# Cross-over Composition - enforcement of fairness under unfair adversary

Joffroy Beauquier, Maria Gradinariu, and Colette Johnen

Laboratoire de Recherche en Informatique, UMR CNRS 8623,
Université de Paris Sud, F91405 Orsay cedex, France
{jb,mariag,colette}@lri.fr

**Abstract.** We study a special type of self-stabilizing algorithms composition : the cross-over composition ($A \diamond B$). The cross-over composition is the generalization of the algorithm compiler idea introduced in [3]. The cross-over composition could be seen as a black box with two entries and one exit. The composition goal is to improve the qualities of the first algorithm $A$, using as medium the second algorithm $B$. Informally, the obtained algorithm is $A$ after the transfer of $B$'s properties.
Here, we provide a complete analysis of the composition, when the algorithms ($A$ and $B$) are deterministic and/or probabilistic algorithms. Moreover, we show that the cross-over composition is a powerful tool in order to enforce a scheduler to have a fair behavior regarding to $A$.

## 1 Introduction

The idea of composing self-stabilizing algorithms in order to improve their adaptability was introduced by Gouda and Herman in [8]. In their approach an algorithm is composed by a number of $k$ layers such that the layer $i$, $1 < i \leq k$ depends on the variables which stabilize due to the actions of the layers from 1 to $i-1$. The proof of convergence of the composed algorithm follows by induction.

In the same paper, the authors present another type of composition which uses a selection predicate. The two modules which enter in the composition do not inter-communicate, but they are allowed to modify the same output variables. At a given time, the selection predicate is true only for one module (module that is allowed to modify the output variables) while the other module is waiting the flipping of predicate value.

Another type of independent module composition was defined by Varghese in [12]. The entities interact by means of their outputs. The obtained algorithm is the composition of the modules.

A special form of composition was defined by Dolev and Herman in [5]. The goal of this composition is to accelerate the self-stabilization of an algorithm $P$. $P$ will pick up the result of the fastest self-stabilizing algorithm of $(S_i)$, $i \in I$ in order to perform its own task. This technique needs some fair scheduler.

The cross-over composition $A \diamond B$ goal is to improve the qualities of the algorithm $A$, using as medium the second algorithm $B$. Informally, the obtained algorithm is the algorithm $A$ after the transfer of $B$'s computation properties. The main use of the cross-over composition is the transformation of a self-stabilizing algorithm $A$ under weak scheduler (central, alternating, k-fair, fair, ...) into an algorithm $(A \diamond B)$ which maintains the self-stabilization property under any unfair scheduler.

Moreover, we guarantee that the composed algorithm will satisfy the conjunction of the properties of the algorithms as in the Varghese composition. We show that all liveness and safety properties of $A$ and $B$ are also conveyed by $A \diamond B$ iff $B$ is fair when $A$ and $B$ are deterministic and/or probabilistic algorithms.

In Sec. 2, the model and self-stabilization definitions for deterministic and probabilistic algorithm are given. The cross-over definition is presented in Sec. 3. In Sect. 4, we study the propagation of the self-stabilization property. We explain how to use the cross-over composition to transform any self-stabilizing algorithm under some specific scheduler into an algorithm that converges under any unfair scheduler in Sec. 5.

## 2 Model

*Distributed Systems.* A distributed system can be modeled by a transition system. A transition system is a three-tuple $S = (\mathcal{C}, \mathcal{T}, \mathcal{I})$ where $\mathcal{C}$ is the collection of all configurations, $\mathcal{I}$ is a subset of $\mathcal{C}$ called the set of initial configurations, and $\mathcal{T}$ is a function from $\mathcal{C}$ to the set of $\mathcal{C}$ subsets. A $\mathcal{C}$ subset of $\mathcal{T}(c)$ is called a $c$ transition. An element of a $c$ transition $t$, is called an output of $t$. In a probabilistic system, there is a probabilistic law defined on the output of a transition; in a deterministic system, each transition has only one output. In Fig. 2, we can see the C00 transition called CH00 that has four outputs : C11, C12, C21 and C22.

The abstract model defined above is a mathematical representation of the reality. In fact, the distributed system is the collection of processors that communicate only with theirs processor neighbors to execute a distributed algorithm.

A *computation* of a distributed system $DS$ is a sequence of computation steps. A *maximal* computation is a sequence such that it is either infinite, or with a deadlock terminal configuration. The computations set of a distributed system $DS$ is denoted by $\mathcal{E}_{DS}$. A maximal computation $e$ is *fair* if and only if any processor performs infinity often an action. A fair computation $e$ is *$k$-fair* if and only if between two actions of a processor, any other processor performs at most $k$ actions. A maximal computation $e$ is *$k$-bounded* if and only if along $e$, till a processor $p$ is enabled to perform an action, another processor can perform at most $k$ actions. A $k$-fair computation is $k$-bounded; but the converse is not true.

When the distributed algorithm prevents the fairness because some processors are no more enabled, the $k$-bounded property guarantees the fairness between "enabled" processor. On a network of 4 processors ($p1$, $p2$, $p3$, $p4$), the following computation is not 1-fair $(p1's\ action, p3's\ action)^*$ but it is 1-bounded if along this computation $p2$ and $p4$ are never enabled to perform any action.

*Scheduler.* In this model, a *scheduler* is a *predicate* over the system computations. In a computation, a transition $(c_i, c_{i+1})$ occurs due to the execution of a nonempty subset of the enabled processors in the configuration $c_i$. In every computation step, this subset is chosen by the scheduler. At a computation step, a *central scheduler* chooses an enabled processor to execute its action; A *distributed unfair scheduler* chooses any nonempty subset of the enabled processors at each computation step. A *$k$-bounded scheduler* produces only $k$-bounded computations : it ensures the $k$-fairness between processors that are enabled to perform an action. An *alternating scheduler* produces only alternating computations : between two actions of a processor $p$ each $p$'s neighbor performs one and only one action.

An algorithm under a scheduler $D$ is *fair* (resp. $k$-fair) if any computation of the algorithm under $D$ is fair (resp. $k$-fair). When the property of fairness (resp. $k$-fairness) is verified by an algorithm under any scheduler then the algorithm is simply called fair (resp. $k$-fair).

Built on previous works on probabilistic automata (see [11, 13, 10]), [4] presented a framework for proving self-stabilization of probabilistic distributed systems based on the notion of strategy. A strategy is the set of computations that can be obtained under a specific scheduler choice. At the initial configuration, the scheduler "chooses" one set of enabled processors (it chooses a transition). For each output of the selected transition, the scheduler chooses a second transition, and so on. The formal strategy definition is based on the tree of computations. Let $c$ be a configuration. A *TS-tree* rooted in $c$, $\mathcal{T}ree(c)$, is the tree-representation of all computations beginning in $c$. Let $nd$ be a node in $\mathcal{T}ree(c)$ (i.e. a configuration), a *branch* rooted in $nd$ is the set of all $\mathcal{T}ree(c)$ computations starting in $nd$ with a computation step of the same $nd$ transition. The *degree* of $nd$ is the number of branches rooted in $nd$. A *sub-TS-tree of degree* 1 rooted in $c$ is a restriction of $\mathcal{T}ree(c)$ such that the degree of any $\mathcal{T}ree(c)$'s node (configuration) is at most 1. Figure 2 contains a strategy rooted in C00. A strategy may have a non-countable number of infinite computations. A strategy is defined as follows :

**Definition 1 (Strategy).** *Let $DS$ be a distributed system, let $D$ be a scheduler and let $c$ be a configuration. We call a strategy of $DS$ under $D$ rooted in $c$ a sub-TS-tree of degree 1 of $\mathcal{T}ree(c)$ such that any computation of the sub-tree satisfies the scheduler $D$.*

Let $st$ be a strategy of the distributed system $DS$, an *st-cone* $\mathcal{C}_h$ is the set of all possible $st$-computations with the same prefix $h$ (for more details see [10]). The last configuration of $h$ is denoted $last(h)$.

We have equipped a strategy with a probabilistic space (see [4] for more details). The measure of an *st*-cone $\mathcal{C}_h$ is the measure of $h$ (i.e., the product of the probability of every computation step occurring in $h$). An *st*-cone $\mathcal{C}_{h'}$ is called a *sub-cone* of $\mathcal{C}_h$ if and only if $h$ is a prefix of $h'$. Let $st$ be the strategy of Fig. 2; let $h$ be the prefix $(C00, ch00, C12)(C12, ch12, C56)$; in $st$, the probability of $\mathcal{C}_h$ is $p_A^2 \cdot (1 - p_B)^2$.

*Deterministic self-stabilization.* In order to define self-stabilization for a distributed system, we use two types of predicates : the legitimate predicate (defined on the system configurations and denoted by $\mathcal{L}$) and the problem specification (defined on the system computations and denoted by $\mathcal{PS}$). To prove the self-stabilization to $\mathcal{SP}$, one has to prove that all computations reach a legitimate configuration and that from a legitimate configuration any computation satisfies the predicate $\mathcal{SP}$. For instance, the leadership problem specification is "there is one leader in the network, called $p$, and $p$ stays the only leader forever". In this case, a legitimate configuration would be a configuration where there is one and only one leader. The correctness proof consists in ensuring that the system does not diverge from a legitimate configuration : once $p$ is the only leader, no other processor becomes leader and $p$ keeps its leadership.

Let $\mathcal{X}$ be a set and $Pred$ be a predicate defined on $\mathcal{X}$. The notation $x \vdash Pred$ means that the element $x$ of $\mathcal{X}$ satisfies the predicate $Pred$.

**Definition 2 (Deterministic self-stabilization).** *Let $DS$ be a distributed system. $DS$ is self-stabilizing for a specification $\mathcal{PS}$ if and only if the following two properties hold :*

- convergence — *all computations of $DS$ reach a configuration that satisfies the legitimate predicate denoted $\mathcal{L}$. Formally, $\forall e \in \mathcal{E}_{DS}$ $:: e = ((c_0, c_1)(c_1, c_2)\dots)$ $: \exists n \geq 1, c_n \vdash \mathcal{L}$;*
- correctness — *all computations starting in configurations satisfying the legitimate predicate satisfy the problem specification $\mathcal{PS}$. Formally, $\forall e \in \mathcal{E}_{DS}$ $:: e = ((c_0, c_1)\ (c_1, c_2)\ \dots)$ $: c_0 \vdash \mathcal{L} \Rightarrow e \vdash \mathcal{PS}$.*

*Probabilistic self-stabilization.* Let $DS$ be a distributed system. A predicate $P$ is closed for the computations of $DS$ if and only if when $P$ holds in a configuration $c$, $P$ also holds in any configuration reachable from $c$.

**Notation 1** *Let $DS$ be a distributed system, $D$ be a scheduler and $st$ be a strategy of $DS$ under $D$. Let $CP$ be the set of all system configurations satisfying a closed predicate $P$ (formally $\forall c \in CP, c \vdash P$). The set of $st$-computations that reach configurations of $CP$ is denoted by $\mathcal{EP}$ and its probability by $Pr_{st}(\mathcal{EP})$.*

**Definition 3 (Probabilistic Stabilization).** *A distributed system $DS$ is* self-stabilizing *under a scheduler $D$ for a specification $\mathcal{PS}$ if and only if there exists a closed legitimate predicate $\mathcal{L}$ defined on configurations such that in any strategy $st$ of $DS$ under $D$, the following conditions hold :*

- convergence — *The probability of the set of st-computations, that reach a configuration satisfying* $\mathcal{L}$ *is* 1. *Formally,* $\forall st, Pr_{st}(\mathcal{EL}) = 1$.
- correctness — *Any computation starting in a configuration satisfying* $\mathcal{L}$ *satisfies the specification* $\mathcal{PS}$.

## 3 Cross-over Composition

### 3.1 Definitions

In the sequel, we define the cross-over composition $A \diamond B$. The cross-over composition could be seen as a black box with two independent entries (two algorithms that do not share any variable) and one exit. The composition goal is to improve the qualities of the first Alg. $A$, using as medium the second Alg. $B$. The two algorithms which enter in the composition have different parts. $A$, referred in the following as the weak algorithm is the target of the transformation. $B$ referred as the strong algorithm is the transformation medium which transfers its properties to the weak algorithm.

The actions of $A$ are synchronized with the actions of $B$ : when an $A$ action is performed then a $B$ action is performed too. Thus, the computations of the composite algorithm under any scheduler have the same properties as the computations of $B$ in term of fairness.

- when a processor $p$ performs an action of $A$ it performs simultaneously an action of $B$ (both action guards were satisfied on $p$);
- a processor $p$ may perform an action of $B$ without performing an action of $A$ (in this case all action guards of $A$ are disabled on $p$).

The strong algorithm $B$ acts as a computation filter for the weak algorithm $A$ : $A$ will only deal with computations that can be obtained by $B$ under the current scheduler: $D$. The obtained algorithm $A \diamond B$ has the properties of $B$ under $D$ and the properties of $A$ under a scheduler that produces "$B$'s computations".

**Definition 4.**
*Let* $A$ *be an algorithm with* $n$ *actions as follows :*
$\forall i \in \{1, \ldots, n\} \quad < guard\ a_i > \quad \Rightarrow \quad < action\ a_i >$
*Let* $B$ *be an algorithm with* $m$ *actions as follows :*
$\forall j \in \{1, \ldots, m\} \quad < guard\ b_i > \quad \Rightarrow \quad < action\ b_j >$
*Assume that* $A$ *and* $B$ *do not share any variable. The cross-over composition* $A \diamond B$ *is the algorithm with the* $m.(n+1)$ *following actions :*
$\forall i \in \{1, \ldots, n\}, \forall j \in \{1, \ldots, m\}$
$< guard\ a_i > \wedge < guard\ b_j > \quad \Rightarrow \quad < action\ a_i >; < action\ b_j >$
$\forall j \in \{1, \ldots, m\}$
$\neg < guard\ a_1 > \wedge \ldots \wedge \neg < guard\ a_n > \wedge < guard\ b_j > \quad \Rightarrow \quad < action\ b_j >$

*Example 1.* Let $r$ be a unidirectional ring of $n$ processors. Algorithm $B$ has one integer variable $v$ and one action : on any processor $p$ ($lp$ being the left neighbor of $p$), $v_p \leq v_{lp} \Rightarrow v_p := v_{lp} + 1$. Any computation of $B$ has a suffix that is alternating. Algorithm $A$ has two variables $Currentlist$ and $BackupList$ (list of processor ids) and one action (without guard) : $\Rightarrow$ $p$ copies the content of $CurrentList$ of its left neighbor into its own $CurrentList$; then it concatenates its own id at the end of the list. If $p$ id appears two times in this list, $p$ copies the segment between its ids into the *backupList*. If in $p$'s $CurrentList$, $p$'s id appears three times then $p$ empties its $CurrentList$. The cross-over composition $A \diamond B$ has three variables : one integer and two lists of processors id; and the following action :

**b1.** $v_p \leq v_{lp} \quad \Rightarrow \quad v_p := v_{lp} + 1$; $p$ copies the content of $CurrentList$ of its left neighbor into its own $CurrentList$; then ... .

As the computations of $A \diamond B$ have an alternating suffix; one may prove that the algorithm stabilizes : every $BackupList$ list will contain the ordered list of processors id on the ring.

A probabilistic self-stabilizing leader election algorithm on anonymous ring is presented in [3]. This algorithm is the cross-over composition of three algorithms $(L \diamond RTC) \diamond DTC$ : $DTC$ ensures that the computations are $k$-fair, $RTC$ provides a token circulation on the ring if the computations are $k$-fair, and $L$ manages the leadership under the assumption that a token circulates in the ring.

**Observation 1** *One may notice that the algorithm $A2 \diamond (A1 \diamond B)$ is not the algorithm $(A2 \diamond A1) \diamond B$. To prove that one may study the cross-over composition of three simple algorithms (i.e. each algorithm has one action).*

### 3.2 Deterministic Properties Propagation

In the following, we study the propagation of deterministic algorithm properties on the obtained algorithm.

**Lemma 1 (propagation of properties on computations).** *Let $A \diamond B$ be the cross-over composition between the algorithms $A$ and $B$. Let $P$ be a predicate on $B$'s computations. If any maximal computation of $B$ under the scheduler $D$ satisfies the predicate $P$ then any maximal computation of $A \diamond B$ under $D$ satisfies $P$.*

*Proof.* Assume that there is at least one maximal computation of $A \diamond B$, $e$, under $D$ which does not satisfy the predicate $P$. The projection of $e$ on $B$ is unique and maximal. Let $e_B$ be this projection. Since $e$ does not satisfy the predicate $P$ then $e_B$ does not either which contradicts the Lemma hypothesis.

**Corollary 1 (propagation of fairness).** *Let $A \diamond B$ be the cross-over composition between the algorithms $A$ and $B$. If Alg. $B$ is fair under $D$ then $A \diamond B$ is a fair algorithm under $D$.*

**Corollary 2 (propagation of $k$-fairness).** *Let $A \diamond B$ be the cross-over composition between the algorithms $A$ and $B$. If $B$ is $k$-fair under a scheduler $D$ then $A \diamond B$ is $k$-fair under $D$.*

The proof of the following Lem. is similar to the proof of the Lem. 1.

**Lemma 2 (propagation of convergence properties).** *Let $A \diamond B$ be the cross-over composition between the algorithms $A$ and $B$. Let $P$ be a predicate on the $B$'s configurations. If any maximal computation of $B$ under the scheduler $D$ reaches a configuration which satisfies the predicate $P$ then any maximal computation of $A \diamond B$ under $D$ reaches a configuration which satisfies $P$.*

**Corollary 3 (propagation of liveness).** *Let $A \diamond B$ be the cross-over composition between the algorithms $A$ and $B$. If $B$ is without deadlock under the scheduler $D$ then $A \diamond B$ is without deadlock under $D$.*

**Lemma 3 (maximality of the weak projection).** *Let $A \diamond B$ be the cross-over composition between $A$ and $B$. If $B$ is fair under the scheduler $D$ then the projection on $A$ of any maximal computation of $A \diamond B$ under $D$ is maximal.*

*Proof.* Let $e$ be a maximal computation of $A \diamond B$. Let $e_A$ be the projection of $e$ on $A$. Assume that $e_A$ is not maximal. Hence $e_A$ is finite and its last configuration is not a deadlock. Let $e$ be $e = e_1 e_2$ where the projection of $e_1$ on $A$ is $e_A$ and the projection of $e_2$ is a maximal computation which does not contain any action of $A$. Let $c$ be the last configuration of $e_1$. The projection of this configuration on $A$ is not a deadlock, then there is at least one processor, $p$, which satisfies a guard of $A$ in $c$. Using the fairness of $B$ and Corollary 1 we prove that $p$ executes an action of $B$ in $e_2$. According to the definition of the cross-over composition, during the computation of $p$'s first action in $e_2$, it executes an action of $A$. There is a contradiction with the assumption that no action of $A$ is executed in $e_2$.

### 3.3 Propagation on a Simple Probabilistic Cross-over Composition

In this section, we study the properties of a strategy of $A \diamond B$ when $X$ ($= A$ or $B$) is a probabilistic algorithm and the other one is a deterministic algorithm. Figure 1 displays an example of such a cross-over composition.

**Lemma 4.** *Let $X$ be a probabilistic algorithm, let $Y$ be a deterministic one, and let $Z = X \diamond Y$ and $W = Y \diamond X$ be their cross-over composition. Let $st$ be a strategy of $Z$ or $W$ under the scheduler $D$. Let $st_X$ be the projection of $st$ on the algorithm $X$. $st_X$ is a strategy of $X$ under $D$.*

**Proof outline :** Let c be the initial configuration of $st$, and $c_X$ its projection on $X$. Let $\mathcal{TS}(c_X)$ be the tree representation of $X$ computations beginning at $c_X$. Let $st'$ be the sub-tree of $\mathcal{TS}(c_X)$ that contains all computations that are $st$ projections. $st'$ is a strategy : all computation steps beginning at $n$ (a node) in $st'$, belong to the same transition : $nd$; and all computation steps of $nd$ transition are in $st'$.

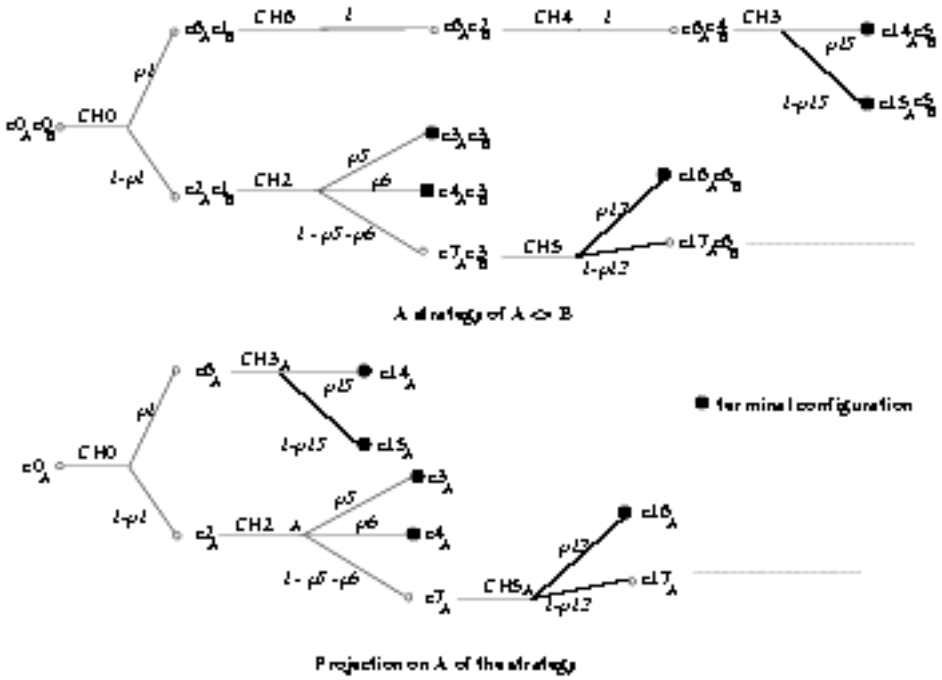



**Fig. 1. The projection of a strategy of A ⋄ B on A (B being a deterministic algorithm)**

**Theorem 1.** *Let $X$ be a probabilistic algorithm, let $Y$ be a deterministic one, and let $Z = X \diamond Y$ and $W = Y \diamond X$ be their cross-over composition. Let $st$ be a strategy of $Z$ or $W$ under the scheduler $D$. Let $st_X$ be the projection of $st$ on $X$. Let $PC_X$ be a predicate over the $X'$s configurations, then $Pr_{st}(\mathcal{EPC}_{\mathcal{X}}) = Pr_{st_X}(\mathcal{EPC}_{\mathcal{X}})$. Let $PE_X$ be a predicate over the $X'$s computations, then $Pr_{st}\{e \in st \mid e \vdash PE_X\} = Pr_{st_X}\{e' \in st_X \mid e' \vdash PE_X\}$.*

**Proof outline :** Let $\mathcal{C}_h$ be a cone of $st$. The projection of $\mathcal{C}_h$ on $X$ is a cone of $st_X : \mathcal{C}_{h'}$ where $h'$ is the projection of $h$ on $X$.

### 3.4 Propagation on a Double Probabilistic Cross-over Composition

In this section, we study the properties of a strategy of $A \diamond B$ when $A$ and $B$ are probabilistic algorithms. The projection of a strategy on an algorithm is not a strategy (Fig. 3 is the projection on $B$ of the strategy of Fig. 2). The projection is decomposed in strategies.

**Definition 5 (derived strategies).** *Let $st$ be a strategy of $A \diamond B$ and let $st_{projX}$ be the projection tree obtained after the projection of the computations in $st$ on*

*$X$ ($X = A$ or $X = B$). A derived strategy of $st_{projX}$ is a subtree of $st_{projX}$ whose degree equals 1.*

**Observation 2** *Let $st$ be a strategy of $A \diamond B$. Let $st_{projX}$ be the projection of $st$ on $X$ ($X = B$ or $A$). For the sake of simplicity, we assume that $X = B$. Let $st_B$ be a strategy derived from $st_{projB}$. Each cone of computations in $st_B$, $\mathcal{C}_{h|B}$, is the projection of a cone of $st$, $\mathcal{C}_h$, which probability is given by the probability of $\mathcal{C}_{h|B}$ in $st_B$ multiplied by $\delta^{hB}$. The weight of the cone of history $h|B$ in $st_B$ (denoted by $\delta^{hB}$) is the probability of the $A$ computation steps executed in the history of $\mathcal{C}_h$. Hence, $Pr_{st}(\mathcal{C}_h) = \delta^{hB} \cdot Pr_{st_B}(\mathcal{C}_{h|B})$.*

**Definition 6 (projection strategies on $X$).** *We call a projection strategy $st_X$ a derived strategy of $st_{projX}$ such that all cones of $st_X$ having the same length have the same weight. We note $\delta_n^{st_X}$ the weight of $n$-length cones of $st_X$.*

For instance, Fig. 4 and Fig. 5 contain 4 projection strategies on $B$ of the strategy $st$ (Fig. 2). Each strategy has a different $\delta_2$ value.

**Observation 3** *Let $N$ be an integer. Let $st$ be a strategy of $A \diamond B$. Let $st_{projX}$ be the projection of $st$ on $X$ ($X = B$ or $A$). For the sake of simplicity, we assume that $X = B$. Let $M^B$ be the set of projection strategies of $st_{projB}$. There is a finite subset of $M^B$, denoted $M_N^B$ such that $\sum_{st_B \in M_N^B} \delta_N^{st_B} = 1$. Such a subset of $M$ is called a $N$-length $B$ picture of $st$.*
*Each $N$-length cone of $st$ has one and only one projection on $B$ in $M_N^B$.*
*There are several subsets of $M_B$ that are "$N$-length $B$ pictures of $st$".*

*The 4 strategies of Fig. 4 and Fig. 5 constitute a $M_2^B$ set. We have $\sum_{i=1}^{i=4} \delta_2^{st_i} = 1$*

We show that if any strategy of an algorithm for which the set of computations reaching a configuration satisfying a predicate $P$ has the probability 1, then in any strategy of the composed algorithm, this set has the probability 1.

**Lemma 5 (probabilistic propagation).** *Let $\mathcal{L}$ be a predicate over the $X$'s states ($X = A$ or $X = B$). Let $st$ be a strategy of $A \diamond B$ under a scheduler $D$. If for every strategy $st_X$, being a projection strategy of $st$ on $X$, we have $P_{st_X}(\mathcal{EL}) = 1$ then $P_{st}(\mathcal{EL}) = 1$.*

*Proof.* Let $st$ be a strategy of $A \diamond B$. Let $st_{projX}$ be the projection of $st$ on $X$ ($X = B$ or $A$). We assume that $X = B$. Let $M_B$ be the set of projection strategies of $st_{projB}$.

We denote by $\mathcal{EL}_N$ the union of the cones of a given strategy that have the following properties : (i) their history length is $N$ and (ii) they have reached a legitimate configuration.

Let $\epsilon$ be a real inferior to 1. By hypothesis, there is an integer $N$ such that on any strategy $st_B$ of $M_N^B$ (a $N$-length $B$ picture of $st$) we have $Pr_{st_B}(\mathcal{EL}_N) \geq 1-\epsilon$.
$Pr_{st}(\mathcal{EL}_N) = \sum_{st_i \in M_N^B}[Pr_{st_i}(\mathcal{EL}_N) \cdot \delta_N^{st_i}] \geq (1-\epsilon).\sum_{st_i \in M_N^B} \delta_N^{st_i} \geq 1-\epsilon$. In $st$, the set of computations reaching legitimate configurations in $N' \leq N$ steps has a probability greater than $1-\epsilon$.

Therefore, for any sequence $\epsilon_1 > \epsilon_2 > \epsilon_3...$ there is a sequence $N_1 \leq N_2 \leq N_3...$ such that $Pr_{st}(\mathcal{EL}_{N_i}) \geq 1-\epsilon_i$. Then, $\lim_{n\to\infty} P_{st}(\mathcal{EL}_n) = P_{st}$(computations reaching a legitimate configuration) $= 1$.

## 4 Cross-over Composition and Self-Stabilization

In the sequel, we study the propagation of the self-stabilization property from an algorithm to the resulting algorithm of a cross-over composition. The propagation with self-stabilization (in the deterministic case) is a direct consequence of Lem. 1 and 2 when the strong algorithm ($B$) is the propagation initiator.

**Lemma 6.** [**self-stabilization propagation to the deterministic Alg.** $A \diamond B$ **from Alg.** $B$] *Let $A \diamond B$ be the cross-over composition between the deterministic algorithms $A$ and $B$. If Alg. $B$ self-stabilizes for the specification $SP$ under the scheduler $D$ then $A \diamond B$ is self-stabilizing for $SP$ under $D$.*

*Proof.* The proof is a direct consequence of the Lem. 1 and 2. In order to prove the convergence we apply Lem. 1 for the property which characterizes the legitimate configurations. The correctness proof results from Lem. 2 applied for the specification $SP$.

In order to ensure the liveness of the weak algorithm, the strong algorithm must be fair.

**Lemma 7.** [**self-stabilization propagation to the deterministic Alg.** $A \diamond B$ **from Alg.** $A$ ] *Let $A \diamond B$ be the cross-over composition between the deterministic algorithms $A$ and $B$ ($B$ is a fair algorithm). If $A$ is self-stabilizing for the specification $SP$ under the scheduler $D$ then $A \diamond B$ stabilizes for the specification $SP$ under $D$.*

*Proof.* Let $e$ be a maximal computation of $A \diamond B$.

- convergence of Alg. $A \diamond B$. Let $P$ be the predicate which characterizes the legitimate configurations of $A$ and let $e_A$ be the projection of $e$ on $A$. According to Lem. 3, $e_A$ is a maximal computation of $A$ and $e_A$ reaches a legitimate configuration ($A$ is self-stabilizing).

- correctness of Alg. $A \diamond B$. Let $e$ be a computation of $A \diamond B$ which starts in a configuration satisfying $P$. Let $e_A$ be its projection on $A$. The computation $e_A$ is maximal and starts in a configuration which satisfies $P$. $A$ is self-stabilizing then $e_A$ satisfies the specification $SP$, hence $e$ satisfies also the specification $SP$.

**Lemma 8.** [**self-stabilization propagation to the probabilistic Alg.** $A \diamond B$ **from Alg.** $B$] *Let $A \diamond B$ be the probabilistic cross-over composition between the algorithms $A$ and $B$. If the algorithm $B$ self-stabilizes for the specification $SP$ under the scheduler $D$ then $A \diamond B$ is a probabilistic self-stabilizing algorithm for $SP$ under $D$.*

*Proof.* Let us study the propagation in the two possible cases : $B$ is a deterministic algorithm or is a probabilistic one. The idea of the proof is to analyze an arbitrary strategy $st$ of $A \diamond B$ under $D$.

- $B$ is deterministic. The projection on $B$ of every computation of the strategy $st$ is maximal. Every computation of $st$ reaches a legitimate configuration; and then, it satisfies the specification $SP$. $A \diamond B$ is self-stabilizing for $SP$ under $D$.
- $B$ is probabilistic. Let $st$ be a strategy of $A \diamond B$. Let $L$ be the legitimate predicate associated with $SP$. According to Lem. 5 or to Theo. 1. $P_{st}(\mathcal{E}L) = 1$. Moreover, according to Lem. 1, all computations of $st$ that reach $L$ have a suffix that satisfies $SP$.

The self-stabilization propagation from the weak algorithm is possible only if the strong algorithm is fair.

**Lemma 9.** [**self-stabilization propagation to the probabilistic Alg.** $A \diamond B$ **from Alg.** $A$] *Let $A \diamond B$ be the probabilistic cross-over composition between the algorithms $A$ and $B$. If Alg. $A$ self-stabilizes for the specification $SP$ under the scheduler $D$ and Alg. $B$ is a fair algorithm under $D$ then $A \diamond B$ is a probabilistic self-stabilizing algorithm for $SP$ under $D$.*

**Theorem 2.** [**self-stabilization propagation from** $A$ **and** $B$] *Let $A \diamond B$ be the probabilistic cross-over composition between the algorithms $A$ and $B$. If Alg. $A$ self-stabilizes for the specification $SP$ under the scheduler $D$ and Alg. $B$ is self-stabilizing for the specification $SR$ and is fair under $D$ then $A \diamond B$ is a probabilistic self-stabilizing algorithm for $SP \wedge SR$ under $D$.*

Note that in both cases — deterministic and probabilistic — the strong algorithm will propagate the self-stabilization property to the result of composition without any restriction, while the propagation initiated by the weaker one can be realized if and only if the strong algorithm is fair.

## 5 Application : Scheduler Transformation

In this section, we present the main application of the cross-over application, the scheduler transformation. We show how to use the cross-over composition to transform any self-stabilizing algorithm under some specific scheduler into an algorithm that converges under any unfair scheduler.

**Definition 7 (fragment of owner $p$).** *Let $e$ be a computation of a distributed system and let $p$ be a processor such that $p$ executes its actions more than one time in $e$. A fragment of $e$ of owner $p$, $f_{pp}$ is a fragment of $e$ such that :*

- *$f_{pp}$ starts and finishes with a configuration where $p$ executes an action;*
- *along $f_{pp}$, $p$ executes exactly two actions (during the first and the last step of $f_{pp}$).*

**Lemma 10 (from $k$-fairness to the $k$-bound property).** *Let us consider the cross-over composition $A \diamond B$. Let $e$ be a computation of $A \diamond B$ under an arbitrary scheduler. If $B$ is $k$-fair then the projection of $e$ on $A$ is $k$-bounded.*

*Proof.* Suppose that $e_A$ is not $k$-bounded. Hence, there is a fragment $f_A$ of $e_A$ such that a processor $q$ performs $k+1$ actions during $f_A$ and such that another processor $p$ performs no action and it is always enabled along $f_A$. $f_A$ is the projection of a fragment of $e$ called $f$. According to the definition of $A \diamond B$, $f$ has the following properties (i) $p$ performs no action in $f$ (ii) $q$ performs at least $k+1$ actions in $f$. $f$ is part of a fragment owned by $p$ called $f_{pp}$ such that $q$ performs at least $k+1$ actions in $f_{pp}$. $f_{pp}$ does not exist because $A \diamond B$ is $k$-fair (Cor. 2).

The following theorem gives a tool for transforming an algorithm $A$ that self-stabilizes under a $k$-bounded scheduler into an algorithm $A'$ that self-stabilizes under an unfair scheduler. Let $B$ be an algorithm whose computations are $k$-fair, the transformed is $A' = A \diamond B$.

**Theorem 3.** *Let $A \diamond B$ be the cross-over composition between $A$ and $B$. $A$ is a self-stabilizing algorithm for the specification $SP$ under a $k$-bounded scheduler. $B$ is a $k$-fair algorithm. The algorithm $A \diamond B$ is self-stabilizing for the specification $SP$ under an unfair scheduler.*

*Proof.* Let $e$ be a maximal computation of $A \diamond B$ and let $e_A$ be its projection on the weak module. Since $B$ is $k$-fair, according to Lem. 10, $e_A$ is a $k$-bounded computation.
*correctness proof:* Let $\mathcal{L}$ be the legitimate predicate associated with $SP$. Once $e_A$ has reached a legitimate configuration (a configuration that satisfies the predicate $\mathcal{L}$), it satisfies the specification $SP$.
*convergence proof:*

- *$A$ is a deterministic algorithm* : $e_A$ reaches a legitimate configuration.

- *A is a probabilistic algorithm.* Let $st$ be a strategy of $A \diamond B$ under a distributed unfair scheduler. Let $st_A$ be a projection strategy of $st$ on $A$. According to Lem. 10 any execution of $st_A$ is $k$-bounded. According to the hypothesis, $P_{st_A}(\mathcal{EL}) = 1$. According to Lem. 5 or to Theo. 1, $P_{st}(\mathcal{EL}) = 1$.

Note that the transformation depends directly on the properties of the strong algorithm of a cross-over composition. The main question is "are-there algorithms able to satisfy the $k$-bound property under any unfair scheduler ?" The answer is positive and in the following we show some examples :

| **Protocol** | **Topology** | **Network type** | **Scheduler transf.** |
|---|---|---|---|
| [7] | general networks, bidir. | with id | central to unfair |
| [1] | general networks, bidir. | with id | central to unfair |
| [1] | general networks, bidir. | with id | $X_1$-bounded to unfair |
| [3] | rings, unidir. | anonymous | $X_1$-bounded to unfair |
| [6] | rings, unidir. | anonymous | alternating to central |
| [2] | general networks, unidir. | anonymous | $X_2$-bounded to unfair |

$X_1 = n - 1$; and $X_2 = n.MaxOut^{Diam}$ where $MaxOut$ is the maximal network out-degree and $Diam$ is the network diameter.

The procotols [1] and [7] are working in the id-based networks. In the case of anonymous networks an algorithm which ensures the transformation of a central scheduler to a distributed scheduler could be the algorithm of [1] executed on top of an algorithm which ensures an unique local naming (neighbor processors do not have the same id; but distant processors may have the same id).

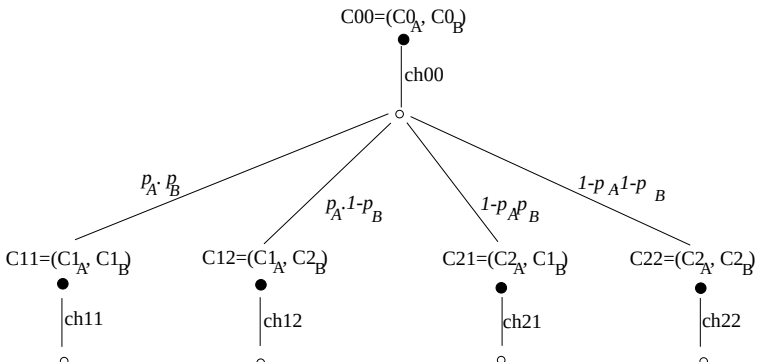
## 6 Conclusion

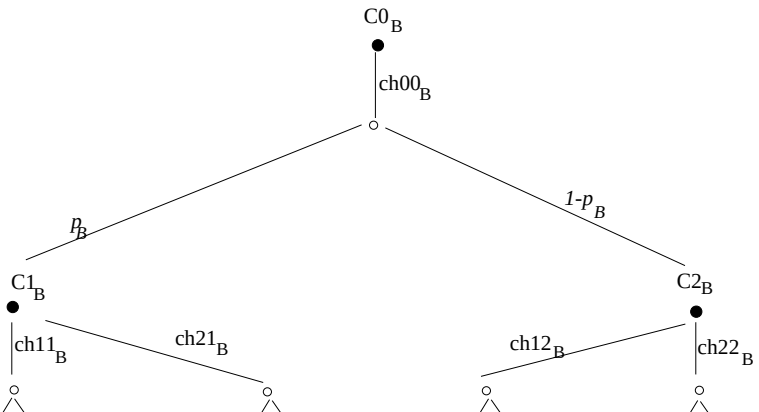
We have presented a transformation technique to transform self-stabilizing algorithms under weak scheduler (k-bounded, central, *alternating*, ...) into algorithms which maintain the self-stabilizing property under unfair and distributed schedulers.

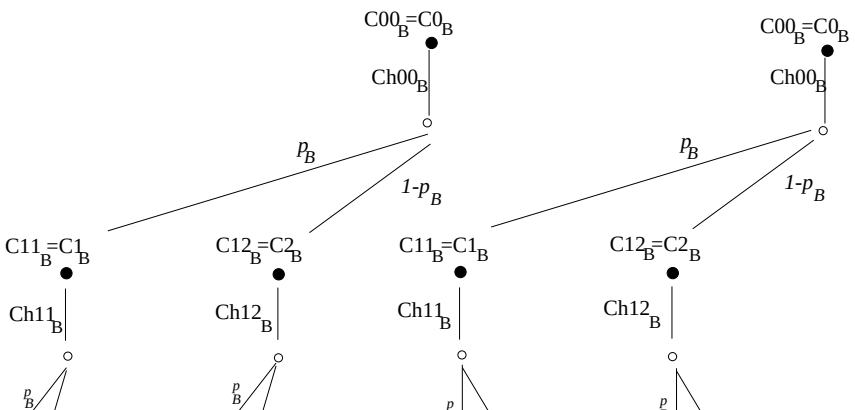
The key of this transformation is the cross-over composition $A \diamond B$ : roughly speaking, the obtained computations are the computations of $A$ under a scheduler that provides the $B$'s computations. The cross-over composition is a powerful tool to obtain only specific computations under any unfair scheduler. Indeed, if all $B$'s computations have "the $D$ properties" then $A$ only needs to be a self stabilizing algorithm for the specification $SP$ under the weak scheduler $D$ to ensure that $A \diamond B$ is a self stabilizing algorithm, for the specification $SP$ under any unfair scheduler.

## References


1. Beauquier J., Datta A., Gradinariu M., and Magniette F.: Self-stabilizing Local Mutual Exclusion and Daemon Refinement. *DISC'2000, 14th International Symposium on Distributed Computing*, LNCS:1914 (2000) 223-237
2. Beauquier J., Durand-Lose J., Gradinariu M., Johnen C.: Token based self-stabilizing uniform algorithms. *tech. Rep. no. 1250, LRI, Universit Paris-Sud; to appear in The Chicago Journal of Theoretical Computer Science* (2000)
3. Beauquier J., Gradinariu M., Johnen C.: Memory Space Requirements for Self-stabilizing Leader Election Protocols. *PODC'99, 18th Annual ACM Symposium on Principles of Distributed Computing* (1999) 199-208
4. Beauquier J., Gradinariu M., Johnen C.: Randomized self-stabilizing optimal leader election under arbitrary scheduler on rings. Tech. Rep. no. 1225, LRI, Universit Paris-Sud (1999)
5. Dolev S., Herman T.: Parallel composition of stabilizing algorithms. *WSS'99, fourth Workshop on Self-Stabilizing Systems* (1999) 25-32
6. Fich F., Johnen C., A space optimal deterministic, self-stabilizing, leader election algorithm for unidirectional Rings. *DISC'2001, 15th International Symposium on Distributed Computing* (2001)
7. Gouda M., Haddix F.: The alternator. *WSS'99, Fourth Workshop on Self-Stabilizing Systems* (1999) 48-53
8. Gouda M., Herman T.: Adaptive programming. *IEEE Transactions on Software Engineering* **17** (1991) 911-921
9. Kakugawa, H., Yamashita, M.: Uniform and Self-stabilizing Token Rings Allowing Unfair Daemon. *IEEE Transactions on Parallel and Distributed Systems* **8** (1997) 154-162
10. Segala R.: *Modeling and Verification of Randomized Distributed Real-time Systems.* PhD thesis, MIT, Departament of Electrical Engineering and Computer Science (1995)
11. Segala R., Lynch N.: Probabilistic simulations for probabilistic processes. *CONCUR'94, 5th International Conference on Concurrency Theory*, LNCS:836, (1994)
12. Varghese G.: Compositional proofs of self-stabilizing protocols. *WSS'97, Third Workshop on Self-stabilizing Systems* (1997) 80-94
13. Wu S. H., Smolka S. A., Stark E. W.: Composition and behaviors of probabilistic I/O automata. *CONCUR'94, 5th International Conference on Concurrency Theory*, LNCS:836, (1994) 513-528


C00=(C0A, C0B)
ch00
pA. pB
pA.1-pB
1-pA pB
1-pA1-p B
C11=(C1A, C1B)
C12=(C1A, C2B)
C21=(C2A, C1B)
C22=(C2A, C2B)
ch11
ch12
ch21
ch22


C0B
ch00B
pB
1-pB
C1B
C2B
ch11B
ch21B
ch12B
ch22B


C00B=C0B
Ch00B
pB
1-pB
C11B=C1B
C12B=C2B
Ch11B
Ch12B
C00B=C0B
Ch00B
pB
1-pB
C11B=C1B
C12B=C2B
Ch11B
Ch12B

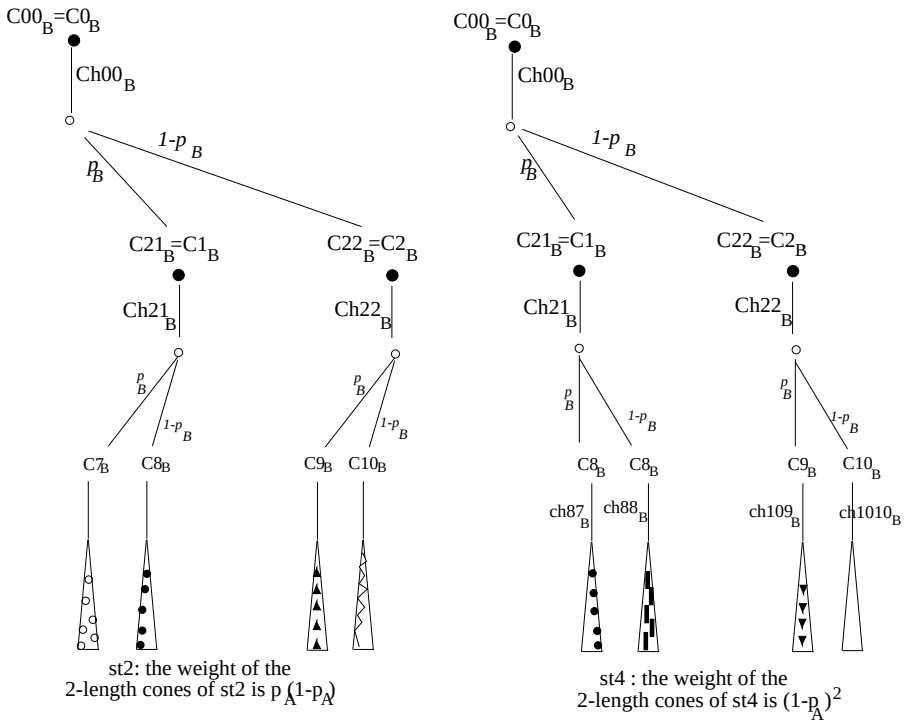



**Fig. 5.** 2-length beginning of projection strategies on $B$