Networks of automata with read arcs: a tool for distributed planning

Loïg Jezequel [*] **Éric Fabre** [**]

[*] *ENS Cachan Bretagne, Rennes, France (loig.jezequel@irisa.fr).*
[**] *INRIA Rennes Bretagne Atlantique, Rennes, France (eric.fabre@inria.fr)*

Abstract: A planning problem consists in driving a system from its current state to a set of target states. Problems are generally expressed by a collection of state variables, and actions that change the value of a subset of these variables. Such problems can be modeled as networks of automata, one per state variable, that partially synchronize on some actions. Distributed planning (or factored planning) consists in driving each of these automata to its goal state(s), while preserving the coherence of their interactions. Real planning problems, however, need to model actions that can only be performed in one component when another component is in a specific state. This paper proposes a mechanism to capture this phenomenon, under the form of automata with read arcs, reproducing what already exists for Petri nets. It is shown that a previous approach to distributed planning, based on automata computations, can be extended to this new setting.

Keywords: distributed planning, factored planning, discrete event system, read arcs, concurrent system, distributed constraint solving, formal language theory

1. INTRODUCTION

Planning consists in (optimally) selecting and organizing a set of actions in order to reach a goal state from a given initial state (Nau et al. (2004)). States correspond to tuples $(v_n)_{1\leq n\leq N}$ of values, one per variable V_n, and the actions read and write on subsets of these variables. States and actions naturally define an automaton, and planning amounts to finding a path in this automaton, which is generally a complex search problem due to the number of variables and actions, and to the possibe concurrency of actions. Recently, factored planning approaches have been developed to reduce this complexity: Brafman and Domshlak (2006, 2008); Amir and Engelhardt (2003); Choi and Amir (2006). They consist in solving the problem by parts. The problem is modeled as a network of automata, typically one automaton per state variable V_n. An action involving several variables will be modeled by synchronized transitions in the automata associated to these variables. The problem then becomes finding a plan in each automaton, while ensuring the coherence of these local plans, i.e. the correct use of synchronized transitions.

Fig. 1 illustrates this idea on a simple example with three automata $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$ (from left to right), where goal states are depicted with double circles. Let us first ignore $\mathcal{A}_3$. Transitions labeled α are private to $\mathcal{A}_1$, those labeled β are private to $\mathcal{A}_2$, but γ represents an action that both involves $\mathcal{A}_1$ and $\mathcal{A}_2$. The pair of words (γ, γ) for $(\mathcal{A}_1, \mathcal{A}_2)$ is a valid factored plan: it (synchronously) drives $\mathcal{A}_1$ and $\mathcal{A}_2$ to their goal states. Similarly $(\alpha\alpha, \beta)$ is another valid factored plan, which does not require any interaction. An advantage of factored planning is that the

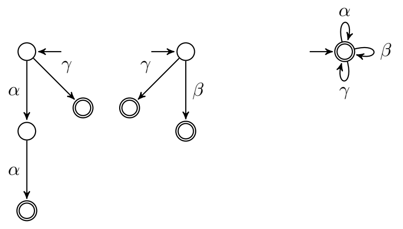

Fig. 1. Network of three partially synchronized automata $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$, from left to right.

concurrency of actions is correctly exploited: the interleaving of local words $\alpha\alpha$ and β is meaningless, and is left unspecified. This means that the three of $\beta\alpha\alpha, \alpha\beta\alpha$ and $\alpha\alpha\beta$ are valid global plans, that actually need not be distinguished. Partially ordered plans are also studied in Hickmott et al. (2007); Bonet et al. (2007).

Several approaches have been proposed to derive such tuples of local plans. Most of them are distributed search methods, that perform a plan search for one state variable, and take the result as a seed for a search on the next state variable, and so on: Brafman and Domshlak (2006, 2008); Amir and Engelhardt (2003); Choi and Amir (2006); Dechter (2003). An alternate approach was proposed in Fabre and Jezequel (2009), and computes all valid factored plans using automata computations. This method can also be refined to compute the best factored plan. In all cases, the computation of a factored plan is based on message exchanges between the variables that may require interactions (*i.e.* synchronized actions): Pearl (1986); Dechter (2003); Fabre (2003).

The work in Fabre and Jezequel (2009) suffers from a weakness however. Planning problems often specify that an action on some variable V_n can only be performed if another variable

[⋆] This work was partly supported by the European Community's 7th Framework Programme under project DISC (DIstributed Supervisory Control of large plants), Grant Agreement INFSO-ICT-224498. This work was also supported by the joint research lab between INRIA and ALU-Bell Labs.

V_m displays some specific value. For example, loading a truck requires the presence of the truck, but it does not change the truck location. Moreover, the presence of the truck may also enable another concurrent action, like filling it up. This ability to read a variable is not encoded by standard automata interactions based on synchronizations. Existing formalisms would require to synchronize the truck with the loading action, then with the filling up action, or conversely. In other words, from the perspective of the truck, it would impose two actions, with two possible orderings, while it is clear that the truck is passive and that only its presence really matters. More formally, consider again the example in Fig. 1, and assume that all actions in $\mathcal{A}_1$ and $\mathcal{A}_2$ can only be performed if $\mathcal{A}_3$ is in its initial (and final) state. To model this, one needs to introduce loops in $\mathcal{A}_3$ that synchronize with α, β and γ. As a consequence, for $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$ to jointly reach their goal, $\mathcal{A}_3$ must perform one word in $\{\gamma, \alpha\alpha\beta, \alpha\beta\alpha, \beta\alpha\alpha\}$, which forces readings of its state variable to be displayed, counted and ordered. In particular, concurrent (simultaneous) readings become impossible. While clearly there is no need of any "action plan" for $\mathcal{A}_3$.

This paper proposes a mechanism that solves this difficulty: it extends the semantics of automata interactions to enable state readings, without forcing a component to fire a transition in order to display its state. In the previous example, this will allow $\mathcal{A}_3$ to stay idle and simply "show" its state in order to enable actions in $\mathcal{A}_1$ and $\mathcal{A}_2$. This mechanism can thus be seen as the extension to automata of the read arcs of Petri nets, that simply check for the presence of tokens in some places to enable a transition, but that do not consume these tokens if the transition fires: see for ex. Baldan et al. (2001). The next section illustrates the principle of this construction in the simplest setting, while Section 3 extends it to the case of networks of automata, with cross and simultaneous readings. Section 4 proves that this setting enjoys the right algebraic properties that enable the distributed computation of factored plans, and thus allows one to recycle the algorithms presented in Fabre and Jezequel (2009).

2. SIMPLE READING MECHANISM

This section illustrates how a reading mechanism can be introduced to define automata interactions. The setting is first limited to the simple case of two automata. The next section extends it to an arbitrary number of automata, with cross and multiple readings.

2.1 Notation

Let us first recall standard notions. An automaton $\mathcal{A}$ is a tuple $\mathcal{A} = (S, S^I, S^F, \Sigma, T)$ where S is a finite set of states, S^I, S^F are subsets of S representing initial and final states, Σ is a finite alphabet of transition labels and $T \subseteq S \times \Sigma \times S$ denotes the transition set. A path in $\mathcal{A}$ is a sequence $\pi = t_1...t_K$ of transitions such that $t_k = (s_{k-1}, \sigma_k, s_k)$, $1 \leq k \leq K$. We adopt notation $s_0 = s^-(\pi)$ and $s_K = s^+(\pi)$. Path π is accepted by $\mathcal{A}$, iff $s^-(\pi) \in S^I$ and $s^+(\pi) \in S^F$. The word associated to π is the label sequence $\Sigma(\pi) = \sigma_1...\sigma_K$. The language of $\mathcal{A}$, denoted by $\mathcal{L}(\mathcal{A})$, is the set of words produced by all paths accepted by $\mathcal{A}$.

2.2 Writing and reading

Beyond the simple composition of two automata $\mathcal{A}_1$ and $\mathcal{A}_2$ by the usual synchronous product, one would like to allow a weak form of synchronization, where $\mathcal{A}_2$ is allowed to perform some of its transitions only when $\mathcal{A}_1$ is in specific states. This requires a double mechanism. First, $\mathcal{A}_1$ should display properties of its states. Rather than associating labels to states, it is equivalently assumed that transitions *output* a readable label. So let us define $\mathcal{A}_1 = (S_1, S_1^I, S_1^F, \Sigma_1 \times O_1, T_1)$ where O_1 is a finite set of readable labels, and so $T_1 \subseteq S_1 \times \Sigma_1 \times O_1 \times S_1$. For simplicity, in this paper a single state property is displayed. The language $\mathcal{L}(\mathcal{A}_1)$ is defined as above by considering $\Sigma_1 \times O_1$ as alphabet. Secondly, $\mathcal{A}_2$ must be able to read these values. Let I_2 be the set of "input values" to $\mathcal{A}_2$, that one can define as $\mathcal{A}_2 = (S_2, S_2^I, S_2^F, I_2 \times \Sigma_2, T_2)$, with $\star \in I_2$, and so $T_2 \subseteq S_2 \times I_2 \times \Sigma_2 \times S_2$. The semantics is that a transition $t_2 = (s_2, a, \sigma, s_2') \in T_2$ will be able to fire only if $\mathcal{A}_1$ has displayed the value a at the output of one of its transitions; the special case of $a = \star$ means that t_2 is not reading in $\mathcal{A}_1$. Again, the language $\mathcal{L}(\mathcal{A}_2)$ is obtained by considering $I_2 \times \Sigma_2$ as alphabet.

To define the interactions of $\mathcal{A}_1$ and $\mathcal{A}_2$ based on this reading mechanism, let us first consider the composition of their languages. It consists of words over the alphabet $I_2 \times \Sigma \times O_1$ where $\Sigma = \Sigma_1 \cup \Sigma_2$. A word $w = (i_1, \sigma_1, o_1)...(i_K, \sigma_K, o_K)$ is *coherent* iff, for $1 \leq k \leq K$, one has

(i) $i_k = o_{k-1}$ or $i_k = \star$ (in particular $i_1 = \star$),
(ii) $o_k = o_{k-1}$ if $\sigma_k \notin \Sigma_1$, and
(iii) $i_k = \star$ if $\sigma_k \notin \Sigma_2$.

In other words, (i) expresses that if label σ_k is attached to a reading, the previous label must have provided this value as output. Condition (ii) expresses that labels of $\Sigma_2 \setminus \Sigma_1$, which correspond to private transitions of $\mathcal{A}_2$, can not change (or must propagate) the output produced by $\mathcal{A}_1$. Symmetrically, (iii) expresses that private transitions of $\mathcal{A}_1$ can not be associated to a reading.

2.3 Operations on languages

The *projection* Π_2 of words over $I_2 \times \Sigma \times O_1$ on $I_2 \times \Sigma_2$ is defined as the monoid morphism generated by $\Pi_2(i, \sigma, o) = (i, \sigma)$ if $\sigma \in \Sigma_2$, otherwise $\Pi_2(i, \sigma, o) = \epsilon$, where ϵ denotes the empty word as usual. Similarly, the projection Π_1 on $\Sigma_1 \times O_1$ is the monoid morphism generated generated by $\Pi_1(i, \sigma, o) = (\sigma, o)$ if $\sigma \in \Sigma_1$, otherwise $\Pi_1(i, \sigma, o) = \epsilon$.

The *composition* or *product* of languages $\mathcal{L}(\mathcal{A}_1) \times_L \mathcal{L}(\mathcal{A}_2)$ is defined by all coherent words w over alphabet $I_2 \times \Sigma \times O_1$ such that $\Pi_1(w) \in \mathcal{L}(\mathcal{A}_1)$ and $\Pi_2(w) \in \mathcal{L}(\mathcal{A}_2)$. Notice that this definition extends the natural synchronous product of languages, where a letter $\sigma \in \Sigma_1 \cap \Sigma_2$ corresponds to synchronous actions of two languages. Here, "letters" (i, σ) and (σ, o) in $\mathcal{L}(\mathcal{A}_2)$ and $\mathcal{L}(\mathcal{A}_1)$ respectively give rise to the synchronous action (i, σ, o) that both needs input i and produces output o.

Example. Let $\mathcal{L}(\mathcal{A}_1) = \{w_1 = (\alpha, 1)(\gamma, 2)(\alpha, 3)(\delta, 4)\}$ and $\mathcal{L}(\mathcal{A}_2) = \{w_2 = (1, \beta)(\star, \gamma)(\star, \beta)(3, \delta), w_2' = (1, \gamma)(1, \beta)\}$, with $\Sigma_1 = \{\alpha, \gamma, \delta\}, \Sigma_2 = \{\beta, \gamma, \delta\}$ and $I = O = \{1, ..., 4\}$. One has $\mathcal{L}(\mathcal{A}_1) \times_L \mathcal{L}(\mathcal{A}_2) = \{\, w,\ w' \,\}$ (Fig. 2) where

$$w = (\star, \alpha, 1)(1, \beta, 1)(\star, \gamma, 2)(\star, \beta, 2)(\star, \alpha, 3)(3, \delta, 4)$$
$$w' = (\star, \alpha, 1)(1, \beta, 1)(\star, \gamma, 2)(\star, \alpha, 3)(\star, \beta, 3)(3, \delta, 4)$$

Observe that w_2' does not synchronize with w_1, due to the impossibility of reading again 1 after action γ has been performed in $\mathcal{A}_1$. By contrast, words w_1 and w_2 synchronize in two different ways, yielding w and w'. Notice that the readings in $\mathcal{A}_2$ constrain the interleaving of the private events of $\mathcal{A}_1$ and $\mathcal{A}_2$ (see the notion of asymmetric conflict in Petri nets with read arcs in Baldan et al. (2001, 2004)). For example, the first occurrence of β has to be performed after first occurrence of α, while both are private. By contrast, since there is no reading in the second occurrence of β, it can be performed either before or after the second occurrence of α. Notice as well that writings in $\mathcal{A}_1$ are propagated along product words by private events of $\mathcal{A}_2$, which are attached to a "stuttering event" of $\mathcal{A}_1$ (circles in Fig. 2). Finally, observe that the usual synchronous product of languages is obtained by positioning all readings to $\star$.

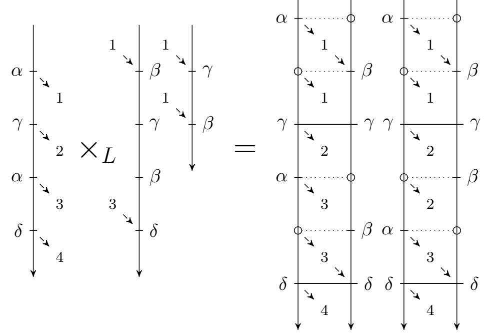

Fig. 2. Product of languages. Arrows denote output and input values. Product words, on the right, are depicted as synchronized threads where circles denote stuttering events.

2.4 Operations on automata

In standard automata theory, one has that $\mathcal{L}(\mathcal{A}_1)\dot{\times}_L\mathcal{L}(\mathcal{A}_2)$ is the language $\mathcal{L}(\mathcal{A}_1 \times \mathcal{A}_2)$, where $\mathcal{A}_1 \times \mathcal{A}_2$ is the synchronous product of automata and $\mathcal{L}(\mathcal{A}_1)\dot{\times}_L\mathcal{L}(\mathcal{A}_2)$ the usual synchronous product of languages defined as $\Pi_1^{-1}(\mathcal{L}(\mathcal{A}_1)) \cap \Pi_2^{-1}(\mathcal{L}(\mathcal{A}_2))$. This property is essential to replace operations on regular languages, which are possibly infinite objects, by operations on their representations as automata, which are finite objects. To extend this property to the writing and reading automata described above, one needs a notion of automaton with *internal* readings and writings.

Let $\mathcal{A} = (S, S^I, S^F, I \times \Sigma \times O, T)$ be an automaton with (internal) inputs and outputs, assuming $\star \in I$. A path $\pi = t_1...t_K$ in $\mathcal{A}$ is said to be *coherent* iff its label sequence $(I \times \Sigma \times O)(\pi)$ forms a coherent word over alphabet $I \times \Sigma \times O$. Coherence here only involves condition (i) because $\Sigma_1 = \Sigma_2 = \Sigma$. The coherent language of $\mathcal{A}$, denoted $\mathcal{L}_c(\mathcal{A})$, is now defined as the restriction of $\mathcal{L}(\mathcal{A})$ to its coherent words.

The product is now defined as $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2$ iff $S = S_1 \times S_2, S^I = S_1^I \times S_2^I, S^F = S_1^F \times S_2^F, I = I_2, \Sigma = \Sigma_1 \cup \Sigma_2, O = O_1$, and the transition set $T = T_s \cup T_{1,p} \cup T_{2,p}$ is defined by

$$T_s = \{ ((s_1,s_2), i_2, \sigma, o_1, (s_1', s_2')) : (s_1, \sigma, o_1, s_1') \in T_1,\ (s_2, i_2, \sigma, s_2') \in T_2 \} \quad (1)$$

$$T_{1,p} = \{ ((s_1,s_2), \star, \sigma_1, o_1, (s_1', s_2)) : \sigma_1 \notin \Sigma_2, (s_1, \sigma_1, o_1, s_1') \in T_1,\ s_2 \in S_2 \} \quad (2)$$

$$T_{2,p} = \{ ((s_1,s_2), i_2, \sigma_2, o_1, (s_1, s_2')) : \sigma_2 \notin \Sigma_1, (s_2, i_2, \sigma_2, s_2') \in T_2,\ s_1 \in S_1,\ o_1 \in O_1 \} \quad (3)$$

Synchronized transitions in T_s correspond of course to $\sigma \in \Sigma_1 \cap \Sigma_2$, private transitions of $\mathcal{A}_1$ appear with no reading in $T_{1,p}$, while private transitions of $\mathcal{A}_2$ reproduce all possible outputs of $\mathcal{A}_1$ in $T_{2,p}$.

Lemma 1. One has $\mathcal{L}_c(\mathcal{A}_1 \times \mathcal{A}_2) = \mathcal{L}(\mathcal{A}_1) \times_L \mathcal{L}(\mathcal{A}_2)$.

One can define as well a notion of projection on automata, such that "$\mathcal{L}(\Pi(\mathcal{A})) = \Pi(\mathcal{L}(\mathcal{A}))$". This is examined later, in a more general setting.

3. NETWORKS OF AUTOMATA WITH READ ARCS

To extend read arcs to networks of automata, several refinements are necessary, and in particular one needs a mechanism to specify where, that is in which component, input values must be read. This relies on the notion of *tag*.

3.1 Reading and writing tags

Consider a finite set O of output labels partitioned into $O = O_1 \uplus ... \uplus O_N$. In the sequel, each O_n will characterize the possible values displayed by component $\mathcal{A}_n$ in a network of automata with read arcs composed of $\mathcal{A}_1, ..., \mathcal{A}_N$.

A *tag* over O is a function $\alpha : \{1, ..., N\} \to O \cup \{\star\}$ such that $\forall n, \alpha(n) \in O_n \cup \{\star\}$. The *support* of α is $Supp(\alpha) = \{n : \alpha(n) \neq \star\}$, and for $J \subseteq \{1, ..., N\}$, $\mathcal{T}_J$ denotes the set of tags whose support is J, $\mathcal{T}_{\subseteq J}$ denotes the set of tags whose support is included in J, and $\mathcal{T}$ denotes the set of all tags.

Two tags α and β are *compatible*, denoted by $\alpha \sim \beta$, iff they coincide on $Supp(\alpha) \cap Supp(\beta)$. The *composition* $\alpha \wedge \beta$ is defined by $(\alpha \wedge \beta)(n) = \alpha(n)$ if $\alpha(n) \neq \star$, otherwise $\beta(n)$. Notice that $Supp(\alpha \wedge \beta) = Supp(\alpha) \cup Supp(\beta)$, and $\alpha \wedge \beta = \beta \wedge \alpha$ when $\alpha \sim \beta$. Composition will only be applied in this case. For $J \subseteq \{1, ..., N\}$, the *restriction* of tag α to J, denoted by $\alpha_{|J}$, is defined by $\alpha_{|J}(n) = \alpha(n)$ for $n \in J$, otherwise $\alpha_{|J}(n) = \star$.

3.2 Automata with read arcs

We assume a partition $O = O_1 \uplus ... \uplus O_N$ given once for all.

An *automaton with read arcs* (ARA) $\mathcal{A}$ on O is defined as the tuple $\mathcal{A} = (S, S^I, S^F, \mathcal{T} \times \Sigma \times \mathcal{T}_W, T, W)$, where $(S, S^I, S^F, \mathcal{T} \times \Sigma \times \mathcal{T}_W, T)$ is an ordinary automaton, and the *writing set* $\emptyset \neq W \subseteq \{1, ..., N\}$ defines the indices of output sets O_n displayed by $\mathcal{A}$. For technical reasons (namely the definition of products below) we also require that Σ contains the special label *start*, used to label each initial transition: $\forall t \in T, s^-(t) \in S^I \Rightarrow \Sigma(t) = start$.

A transition $t = (s, \alpha, \sigma, \beta, s') \in T$ moves from s to s' in $\mathcal{A}$ when action σ is performed, assuming t manages to read the values specified by the *reading tag* $\alpha \in \mathcal{T}$. As a result of the firing, t produces the *writing tag* $\beta \in \mathcal{T}_W$, which means that it changes the output values in each O_n, for $n \in W$. Intuitively, transition t changes the state property displayed by each component $\mathcal{A}_n$ for $n \in W$. This is made clear by the semantics of an ARA, and by the composition operations defined below. The *reading set* R of $\mathcal{A}$ is defined as the smallest subset of $\{1, ..., N\}$ such that $T \subseteq S \times \mathcal{T}_{\subseteq R} \times \Sigma \times \mathcal{T}_W \times S$, i.e. such that reading tags of transitions all have their support in R. When needed, R can be appended at the end of the tuple defining $\mathcal{A}$.

Let $\pi = t_1...t_K$ be a path in $\mathcal{A}$, and $(\mathcal{T} \times \Sigma \times \mathcal{T}_W)(\pi) = (\alpha_1, \sigma_1, \beta_1)...(\alpha_K, \sigma_K, \beta_K)$ its associated word. This word is *coherent over* W iff

$$\forall\, 2 \leq k \leq K,\ (\alpha_k)_{|W} \sim (\beta_{k-1})_{|W} \qquad (4)$$

In other words, readings and writings along this path must be consistent for every component O_n of the reading and writing tags, for $n \in W$. Given that $Supp(\beta_{k-1}) = W$, (4) reproduces condition (i) of Section 2, for every $n \in W$. The readings performed outside W by transitions of $\mathcal{A}$ are not considered. The *coherent language* of $\mathcal{A}$, denoted $\mathcal{L}_c(\mathcal{A})$, is defined as the subset of words in $\mathcal{L}(\mathcal{A})$ that are coherent over its writing set W.

Notice that the above definitions of ARA and their semantics are simply the extension of the automata with inputs and outputs of Section 2 to the case of vector readings and writings.

A *network of automata with read arcs* is defined as an N-uple $(\mathcal{A}_1, ..., \mathcal{A}_N)$ of ARA such that $\mathcal{A}_n = (S_n, S_n^I, S_n^F, \mathcal{T}_{\subseteq R_n} \times \Sigma_n \times \mathcal{T}_{\{n\}}, T_n, \{n\}, R_n)$, $1 \leq n \leq N$. Each *component* $\mathcal{A}_n$ is thus in charge of displaying values in its private set O_n of state properties, by means of the writing tags in $\mathcal{T}_{\{n\}}$ attached to transitions. Notice that $\mathcal{A}_n$ may also read values in its own O_n by the reading tags in $\mathcal{T}_{\subseteq R_n}$, which is somehow redundant with (but not equivalent to) reading its own internal state to enable the firing of a transition. The interest of this phenomenon becomes clear in the definition of the composition of ARA, since it allows cross-readings. It is also of crucial importance for working with languages, regardless of the actual automata that produced them.

3.3 Operations on languages

Weak inclusion. Let $\mathcal{L}, \mathcal{L}'$ be two languages over $\mathcal{T}_R \times \Sigma \times \mathcal{T}_W$. We write $\mathcal{L} \sqsubseteq \mathcal{L}'$ iff $\forall w = (\alpha_1, \sigma_1, \beta_1)...(\alpha_K, \sigma_K, \beta_K) \in \mathcal{L}$ there exists a word $w' = (\alpha'_1, \sigma_1, \beta_1)...(\alpha'_K, \sigma_K, \beta_K) \in \mathcal{L}'$ such that α'_k is a restriction of α_k, $1 \leq k \leq K$. In other words, w' is identical to w up to reading tags, that may be less specific. We denote it by $w \sqsubseteq w'$, with a light abuse of notation.

Projection. Consider letter $(\alpha, \sigma, \beta) \in \mathcal{T} \times \Sigma \times \mathcal{T}_W$, and let R', Σ', W' be respectively reading, label and writing sets. Projection $\Pi_{R',\Sigma',W'}$ is defined on letters by $\Pi_{R',\Sigma',W'}(\alpha, \sigma, \beta) = (\alpha_{|R'}, \sigma, \beta_{|W'})$ if $\sigma \in \Sigma'$, otherwise $\Pi_{R',\Sigma',W'}(\alpha, \sigma, \beta) = \epsilon$. Projection is then extended to words in $(\mathcal{T} \times \Sigma \times \mathcal{T}_W)^*$ as the induced monoid morphism, and to languages by union. Notice that if $\mathcal{L}$ is a coherent language over W, its projection $\Pi_{R',\Sigma',W'}(\mathcal{L})$ may lose coherence, due to the missing writing tags attached to letters $\sigma \notin \Sigma'$. Remark: for $w = (\alpha_1, \sigma_1, \beta_1)...(\alpha_K, \sigma_K, \beta_K)$, the definition of $\Pi_{R',\Sigma',W'}$ as a monoid morphism allows one to associate uniquely any letter $(\alpha_k, \sigma_k, \beta_k)$ of w to its image in $\Pi_{R',\Sigma',W'}(w)$, when $\sigma_k \in \Sigma'$.

Product. For $i = 1, 2$, let $w_i \in (\mathcal{T}_{\subseteq R_i} \times \Sigma_i \times \mathcal{T}_{W_i})^*$ be a coherent word over W_i. The product $w_1 \times_L w_2$ is defined as the set of words $w \in (\mathcal{T}_{\subseteq R} \times \Sigma \times \mathcal{T}_W)^*$ that are coherent over W, with $R = R_1 \cup R_2$, $\Sigma = \Sigma_1 \cup \Sigma_2$ and $W = W_1 \cup W_2$, and such that

(I) $\Pi_{R_i,\Sigma_i,W_i}(w) \sqsubseteq w_i$, for $i = 1, 2$

(II) for any letter (α, σ, β) along w

  (a) if $\sigma \in \Sigma_1 \cap \Sigma_2$, let $(\alpha_{1,k}, \sigma, \beta_{1,k})$ and $(\alpha_{2,l}, \sigma, \beta_{2,l})$ be the projected images of (α, σ, β) in w_1 and w_2, resp., then $\alpha = \alpha_{1,k} \wedge \alpha_{2,l}$ and $\beta = \beta_{1,k} \wedge \beta_{2,l}$.

  (b) if $\sigma \notin \Sigma_2$, let $(\alpha', \sigma', \beta')$ be the last letter before (α, σ, β) in w such that $\sigma' \in \Sigma_2$; let $(\alpha_{1,k}, \sigma, \beta_{1,k})$ be the image of (α, σ, β) in w_1, and $(\alpha'_{2,l}, \sigma', \beta'_{2,l})$ the image of $(\alpha', \sigma', \beta')$ in w_2, then $\alpha = \alpha_{1,k} \wedge (\beta'_{2,l})_{|W_2 \setminus W_1}$ and $\beta = \beta_{1,k} \wedge (\beta'_{2,l})_{|W_2 \setminus W_1}$.

  (c) if $\sigma \notin \Sigma_1$, symmetric conditions of (b).

Condition (I) ensures that w reproduces the labels of w_1 and w_2, as in a standard product of languages, and that it reproduces also their writings, and at least their readings. But the readings in w could be much stronger: they could erase more $\star$ than strictly necessary, i.e. require more values than those specified in w_1 and w_2. So (II) ensures that the readings of letters in w combine exactly those required by the associated letters in w_1 and w_2, and not more. Specifically, (II.a) combines the reading and writing tags of the two image letters in w_1, w_2; the compatibility of these tags is guaranteed by (I). In (II.b), a private event of w_1 has to be reflected in w. Its reading and writing tags are thus augmented to push forward the output values previously positioned by $(\alpha'_{2,l}, \sigma', \beta'_{2,l})$ on $W_2 \setminus W_1$. Indeed, observe that the image of (α, σ, β) in w_2 is an epsilon, "immediately preceded" by $(\alpha'_{2,l}, \sigma', \beta'_{2,l})$. Notice also that $\alpha_{1,k} \sim \beta'_{2,l}$, thanks again to (I), so $\alpha = \alpha_{1,k} \wedge (\beta'_{2,l})_{|W_2 \setminus W_1} = \alpha_{1,k} \wedge \beta'_{2,l}$.

Notice that conditions (I,II) above can easily be turned into a recursive construction of an element in $w_1 \times_L w_2$, that would interleave letters of these two words, provided the coherence of reading and writing tags is checked before each composition, in order to ensure the coherence of the resulting w over W.

The definition of the product of two words naturally extends to languages by union. It is also clearly associative.

Lemma 2. Let L_i be a language over $\mathcal{T}_{\subseteq R_i} \times \Sigma_i \times \mathcal{T}_{W_i}$, $i = 1, 2$. Then $\Pi_{R_i,\Sigma_i,W_i}(\mathcal{L}_1 \times_L \mathcal{L}_2)$ is a coherent language over W_i, and satisfies

$$\Pi_{R_i,\Sigma_i,W_i}(\mathcal{L}_1 \times_L \mathcal{L}_2) \sqsubseteq \mathcal{L}_i.$$

Notice that the product $\mathcal{L}_1 \times_L \mathcal{L}_2$ removes more words in each $\mathcal{L}_i$ than the usual synchronous product (that it actually reinforces). This is due to the necessity of checking more properties in pairs of words w_1 and w_2 that are combined, namely the coherence of their cross readings and common writings over $W_1 \cap W_2$. Notice as well that $\Pi_{R_i,\Sigma_i,W_i}(w_1 \times_L w_2)$ is not w_i in general, but a set of versions of w_i where reading tags are reinforced. This is due to readings inherited from the letters of the other word w_j with which they may synchronize.

3.4 Product of automata with read arcs

Consider $\mathcal{A}_i = (S_i, S_i^I, S_i^F, \mathcal{T} \times \Sigma_i \times \mathcal{T}_{W_i}, T_i, W_i)$, $i = 1, 2$. The *product* $\mathcal{A}_1 \times \mathcal{A}_2$ is defined by $(S, S^I, S^F, \mathcal{T} \times \Sigma \times \mathcal{T}_W, T, W)$ where $S = S_1 \times S_2, S^I = S_1^I \times S_2^I, S^F = S_1^F \times S_2^F, \Sigma = \Sigma_1 \cup \Sigma_2, W = W_1 \cup W_2$, and the transition set $T = T_s \cup T_{1,p} \cup T_{2,p}$ is given by

$$\begin{aligned} T_s = \{\, & ((s_1, s_2), \alpha_1 \wedge \alpha_2, \sigma, \beta_1 \wedge \beta_2, (s'_1, s'_2)) : \\ & (s_i, \alpha_i, \sigma, \beta_i, s'_i) \in T_i,\ \alpha_1 \sim \alpha_2,\ \beta_1 \sim \beta_2 \,\} \end{aligned} \qquad (5)$$

$$
\begin{aligned}
T_{1,p} = \{\, &((s_1,s_2),\alpha_1\wedge\beta_2,\sigma_1,\beta_1\wedge\beta_2,(s_1',s_2)) : \\
&\sigma_1\notin\Sigma_2,\ (s_1,\alpha_1,\sigma_1,\beta_1,s_1')\in T_1,\ s_2\in S_2, \\
&\beta_2\in\mathcal{T}_{W_2\setminus W_1},\ \alpha_1\sim\beta_2\,\} \qquad (6)\\
T_{2,p} = \{\, &((s_1,s_2),\alpha_2\wedge\beta_1,\sigma_2,\beta_1\wedge\beta_2,(s_1,s_2')) : \\
&\sigma_2\notin\Sigma_1,\ (s_2,\alpha_2,\sigma_2,\beta_2,s_2')\in T_2,\ s_1\in S_1, \\
&\beta_1\in\mathcal{T}_{W_1\setminus W_2},\ \alpha_2\sim\beta_1\,\} \qquad (7)
\end{aligned}
$$

(5) merges/synchronizes transitions carrying shared labels, provided they agree on readings and writings. (6) extends private transitions of $\mathcal{A}_1$ to all possible values of s_2, and to all possible values of tags over $W_2\setminus W_1$ that are used both for reading and writing, *i.e.* for the propagation of the output values set by $\mathcal{A}_2$ over $W_2\setminus W_1$. (7) does the symmetric operation for private transitions of $\mathcal{A}_2$.

Not all transitions defined above are accessible and coaccessible in $\mathcal{A}_1\times\mathcal{A}_2$. Therefore some trimming should be performed to remove useless transitions, taking into account that (co-) accessibility here must incorporate the coherence over W.

Theorem 3. $\mathcal{L}_c(\mathcal{A}_1\times\mathcal{A}_2)=\mathcal{L}_c(\mathcal{A}_1)\times_L\mathcal{L}_c(\mathcal{A}_2)$

Theorem 3 can be plugged into Lemma 2, which reveals that $\Pi_{R_1,\Sigma_1,W_1}(\mathcal{L}_c(\mathcal{A}_1\times\mathcal{A}_2))\sqsubseteq\mathcal{L}_c(\mathcal{A}_1)$ (and symmetrically for $\mathcal{A}_2$). So the synchronization of $\mathcal{A}_1$ with $\mathcal{A}_2$ removes words w_1 of $\mathcal{A}_1$ that have no compatible companion w_2 in $\mathcal{A}_2$, both for the firing of common labels in $\Sigma_1\cap\Sigma_2$ and for the compatibility of readings and writings. And for words w_1 of $\mathcal{A}_1$ that are preserved, their writing tags are unchanged, but their reading tags may inherit extra conditions from their companion w_2 in $\mathcal{A}_2$.

4. DISTRIBUTED PLANNING IN NETWORKS OF ARA

As mentioned in the introduction, a planning problem can be modeled as a network of automata with read arcs. Given $(\mathcal{A}_1,...,\mathcal{A}_N)$, a global plan is simply a (coherent) word w in $\mathcal{L}_c(\mathcal{A}_1\times...\times\mathcal{A}_N)$, i.e. a sequence of events reaching a marked or final state from the/an initial state, ensuring both that synchronized actions are performed correctly, and that readings and writings in each component $\mathcal{A}_n$ are also performed correctly. Factored planning would consist in deriving w under a factored form, i.e. as a tuple $(w_1,...,w_N)$ of words, with $w_n=\Pi_{R_n,\Sigma_n,W_n}(w)\in\Pi_{R_n,\Sigma_n,W_n}(\mathcal{L}_c(\mathcal{A}_1\times...\times\mathcal{A}_N))\sqsubseteq\mathcal{L}_c(\mathcal{A}_n)$. Distributed planning goes a little further in this direction, and requires that the w_i be computed *without computing* w, since working with $\mathcal{A}_1\times...\times\mathcal{A}_N$ might be intractable. Surprisingly, this can be achieved if component interactions are sparse enough, as it was already proved in Fabre and Jezequel (2009). The idea is that the projected languages $\Pi_{R_n,\Sigma_n,W_n}(\mathcal{L}_c(\mathcal{A}_1\times...\times\mathcal{A}_N))$ can be derived without computing first $\mathcal{L}_c(\mathcal{A}_1\times...\times\mathcal{A}_N)$, by means of local and coordinated computations. The same procedure can be used as well to select a factored plan $(w_1,...,w_N)$ in these projections. The central tool to achieve this is a relation between product and projection on languages, that we immediately translate into a relation between product and projection of ARA, in order to handle finite objects.

4.1 Projection of an ARA

The *projection* of an automaton with read arcs $\mathcal{A}=(S,S^I,S^F,\mathcal{T}\times\Sigma\times\mathcal{T}_W,T,W)$ on R',Σ',W', respectively reading, label and writing sets, is defined as the ARA $\Pi_{R',\Sigma',W'}(\mathcal{A})=(S,S^I,S'^F,\mathcal{T}_{\subseteq R'}\times\Sigma'\times\mathcal{T}_{W'},T',W')$. Transitions are defined from a (possibly empty) silent path π, i.e. labeled by $\Sigma\setminus\Sigma'$, followed by a visible transition, i.e. labeled by Σ'.

$$
\begin{aligned}
T'=\{\,&(s,\alpha_{|R'},\sigma,\beta_{|W'},s')\ :\ \sigma\in\Sigma',\\
&\exists(s'',\alpha,\sigma,\beta,s')\in T,\\
&\exists\pi \text{ a path in } \mathcal{A},\Sigma(\pi)\in(\Sigma\setminus\Sigma')^*,\\
&s^-(\pi)=s,\ s^+(\pi)=s''\}.
\end{aligned}
$$

The fact that only $\alpha_{|R'}$ is considered requires a comment. In the sequel, R',Σ',W' will be the reading, label and writing sets of some automaton, and thus will select the transitions of this automaton. So any discarded transition t' of path π will be a private transition of another automaton, which can not modify the values necessary to $\alpha_{|W'}$. So $(s,\alpha_{|R'},\sigma,\beta_{|W'},s')$ will remain firable in the projection, since coherence is only tested over W'. Finally, the set S'^F is also constructed from paths of transitions from $\Sigma\setminus\Sigma'$: $S'^F=S^F\cup\{s\in S\ :\ \exists\pi \text{ a path in } \mathcal{A},\Sigma(\pi)\in(\Sigma\setminus\Sigma')^*,\ s^-(\pi)=s,\ s^+(\pi)\in S^F\}$. As for the product, this definition may produce states and transitions that are not accessible and coaccessible, so a trimming may be necessary.

The construction above corresponds to an ε-reduction: useless transitions are bypassed. It generally results in non-deterministic automata. Standard determinization and minimization procedures can be performed (see for example Cassandras and Lafortune (1999)): as they preserve the language over alphabet $\mathcal{T}_{\subseteq R'}\times\Sigma'\times\mathcal{T}_{W'}$, they also preserve the coherent language. Determinization may however incur an exponential blowup in the number of states, in the worst case, although in practice this is rarely the case and one observes a reduction in size (at least for the planning problem benchmarks that we have considered). See also Mohri (1997), Section 4, witnessing size reductions performed by the determinization of automata related to speech processing benchmarks.

The projection of an ARA is related to the projection of its language provided an extra structural property is satisfied. The ARA $\mathcal{A}=(S,S^I,S^F,\mathcal{T}\times\Sigma\times\mathcal{T}_W,T,W)$ is *state labelled* iff for any pair t_1,t_2 of transitions arriving at the same state s, *i.e.* $t_i=(s_i,\alpha_i,\sigma_i,\beta_i,s)$, one has $\beta_1=\beta_2$. In other words, the output label of a transition only depends on its final state, and thus can be attached to the latter. This is clearly true for planning problems, or it can be made true with a minimal transform. The property of being state labelled is preserved by standard operations on automata (product, projection, minimization,...).

In the sequel, we denote by $\Pi_{\mathcal{A}_i}$ the projection Π_{R_i,Σ_i,W_i}, where $\mathcal{A}_i=(S_i,S_i^I,S_i^F,\mathcal{T}\times\Sigma_i\times\mathcal{T}_{W_i},T_i,W_i,R_i)$, and R_i is the writing set of $\mathcal{A}_i$.

Theorem 4. Let $\mathcal{A}_1,\mathcal{A}_2$ be state labelled ARA, then $\mathcal{L}_c(\Pi_{\mathcal{A}_i}(\mathcal{A}_1\times\mathcal{A}_2))=\Pi_{\mathcal{A}_i}(\mathcal{L}_c(\mathcal{A}_1\times\mathcal{A}_2))$

The importance of being state labelled comes from the fact that the coherence of the words produced by $\mathcal{A}_1\times\mathcal{A}_2$ is only checked when taking the language. Taking the projection $\Pi_{\mathcal{A}_i}(\mathcal{A}_1\times\mathcal{A}_2)$ first can thus erase some of the coherence conditions, unless if they are stucturally reflected in the automata.

As explained below, it is never requested to use projections that are not of the form $\Pi_{\mathcal{A}_i}(\mathcal{A}_1\times\mathcal{A}_2)$. In consequence, Theorems 3 and 4 allow one to work directly on automata instead of languages, which is required in practice as automata are finite objects while languages may be infinite.

4.2 Central relation between product and projection

This part presents the key result that motivated the above construction. It enables the computation of a factored plan by distributed computations, of lower complexity, as soon as component interactions are sparse enough.

Let $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$ have disjoint writing sets W_i, $\mathcal{A}_2$ *separates* $\mathcal{A}_1$ and $\mathcal{A}_3$ iff:

(1) $\Sigma_3 \cap \Sigma_1 \subseteq \Sigma_2$ (separation of actions),
(2) $R_3 \cap W_1 \subseteq R_2$, and $\forall\, t_3 = (s, \alpha, \sigma, \beta, s') \in T_3$, $\alpha_{|W_1} \neq \star \;\Rightarrow\; \sigma \in \Sigma_2$ (propagation of readings),
(3) and symmetrically, by inverting indexes 1 and 3.

Condition (2) expresses that any reading performed by $\mathcal{A}_3$ inside $\mathcal{A}_1$ corresponds to an action σ that is shared with $\mathcal{A}_2$. So in the product $\mathcal{A}_3 \times \mathcal{A}_2$, the latter will inherit the readings in $\mathcal{A}_1$ that are necessary to $\mathcal{A}_3$. This induces the following property

Theorem 5. If $\mathcal{A}_2$ separates $\mathcal{A}_1$ from $\mathcal{A}_3$, then

$$\Pi_{\mathcal{A}_1}(\mathcal{A}_1 \times \mathcal{A}_2 \times \mathcal{A}_3) = \Pi_{\mathcal{A}_1}(\mathcal{A}_1 \times \Pi_{\mathcal{A}_2}(\mathcal{A}_2 \times \mathcal{A}_3)).$$

The practical consequence of this result is that $\Pi_{\mathcal{A}_1}(\mathcal{A})$, which language describes the local plans of $\mathcal{A}_1$, can be derived without computing first the big product $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \times \mathcal{A}_3$, which language describes all global plans. Indeed, it suffices to combine $\mathcal{A}_1$ with the smaller *message* $\Pi_{\mathcal{A}_2}(\mathcal{A}_2 \times \mathcal{A}_3)$ from $\mathcal{A}_2$, and project the result on $\mathcal{A}_1$. Similar ideas allow one to derive as well all the local plan descriptions $\Pi_{\mathcal{A}_i}(\mathcal{A})$. We refer the reader to an forthcoming longer version of this work for a detailed application example of Theorem 5, or to Fabre and Jezequel (2009) for an illustration in a slightly different setting.

5. CONCLUSION

We have proposed a new notion of automaton with read arcs, to capture weak or asymmetric interactions between systems. This feature is demanded by most planning problems. It was also shown that a distributed planning approach previously derived for networks of weighted automata still works in this new setting.

Automata with read arcs relate to the model of asynchronous automata presented in Zielonka (1995). The latter are discrete event systems where actions read and write in some registers, with the property that writings are private, like in our case. Asynchronous automata capture concurrency (when two actions operate on different registers) and causality. They generate words that are partially ordered, i.e. traces. This phenomenon appears as well in networks of ARA: events in a single ARA are totally ordered, but a factored plan $(w_1, ..., w_N)$ is a tuple of partially synchronized sequences, and thus a partial order. In fact it is possible to encode an asynchronous automaton as a network of automata with readings (the idea is to use one automaton per register, states being the possible values in this register). The converse is also possible.

Representing plans as partial orders is extremely useful: it reduces the search space since useless interleavings of concurrent events need not be explored. Notice that the set of factored plans $(w_1, ..., w_N)$ in networks of ARA gives rise to a notion of asymmetric conflict, as it is the case in Petri nets with read arcs (Baldan et al. (2001), Baldan et al. (2004)). One may have that two actions a and b in w_i and w_j are possible simultaneously, or with a before b, but not with b before a. This is of course due to the enabling by read arcs.

Distributed planning with networks of automata with readings was only explained in a three-automata network. However it generalizes to larger networks. The idea is to express a more general separation property, which leads to a notion of *communication graph* between automata of a network. When this graph is a tree, the separation property allows one to derive a message passing algorithm that exploits a generalization of Theorem 5. This kind of algorithm was studied *per se* in Fabre (2007), and adapted to networks of weighted automata (without reading) in Fabre and Jezequel (2009).

As we are interested in cost-optimal planning – i.e. finding minimal-cost plans when actions are equipped with additive costs – our future work will combine weighted automata and read arc mechanisms.

REFERENCES

Amir, E. and Engelhardt, B. (2003). Factored planning. In *IJCAI-03*.

Baldan, P., Busi, N., Corradini, A., and Pinna, G.M. (2004). Domain and event structure semantics for petri nets with read and inhibitor arcs. *Theoretical Computer Science*, 323 (1-3), 129–189.

Baldan, P., Corradini, A., and Montanari, U. (2001). Contextual petri nets, asymmetric event structures and processes. *Information and Computation*, 171, 1–49.

Bonet, B., Haslum, P., Hickmott, S., and Thiebaux, S. (2007). Directed unfolding of petri nets. In *UFO-07*.

Brafman, R.I. and Domshlak, C. (2006). Factored planning: How, when, and when not. In *AAAI-06*.

Brafman, R.I. and Domshlak, C. (2008). From one to many: Planning for loosely coupled multi-agent systems. In *ICAPS-08*.

Cassandras, C. and Lafortune, S. (1999). *Introduction to Discrete Event Systems*. Kluwer Academic Publishers.

Choi, J. and Amir, E. (2006). Factored planning for controlling a robotic arm: theory. In *CogRob 2006*.

Dechter, R. (2003). *Constraint Processing*, chapter 9, 245–270. Morgan Kaufmann Publishers.

Fabre, E. (2003). Convergence of the turbo algorithm for systems defined by local constraints. Technical Report PI 4860, INRIA.

Fabre, E. (2007). *Bayesian Networks of Dynamic Systems*, chapter 2, 21–37. Habilitation à diriger des recherches, Université Rennes 1.

Fabre, E. and Jezequel, L. (2009). Distributed optimal planning: an approach by weighted automata calculus. In *CDC'09*.

Hickmott, S., Rintanen, J., Thiebaux, S., and White, L. (2007). Planning via petri net unfolding. In *IJCAI-07*.

Mohri, M. (1997). Finite-state transducers in language and speech processing. *Computational Linguistics*, 23:2.

Nau, D., Ghallab, M., and Traverso, P. (2004). *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers.

Pearl, J. (1986). Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, 29, 241–288.

Zielonka, W. (1995). *The Book of Traces*, chapter 7, 205–248. World Scientific.