Cross-over composition *

Joffroy Beauquier Maria Gradinariu Colette Johnen

L.R.I./C.N.R.S., Université de Paris-Sud, bat 490, 91405 Orsay Cedex, France
jb,colette,mariag@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 [BGJ99a].

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 the algorithm A after the transfer of B's properties.

Here, we provide a complete analyze of the composition, when the components (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 the algorithm A.

1 Introduction

The idea of composing self-stabilizing algorithms in order to improve their adaptability was introduced by Gouda and Herman in [GH91]. 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 convergence of the composed algorithm results 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 does 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 for the other module is not enabled.

Another type of independent modules composition was defined by Varghese in [Var97]. The entries interact by means of their outputs. The obtained algorithm is the conjugate of the modules.

A special form of composition was defined by Dolev and Herman in [DH99]. The goal of this composition is to accelerate the self-stabilization of an algorithm P. P will pick up the result of

*contact author : Colette Johnen, LRI, Université de Paris-Sud, centre d'Orsay - bat 490, 91405 Orsay Cedex, France. fax: +33 1 69 25 65 86, e-mail: colette@lri.fr, web: www.lri.fr/~colette/

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, 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 verify the conjunction of the properties of the components as in the Varghese composition. We provide a complete analyze the $A \diamond B$ properties according to the properties of A and B when A and B are deterministic and/or probabilistic algorithms.

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 on the output of a transition; in a deterministic system, each transition has only one output. In Figure 2, we can see the C00 transition called CH00 that has four outputs : C11, C12, C21 and C22.

A *computation* of a distributed system DS is a *maximal* sequence of computation steps. *Maximality* means that the sequence is either infinite, or the terminal configuration is a deadlock. All computations considered in this paper are assumed to be maximal. The computation set of a distributed system DS is denoted by $\mathcal{E}_{DS}$.

A 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 computation e is *k-bounded* if and only if along e, till a processor p is enabled another processor can perform at most k actions. A k-fair computation is k-bounded; but the converse is not true.

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.

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).

Building on previous works on probabilistic automata (see [SL94, WSS94, Seg95]). [BGJ99b] 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 strategy formal definition is based on the tree of computations. Let c be a system 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 is defined as follows :

Definition 2.1 (Strategy) *Let DS be a distributed system, let D be a scheduler and let c be a TS 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 verifies 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 [Seg95]). The last configuration of h is denoted $last(h)$.

We have equipped a strategy with a probabilistic space (see [BGJ99b] for more details). The measure of an st-cone $\mathcal{C}_h$ is the measure of the prefix 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 $h' = [hx]$, where x is a computation factor. Let st be the strategy of the Figure 2; let h be the prefix $(C00, ch00, C12)(C12, ch12, C56)$; in st, the probability of $\mathcal{C}_h$ is $p_A^2.(1-p_B)^2$.

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 $\mathcal{PS}$.

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

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 2.1 *Let $\mathcal{DS}$ 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 2.3 (Probabilistic Stabilization) *A distributed system $\mathcal{DS}$ is* self-stabilizing *under a scheduler D for a specification PS if and only if there exists a closed legitimate predicate $\mathcal{L}$ on configurations such that in any strategy st of $\mathcal{S}$ 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 verifying $\mathcal{L}$ satisfies the specification PS.*

3 Cross-over composition

3.1 Definitions

In the following, we define the cross-over composition $A \diamond B$. 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. The two algorithms which enter in the composition have different parts. A, referred in the following as the weak component is the target of the transformation. B referred as the strong component 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 (the 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 no action guard of A was satisfied on p).

Definition 3.1
Let A be an algorithm with n rules as follows :
$\forall i \in \{1, \ldots, n\} \quad < guard\ a_i > \Rightarrow < action\ a_i >$
Let B be an algorithm with m rules as follows :
$\forall j \in \{1, \ldots, m\} \quad < guard\ b_i > \Rightarrow < action\ b_j >$
The cross-over composition $A \diamond B$ is the algorithm with the $m.(n+1)$ following rules :
$\forall i \in \{1, \ldots, n\}, \forall j \in \{1, \ldots, m\} \quad < guard\ a_i > \wedge < guard\ b_j > \Rightarrow < action\ a_i >; < action\ b_j >$
$\forall j \in \{1, \ldots, m\} \quad \neg < guard\ a_i > \wedge \ldots \wedge \neg < guard\ a_n > \wedge < guard\ b_j > \Rightarrow < action\ b_j >$

3.2 Deterministic Properties Propagation

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

Lemma 3.1 (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 the B's computations. If any maximal computation of B under the scheduler D verifies the predicate P then any maximal computation of $A \diamond B$ under D verifies P.*

Proof: Assume that there is at least one maximal computation of $A \diamond B$, e, under D which does not verify the predicate P. The projection of e on B is unique and maximal. Let e_B be this projection. Since e does not verify the predicate P then neither e_B which contradicts the lemma hypothesis. □

Corollary 3.1 (propagation of fairness) *Let $A \diamond B$ be the cross-over composition between the algorithms A and B. If the algorithm B is fair under D then $A \diamond B$ is a fair algorithm under D.*

Corollary 3.2 (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.*

Lemma 3.2 (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 verifies the predicate P then any maximal computation of $A \diamond B$ under D reaches a configuration which verifies P.*

Proof: Assume that there is at least one maximal computation of $A \diamond B$, e, under D which does not reach a configuration which verifies P. Let e_B be the projection of e on B. Following the construction of the projection, e_B is unique and maximal. Since no configuration in e verifies the predicate P then no configuration in e_B verifies the predicate P, contradiction with the lemma hypothesis. □

Corollary 3.3 (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 configuration under D.*

Lemma 3.3 (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 any maximal computation of $A \diamond B$ under the scheduler D has a maximal projection on the weak algorithm.*

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 verifies a guard of A in c. Using the fairness of B and Corollary 3.1 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 Properties Propagation on a simple probabilistic cross-over composition

In this section, we study the properties of a strategy of $A \diamondsuit B$ 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.

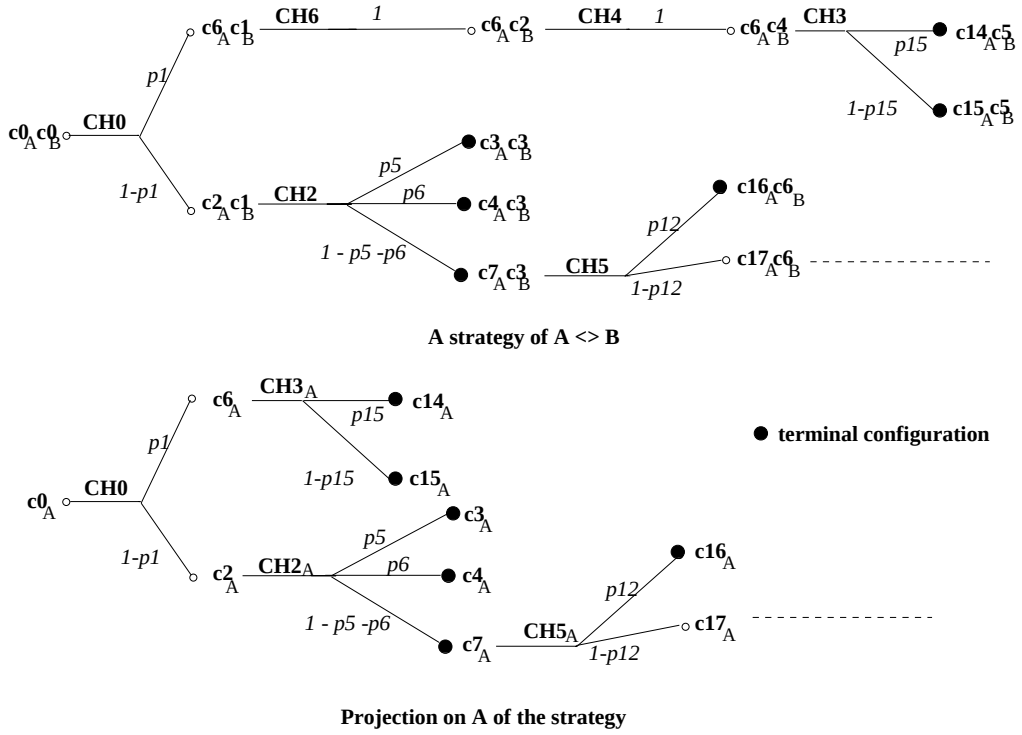

Figure 1: **The projection of a strategy of A $\diamondsuit$ B on A (B being a deterministic algorithm)**

Lemma 3.4 *Let X be a probabilistic algorithm, let Y be a deterministic one, and let $A \diamond B$ their cross-over composition where $X = A$ or B, and Y is the other component. Let st be a strategy of the game between the $A \diamond B$ under the scheduler D. Let st_X be the projection of st on the module X. st_X is a strategy of the game between X and the scheduler D.*

Theorem 3.1 *Either A is a probabilistic algorithm and B is a deterministic and fair algorithm ($X = A$); or A is a deterministic algorithm and B is a fair probabilistic algorithm ($X = B$). Let st be a strategy of $A \diamond B$ and let st_X be the projection of st on X. Let PC_X be a predicate over the X's configurations, then $Pr_{st}(\mathcal{EPC}_X) = Pr_{st_X}(\mathcal{EPC}_X)$*
Let PE_X be a predicate over the X's computations, then $Pr_{st}\{e \in st \mid e \vdash PE_X\} = Pr_{st_X}\{e' \in st_X \mid e' \vdash PE_X\}$.

3.4 Properties Propagation on a double probabilistic cross-over composition

In this section, we study the properties of a strategy of $A \diamondsuit B$ when A and B are probabilistic algorithms. The straight projection of a strategy on the modules is not a strategy (Figure 3 is the

straight projection on B of the strategy of Figure 2). The straight projection is decomposed in a countable number of strategies.

Definition 3.2 (derived strategies) *Let st be a strategy of $A \diamondsuit B$ 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.*

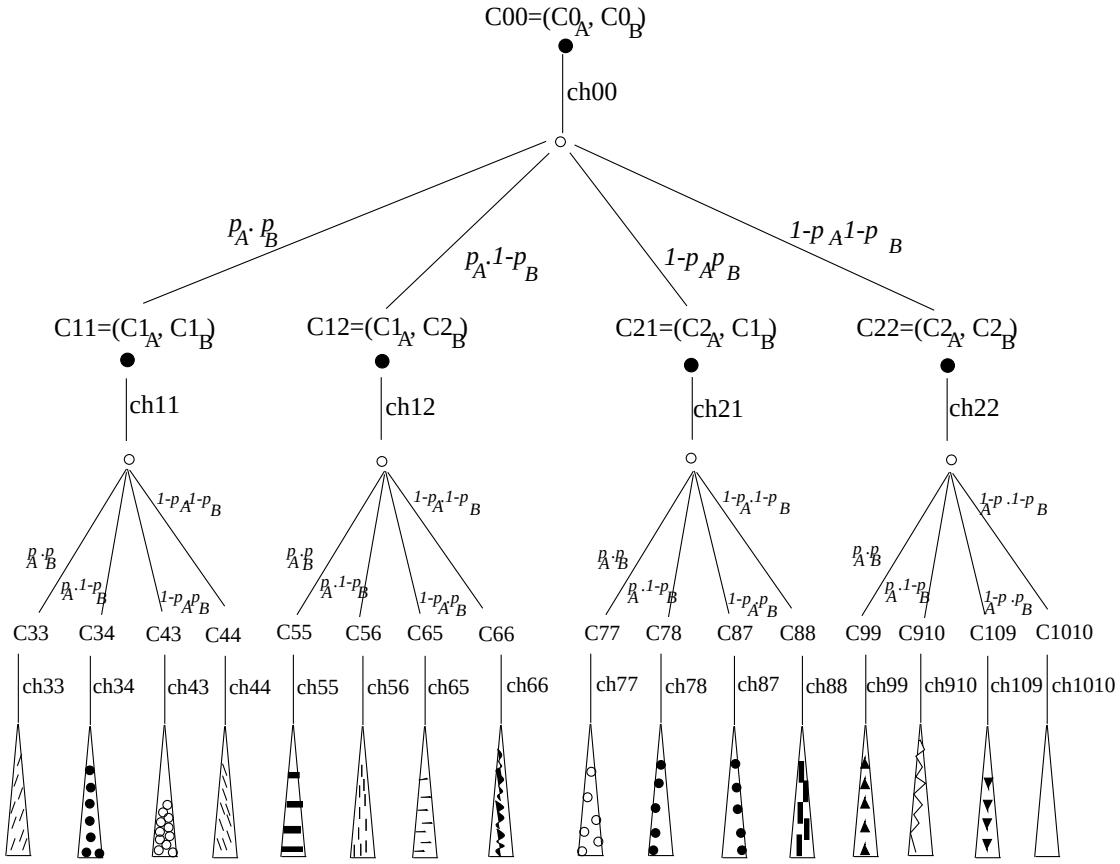

Figure 2: The beginning of the strategy st of $A \diamondsuit B$

Observation 3.1 *Let st be a strategy of $A \diamond B$. Let st_{projX} the projection of st on X ($X = B$ or A). For the simplicity sake, we assume that $X = B$. Let st_B be a strategy derived of 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 δ^{hB}. δ^{hB} (called the weight of the cone of history $h|B$ in st_B) is the probability of the A computation steps executed in the history of $\mathcal{C}_h$. Hence, $Pr_{st}(\mathcal{C}_h) = \delta^{hB}.Pr_{st_B}(\mathcal{C}_{h|B})$.*

Definition 3.3 (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.*

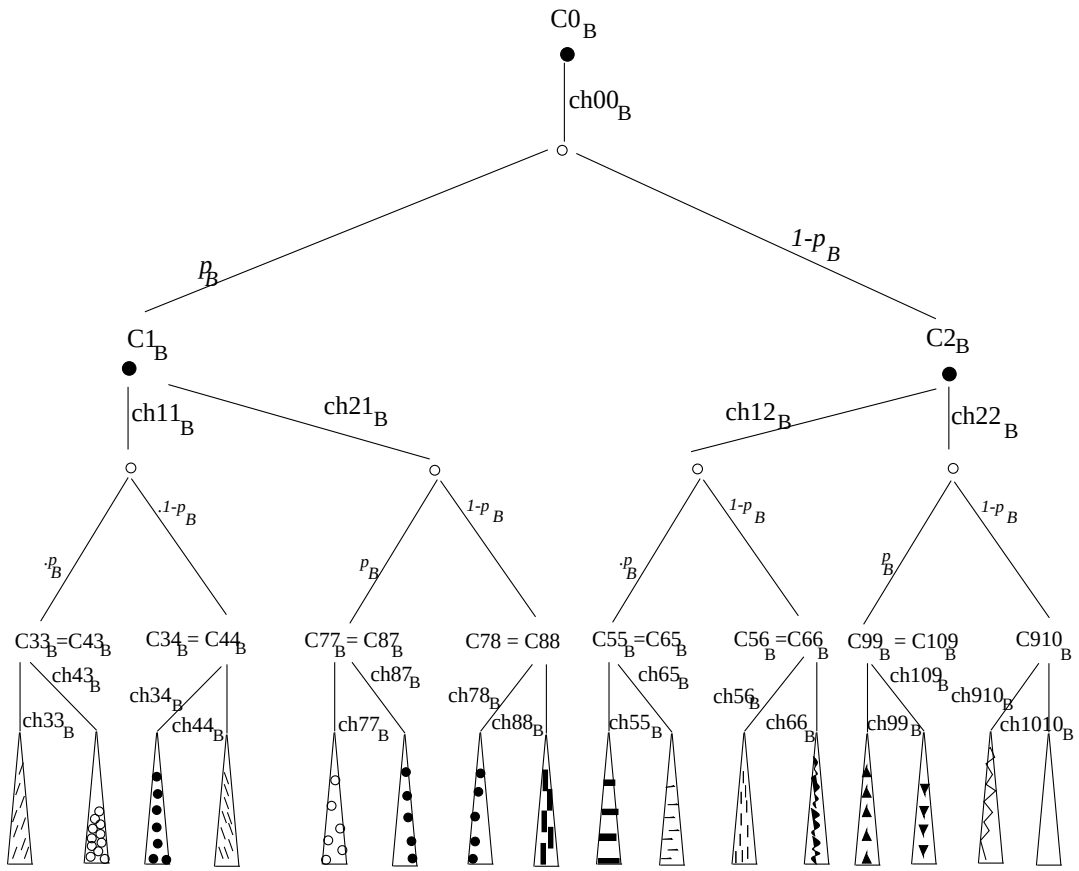

Figure 3: The beginning of the projection of the strategy st on B

For instance, the Figure 4 contains 4 projection strategies on B of the strategy *st* (Figure 2). Each strategy has a different δ_2 value.

Observation 3.2 *Let st be a strategy of $A \diamond B$. Let st_{projX} the projection of st on X ($X = B$ or A). For the simplicity sake, we assume that $X = B$. Let M^B be the set of projection strategy of st_{projB}. Let N be an integer. There is a finite subset of M^B, denoted M_N^B (called a B picture of N-length prefix of st) such that $\sum_{st_B \in M_N^B} \delta_N^{st_B} = 1$. The 4 strategies of the Figure 4 constitute the set M_2^B.*
Each N-length cones of st have one and only one projection on B in M_N^B.
There are several subsets of M_B that are "B picture of N-length prefix of st".

In the sequel, we show that if any strategy of a component, the set of computations reaching a configuration which verifies a predicate P has the probability 1 then in any strategy of the composed algorithm, this set has the probability 1.

Lemma 3.5 (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. 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} the projection of st on X ($X = B$ or A). For the simplicity sake, 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 ϵ 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 B picture of N-length st prefix) we have $Pr_{st_B}(\mathcal{EL}_N) \geq 1 - \epsilon$.
$Pr_{st}(\mathcal{EL}_N) = \sum_{st_i \in M_N^B}[Pr_{st_i}(\mathcal{EL}_N) \ .\delta_N^{st_i}] \geq (1-\epsilon).\sum_{st_i \in M_N^B} \delta_N^{st_i} \geq 1 - \epsilon$. In st, the set of computations reaching legitimate configurations in $N' \geq N$ 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 $Pr_{st}(\mathcal{EL}_{N_i}) \geq 1 - \epsilon_i$.
$\lim_{n \to \infty} P_{st}(\mathcal{EL}_n) = P_{st}(\text{computations reaching a legitimate configuration}) = 1$. □

4 Cross-over composition and self-stabilization

In the sequel, we study the propagation of the self-stabilization property from a component to the resulting algorithm of the cross-over composition. The propagation with self-stabilization (in the deterministic case) is a direct consequence of Lemmas 3.1 and 3.2 when the strong component is the propagation initiator.

Lemma 4.1 [self-stabilization propagation to the deterministic $A \diamond B$ algorithm from the B algorithm] *Let $A \diamond B$ be the cross-over composition between the deterministic components A and B. If the algorithm 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 Lemmas 3.1 and 3.2. In order to prove the convergence we apply Lemma 3.1 for the property which characterizes the legitimate configurations. The correctness proof results from Lemma 3.2 applied for the specification SP. □

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

Lemma 4.2 [self-stabilization propagation to the deterministic $A \diamond B$ algorithm from the A algorithm] *Let $A \diamond B$ be the cross-over composition between the deterministic components 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 Algorithm $A \diamond B$. 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 Lemma 3.3, e_A is a maximal computation of A and e_A reaches a legitimate configuration (A is self-stabilizing).
- correctness of Algorithm $A \diamond B$. 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 so e_A verifies the specification SP, hence e verifies also the specification SP.

□

Lemma 4.3 [self-stabilization propagation to the probabilistic $A \diamond B$ algorithm from the B algorithm] *Let $A \diamond B$ be the probabilistic cross-over composition between the components A and B. If the algorithm B self-stabilizes for the specification SP under the scheduler D then $A \diamond B$ is a probabilistic algorithm self-stabilizing for SP under D.*

Proof: Let us study the propagation for 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 the game between $A \diamond B$ and the scheduler D.

- B is deterministic. Every computation of the strategy st has a maximal projection on B. 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 Lemma 3.5 or to the Theorem 3.1. $P_{st}(\mathcal{E}L) = 1$. Moreover, according to Lemma 3.1, all computations of st that reaches L have a suffix that verifies SP.

□

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

Lemma 4.4 [self-stabilization propagation to the probabilistic $A \diamond B$ algorithm from the A algorithm] *Let $A \diamond B$ be the probabilistic cross-over composition between the components A and B. If the algorithm A self-stabilizes for the specification SP under the scheduler D and the algorithm B is a fair algorithm under D then $A \diamond B$ is a probabilistic algorithm self-stabilizing for SP under D.*

Theorem 4.1 [self-stabilization propagation from the weak and the strong algorithm] *Let $A \diamond B$ be the probabilistic cross-over composition between the components A and B. If the algorithm A self-stabilizes for the specification SP under the scheduler D and the algorithm B is self-stabilizing for the specification SR and it is fair under D then $A \diamond B$ is a probabilistic algorithm self-stabilizing for $SP \wedge SR$ under D.*

Note that in the both cases — deterministic and probabilistic — the strong component will contaminate with self-stabilization the result of composition without any restriction, while the propagation initiated by the weaker one could be realized if and only if the strong component 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 5.1 (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 of f_{pp}, p executes exactly two actions (during the first and the last step of f_{pp}).*

Lemma 5.1 (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 $A \diamondsuit B$, f has the following property (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 $A \diamondsuit B$ is k-fair (Corollary 3.2). □

Theorem 5.1 *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 a self-stabilizing algorithm 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 Lemma 5.1, 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 verifies the predicate $\mathcal{L}$), it verifies 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 Lemma 5.1 any execution of st_A is k-bounded. According to the hypothesis, $P_{st_A}(\mathcal{EL}) = 1$. According to Lemma 3.5 or to the Theorem 3.1, $P_{st}(\mathcal{EL}) = 1$.

□

Note that the transformation depends directly on the properties of the strong component of a cross-over composition. The main question is if there are algorithms able to verify 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.**
[GH99]	general networks, bidirectional	with id	central to unfair
[BDGM00]	general networks, bidirectional	with id	central to unfair
[BDGM00]	general networks, bidirectional	with id	X_1-bounded to unfair
[BGJ99a]	rings, unidirectional	anonymous	X_1-bounded to unfair
[BDLGJ]	general networks, unidirectional	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 [BDGM00] and [GH99] 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 [BDGM00] 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-fair, fair, ...) into algorithms which maintain the self-stabilizing property under unfair scheduler.

The key of this transformation is the cross-over composition $A \diamond B$: roughly, 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 (regarding the self-stabilizing algorithm A) 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

[BDGM00] J. Beauquier, A. Datta, M. Gradinariu, and F. Magniette. Self-stabilizing local mutual exclusion and daemon refinement. *DISC'2000*, pages 223–237, 2000.

[BDLGJ] J. Beauquier, J. Durand-Lose, M. Gradinariu, and C. Johnen. Token based self-stabilizing uniform algorithms. *Raport technique no.1250, LRI, Université Paris Sud; à paraître dans The Chicago Journal of Theoretical Computer Science.*

[BGJ99a] J. Beauquier, M. Gradinariu, and C. Johnen. Randomized self-stabilizing and space optimal leader election under arbitrary scheduler on rings. *Tech. Rep. 99-1225 Universite Paris Sud*, 1999.

[BGJ99b] J. Beauquier, M. Gradinariu, and C. Johnen. Randomized self-stabilizing optimal leader election under arbitrary scheduler on rings. Technical Report 1225, Laboratoire de Recherche en Informatique, September 1999.

[DH99] S. Dolev and T. Herman. Parallel composition of stabilizing algorithms. In *Proceedings of the fourth Workshop on Self-Stabilizing Systems*, pages 25–32, 1999.

[GH91] M Gouda and T Herman. Adaptive programming. *ieeetse*, 17:911–921, 1991.

[GH99] M. Gouda and F. Haddix. The alternator. *In Proceedings of the Third Workshop on Self-Stabilizing Systems (published in association with ICDCS99 The 19th IEEE International Conference on Distributed Computing Systems)*, pages 48–53, 1999.

[Seg95] R. Segala. *Modeling and Verification of Randomized Distributed Real-time Systems.* PhD thesis, MIT, Departament of Electrical Engineering and Computer Science, 1995.

[SL94] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. In Springer-Verlag, editor, *CONCUR '94, Concurrency Theory, 5th International Conference , LNCS:836*, Uppsala, Sweden, August 1994.

[Var97] G. Varghese. Compositional proofs of self-stabilizing protocols. In Carleton University Press, editor, *Proceedings of the Third Workshop on Self-stabilizing Systems*, pages 80–94, 1997.

[WSS94] S. H. Wu, S. A. Smolka, and E. W. Stark. Composition and behaviors of probabilistic i/o automata. In *fifth International Conference Concurrency theory LNCS:836 (CONCUR'94)*, pages 513–528, 994.

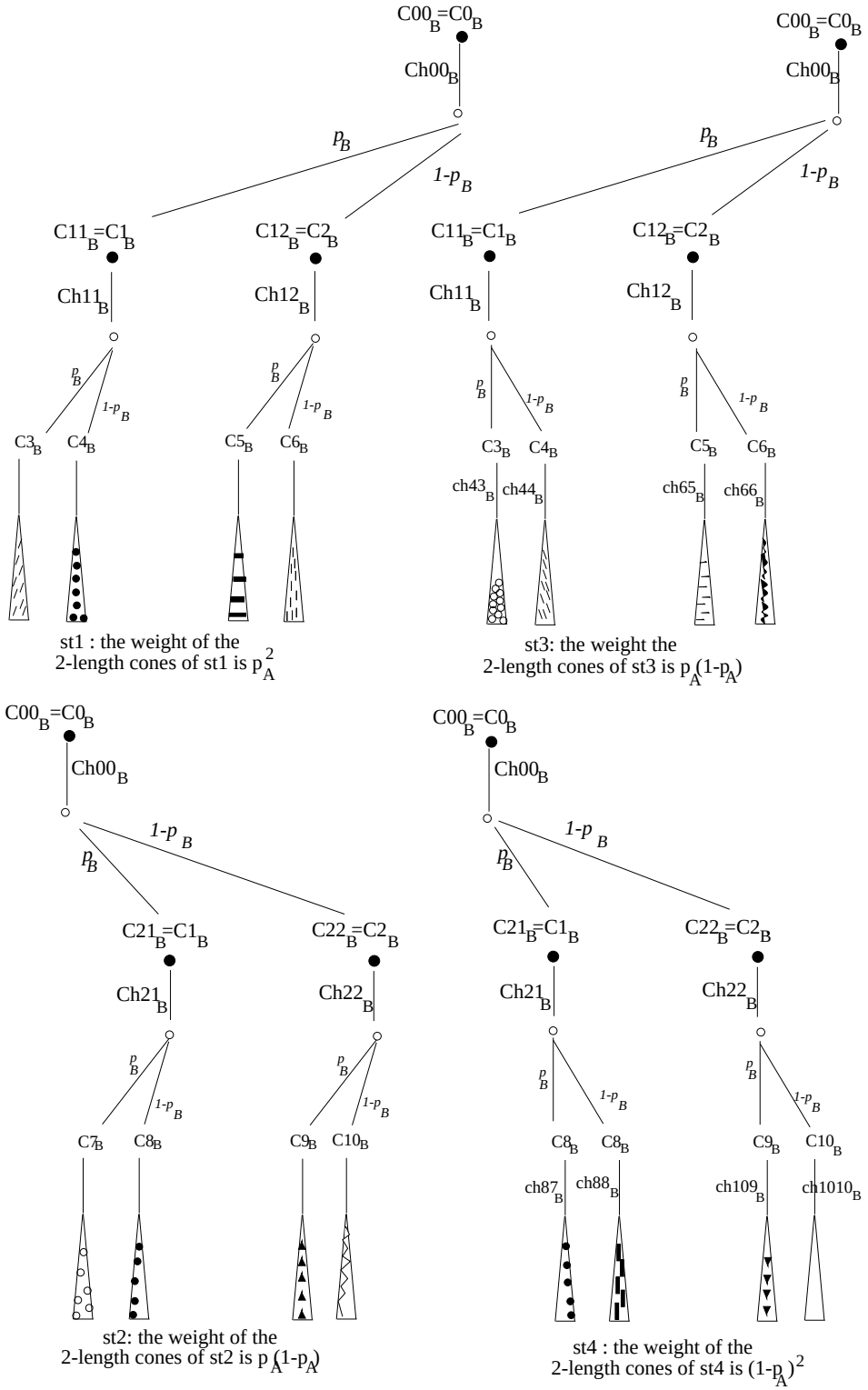

Figure 4: The 2-length beginning of all projection strategies on B