defined, otherwise $\sigma(w.p)$ *is left undefined).*

A global strategy $\sigma : (E.P)^+ \to E$ *is a* distributed strategy *if* σ *equals the composition* $\sigma_1 \otimes \ldots \otimes \sigma_n$ *of some* n *local strategies.*

Note that global strategies are not always distributed. Moreover, there are distributed games in which the Processes have a winning strategy, but no winning distributed strategy.

From this, we can derive an important fact: the distributed game are not determined, in the sense that even when the environment does not have a winning strategy, the processes may not have a winning *distributed* strategy. Furthermore, using the fact that the processes do not share the same information, we are able to provide the following undecidability result:

Theorem 4. *The problem of finding a winning distributed strategy in a 3-process distributed game with safety or reachability winning condition is undecidable.*

The proof is omitted here due to space restriction. Suffice it to say that it proceeds by reduction to the Post correspondence problem, and relies heavily on the fact that there are three processes in the game. It is an open problem whether solving a 2-process distributed game is decidable or not.

3 Tree Automata and Distributed Games

We first mix games and automata, defining a winning condition by means of a tree-automaton that recognizes the set of trees of winning strategies. We illustrate this new concept by defining a pipeline game over the pipeline arena. Then, we present an algorithm to solve such a game, using the notion of leader in a distributed game.

Definition 9 (External Winning Condition). *A* game with external winning condition *is a tuple*

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

where $\langle P, E, T_P, T_E \rangle$ *is a simple arena,* $e_0 \in E$ *is the initial position, and* $\mathcal{A}$ *is a* (P, E)*-tree automaton. In such a game, a strategy is winning if its strategy tree belongs to* $L(\mathcal{A})$*. This definition extends to distributed games.*

In the sequel, in order to avoid confusion, a game with a winning condition defined as in section 2 is called *game with internal winning condition.*

As we are going to show, games with external winning condition are not essentially more expressive than games with internal one.

Theorem 5 (Internalization). *For any n-process game G with external winning condition, there exists a $n+1$-process game G' with internal winning condition such that $G'[1,\ldots,n] = G$, and such that the processes have a winning strategy σ in G if and only if the processes have a winning strategy of the form $\sigma \otimes \sigma'$ in G'.*

Proof. (sketch) Let $G = \langle P, E, T_P, T_E, e_0, \mathcal{A}\rangle$ (where $\mathcal{A} = \langle Q^\forall \uplus Q^\exists, P, E, q_0, \delta = \delta^\forall \cup \delta^\exists, Acc\rangle$) be a distributed game with external winning condition. The game $G' = \langle P', E', T'_P, T'_E, e'_0, \mathcal{W}\rangle$ is defined as follows. The positions and the winning condition are given by:

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

and moves are (repeatedly) defined by: from an environment position $(e,q) \in E \times Q^\exists$ (or the initial position):

1. first, Environment (deterministically) moves to the process position $(e,(q,e)) \in E \times (Q^\exists \times E)$,
2. then, the new (automaton) process locally chooses $q' \in \delta^\exists(q,e)$, the other processes stay idle, thus the play proceeds in G', to the environment position $(e,q') \in E \times Q^\forall$,
3. then, Environment chooses $p \in T_E(e)$ and $q_1 \in \delta^\forall(q',p)$, and the play proceeds to the Process position $(p,(q_1,\#)) \in P \times Q^\exists$,
4. finally, processes 1 to n (on game G) choose some $e_1 \in T_P(p)$, the new (automaton) process stays almost idle (he simply deletes the # sign), and the play proceeds to the Environment position $(e_1,q_1) \in E \times Q^\exists$.

If ρ is an accepting run of $\mathcal{A}$ over t_σ (for some strategy σ in G), one deduce from ρ a strategy σ' such that $\sigma \otimes \sigma'$ is winning in G'. Conversely, if $\sigma \otimes \sigma'$ is a winning strategy in G', one can infer an accepting run of $\mathcal{A}$ over t_σ from σ'.

Moreover, when G is a simple game with external winning condition, the internalization procedure can be further simplified (and amounts essentially to build the product of G with the automaton), and the resulting game with internal condition is a simple game as well.

*Example 2 (**Pipeline Example Continued**).* Following the presentation from [3], the synthesis problem for distributed architectures is presented as follows: given a distributed architecture $\mathbb{A}$ and a vector of programs $(f_i)_{1\leq i\leq n}$ (one for each site of $\mathbb{A}$), the *computation tree* of the system is a $(\prod_{1\leq i\leq n} X_i)$-labeled X_n-tree, where each node w is labeled by the values held by the communication channels after input w to the system.

A *specification* for the system is a language of such trees specified by a tree automaton $\mathcal{A}$ (or equivalently by a MSO-formula).

The synthesis problem is then: does there exists a vector of programs such that the resulting computation tree belongs to the specification ?

Building upon the pipeline arena $G_{\mathbb{A}}$ as defined in example 1, we can now define a distributed game in which the processes have a winning strategy if and only if there is a solution to the synthesis problem in the pipeline architecture. Suppose the specification for $\mathbb{A}$ is given by the finite $(X_n, \prod_{0\leq i\leq n-1} X_i)$-automaton $\mathcal{A}$, we can easily define a $(\prod_{1\leq i\leq n} X_i, \prod_{0\leq i\leq n-1} X_i)$-automaton $\mathcal{A}'$ that accepts a tree t' if and only if it is the widening of some tree $t \in \mathcal{L}(\mathcal{A})$ (i.e. if $t'(w) = t(w[n])$ for all $w \in Dom(t')$).

Using this automaton as an external winning condition, we get the encoding of the synthesis problem for the pipeline architecture in a distributed game.

Observe that in this game, for each $i \in \{1, \ldots, n\}$, provided that Process i knows the strategy for all the processes from 1 to $i-1$, then he can predict the position in each of the local arenas from 1 to $i-1$.

Using the above observation, one may ask now whether an inverse construction to internalization is possible or not. Intuitively, assuming that there is a process in an $n+1$-distributed game that can predict, at every step, what is the global position in the game, can we *externalize* it into an external winning condition such that, the resulting n-process distributed game with external condition is equivalent, in some effective sense, to the initial game ?

The notion of leader defined below follows this intuition. In fact, it provides a local condition that is sufficient for such a global knowledge to be available to a Process player.

Definition 10 (Leader). *Given a 2-process game $G = \langle P, E, T_P, T_E, e_0, \mathcal{A}\rangle$, we say that Process 2 is a leader when, for any Environment position $e \in E$, any Processes positions x and $y \in P$ such that both $(e, x) \in T_E$ and $(e, y) \in T_E$,*

- *if $x[2] = y[2]$ then $x[1] = y[1]$,*
- *if $x[2] \in E[2]$ or $y[2] \in E[2]$ then $x = y$.*

Intuitively, Process 2 is a leader when, as soon as he knows a global Environment position then, after an Environment move (or several consecutive moves if Process 2 stays idle for some time), Process 2 can predict, from his own position, the global Processes position of the game.

This local property has the following formulation when it comes to considering plays:

Lemma 1. *Let $G = \langle P, E, T_P, T_E, e_0\rangle$ be a 2-process arena with initial position e_0. For any strategy σ for the processes, the restriction of $view_2$ to the plays that are consistent with σ and active for Process 2 is one-to-one.*

Proof. Immediate from the definition.

Rephrased in a more useful way, this observation leads to the following result:

Lemma 2. *For any 2-process game $G = \langle P, E, T_P, T_E, e_0, \mathcal{W}\rangle$ such that Process 2 is a leader, there exists a $(P[1], E[1])$-automaton $\mathcal{A}_2$ such that for any strategies σ on G, σ_1 on G_1, the following propositions are equivalent:*

(1) *there exists a strategy* σ_2 *on* $G[2]$ *such that* $\sigma = \sigma_1 \otimes \sigma_2$
(2) *there is an accepting run* ρ *of* $\mathcal{A}_2$ *over* t_{σ_1} *such that* $\rho = t_{\sigma_1}$.

Proof. (sketch) We first give here a construction for $\mathcal{A}_2$ in the case both Process 1 and Process 2 are always active in the positions for Processes.

Automaton $\mathcal{A}_2 = \langle Q_2, P[1], E[1], q_{0,2}, \delta_2, Acc_2 \rangle$ can be defined as follows:

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

The correspondence between runs of $\mathcal{A}_2$ on strategy trees in $G[1]$ and strategy trees in G easily follows from this construction, and from the fact that Process 2 is a leader in G.

In the case Process 2 may be inactivated by Environment one can check that, since Process 2 is a leader, game G can be first normalized so that this no longer happens (details are not given due to lack of space).

In the case Process 1 may be inactivated by Environment, then the construction below can be extended, defining (quite easily though tediously) an automaton $\mathcal{A}_2$ with ϵ-transition. However, the main arguments remain the same.

Since the previous result holds for arbitrary external condition and arbitrary strategies in $G[1]$ (even if $G[1]$ is itself a distributed game), it follows:

Theorem 6 (Externalization).
For any n*-process distributed game* $G = \langle P, E, T_P, T_E, e_0, \mathcal{A} \rangle$ *with non deterministic external winning condition* $\mathcal{A}$ *such that Process* n *is a leader, there is a* $(P[1 \ldots n-1], E[1 \ldots n-1])$*-automaton* $\mathcal{A}_n$ *such that the following propositions are equivalent:*
(1) *the processes have a distributed winning strategy on* G.
(2) *the processes have a distributed winning strategy in* $\langle G[1 \ldots n-1], e_0[1 \ldots n-1], \mathcal{A} \circ \mathcal{A}_n \rangle$.

*Example 3 (**The Pipeline: End**).* We have already mentioned that, in the n-process pipeline arena, from any initial position, Process n is a leader. It follows that Theorem 6 applies.

Moreover, observe that the resulting $(n-1)$-process game arena $G[1 \ldots n-1]$ is nothing but a $(n-1)$-process pipeline arena. This says that Theorem 6 can be applied repeatedly till the number of processes is reduced to one. Now, one can internalize the automaton, and compute a winning strategy in the resulting simple game using Theorem 3.

Transposed on our more abstract setting, this can be expressed as the following corollary of the theorem.

Corollary 1. *For any* n*-process* $(n \geq 2)$ *distributed game* G *such that for each* $i \in \{2, \ldots, n\}$ *process* i *is a leader in* $G[1 \ldots i]$*, the problem of determining whether the processes have a winning strategy is decidable.*

Remark. At every step, the external condition we get from the composition is an alternating automaton that needs to be simulated by a non alternating one so that the composition can be iterated. This means that the complexity of solving the pipeline architecture synthesis problem by means of its encoding into a distributed game is a tower of exponents of depth at least the number of components in the pipeline. This (bad) complexity was expected, since this problem is non-elementary [10].

4 Concluding Remarks

We have defined a set of automata theoretic tools that can be used to solve various distributed synthesis problems, e.g. the pipeline architecture [3].

Compared to [6] we do obtain an automata theoretic interpretation of most of the operations defined there: in their approach, applying successively DIVIDE and GLUE to a game where both 0 and n are leaders amounts, in our setting, to externalize 0, to apply the simulation theorem, to externalize n, and eventually to internalize the resulting automaton.

Still, one application case presented by the authors to solve the local specification case [5] is not solved in this paper. This is left for further studies. There is a chance that tree automata theory will still provide arguments.

References

1. A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controllers with partial observation. to appear in Theoretical Computer Sciences, 2002.
2. E.A. Emerson and C.S. Jutla. Tree automata, mu-calculus and determinacy. In *Proc. 32th Symp. on Foudations of Computer Sciences*, pages 368–377. IEEE, 1991.
3. O. Kupferman and M. Y. Vardi. Synthesizing distributed systems. In *Logic in Computer Sciences*, pages 389–398, 2001.
4. 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.
5. P. Madhusudan and P.S. Thiagarajan. Distributed controller synthesis for local specifications. In *28th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 2076 of *LNCS*, pages 396–407, 2001.
6. S. Mohalik and I. Walukiewicz. Distributed games. In *Foundations of Software Technology and Theoretical Computer Science*, pages 338–351, 2003.
7. D.E. Muller and P.E. Schupp. Simulating alternating tree automata by nondeterministic automata. *Theoretical Computer Sciences*, 141:67–107, 1995.
8. G.L. Peterson and J.H. Reif. Multiple-person alternation. In *20th Annual IEEE Symposium on Foundations of Computer Sciences*, pages 348–363, october 1979.
9. G.L. Peterson, J.H. Reif, and S. Azhar. Decision algorithms for multiplayer noncooperative games of incomplete information. *Computers and Mathematics with Applications*, 43:179–206, january 2002.
10. Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *IEEE Symposium on Foundations of Computer Science*, pages 746–757, 1990.
11. M.O. Rabin. Decidability of second order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.