1 Content of the Technical Report

This technical report is a companion document for the article *Concurrency as a Random Number Generator*. Section 2 presents a detailed description of the schedule reconstruction algorithm CO-Rec while Section 3 proves that if the real scheduler is Markovian and non-degenerate, then the trace computed by CO-Rec is also Markovian and non-degenerate.

2 Schedule Reconstruction: CO-Rec

In this section, we give more details about the CO-Rec algorithm. Figure 1a recalls the reconstruction graph used by CO-Rec to describe the interleaving of memory accesses generated by CO-Obs whose pseudo-code appears in Figure 1b.

Algorithm. The purpose of CO-Rec is to rebuild the schedule of the shared memory operations of the two threads of CO-Obs. CO-Obs ensures that there is at least one sequential order that is consistent with the observations it produces. But some observations can be explained by several total orders on the shared memory operations of the two threads.
However, for each possible execution of Co-Obs, the set of possible sequential explanations is exactly captured by a path of length $N$ in the reconstruction graph of Figure 1a. Note that this path starts necessarily in one of the three states on the left ([$w_0$|$w_1$], [$w_0$ · $r_0$] or [$w_1$ · $r_1$]) since both threads of Co-Obs start by writing (Line 3 of Figure 1b). Reciprocally, assuming complete asynchrony, any valid path (starting in one of these states and containing $N$ occurrences of each of the symbols $w_0$, $w_1$, $r_0$ and $r_1$) could theoretically be observed by Co-Obs.

The reconstruction algorithm underlying Co-Rec is presented in Figure 2. The rebuilt path is stored in the array denoted by $trace[0, \ldots, N]$. The variable $n_0$ (resp. $n_1$) stores the number of rounds that have been rebuilt so far for $P_0$ (resp. $P_1$). Accordingly, $n_0$ (resp. $n_1$) is also the number of the next round to rebuild for $P_0$ (resp. $P_1$). The length of the rebuilt path in the reconstruction graph of Figure 1a is consequently $n_0 + n_1$, i.e., the total number of rounds rebuilt.

The reconstruction loop (Lines 3 to 30) runs until $N$ rounds are rebuilt for each thread. Each iteration of the reconstruction loop always rebuilds an integral number\(^1\) of rounds for each thread. Thus, at the beginning of each iteration of the reconstruction loop, both threads are about to write in the rebuilt schedule. In particular, the last state inserted (in $trace[n_0 + n_1 - 1]$) is either [$r_0$|$r_1$], [$w_0$ · $r_0$] or [$w_1$ · $r_1$].

**Solo prefix.** The conditions of lines 4 and 7 capture the possibility that one of the threads executes one or several rounds before the first write of the other thread. Note that it is impossible for these two conditions to hold simultaneously since this would mean that the two threads have performed at least one round each without seeing each other: this is impossible because no sequential ordering of the shared memory operations could explain that. In addition, once one thread has completed one round, all the values read by the other thread afterwards are non-$\bot$. Once each of them sees the other, we have $obs_0[n_0] \neq \bot \land obs_1[n_1] \neq \bot$ for the remaining of the execution.

**Solo write-then-read.** At the beginning of each iteration of the reconstruction loop, the following is verified: in the rebuilt schedule $P_0$ and $P_1$ are respectively about to write round numbers $n_0$ and $n_1$ (unless one of them has reached $N$). They then proceed to read and store the result in $obs_0[n_0]$ and $obs_1[n_1]$.

\(^1\)no half-rounds for instance
If the condition of Line 10 is verified, then the next read by $P_0$ precedes the next write by $P_1$. Consequently, Co-Rec appends the state $[w_0 \cdot r_0]$ to the rebuilt schedule, increases $n_0$, the number of iterations rebuilt for $P_0$, and proceeds to the next iteration of the reconstruction loop. A symmetric behavior is triggered if the condition of Line 13 is fulfilled.

**Parallel writes.** If, at the beginning of an iteration of the reconstruction loop, both $obs_0[n_0] \geq n_1$ and $obs_1[n_1] \geq n_0$ are verified, then, in the next steps of the rebuilt schedule, $P_0$ and $P_1$ both wrote (respectively $n_0$ and $n_1$) before any of them read. To capture the uncertainty about the ordering of these two write operations, Co-Rec then appends $[w_0||w_1]$ to trace at Line 17.

**Eventual parallel reads.** At that point in the reconstructed schedule (Line 18), $P_0$ and $P_1$ are both about to read the memory location of each other, so the next state to append to the trace is either $[r_0||r_1]$, $[r_0 \cdot w_0]$ or $[r_1 \cdot w_1]$. Since both threads finish their round of Co-Obs by reading, i.e., finish in one of the states $[w_0 \cdot r_0]$, $[w_1 \cdot r_1]$ or $[r_0||r_1]$ of Figure 1a, we know that we will have to append the state $[r_0||r_1]$ at least once before the end of the trace. After this happens for the first time (Line 27), both threads are, again, about to write and we end the current iteration of the reconstruction loop. Note that both $n_0$ and $n_1$ are incremented at Line 28 to capture the fact that the states $[w_0||w_1]$ and $[r_0||r_1]$ appended at Lines 17 and 27, once combined, constitute a full iteration of Co-Obs loop for each thread.

**Solo read-then-write.** When the eventual parallel reads occur, each thread observes precisely the value the other just wrote: $obs_0[n_0] = n_1$ and $obs_1[n_1] = n_0$. Before that, any sequence of the two states $[r_0 \cdot w_0]$ and $[r_1 \cdot w_1]$ may have happened. The loop of Lines 18 to 26 captures these sequences. If $P_1$ observed a value strictly greater than what $P_0$ just wrote (condition Line 19), then $P_0$ (read and) wrote $obs_1[n_1] - n_0$ times before the next read by $P_1$. Co-Rec then consequently appends $[r_0 \cdot w_0]$ to the trace and increases $n_0$. This procedure and its symmetric are repeated until both threads are about to read in parallel in the rebuilt schedule.

After the whole schedule is rebuilt, the potential solo runs at the beginning and end of the trace (repetitions of one of the states $[w_0 \cdot r_0]$ or $[w_1 \cdot r_1]$) are discarded and the trace is passed to BLUM-ÉLIAS that uses it to extract a random binary sequence. In practice, these solo runs are essentially due to the fact that the threads may not start exactly at the same time and may progress at slightly different speeds.

### 3 Markovian Scheduler Implies Markovian Trace

We model the scheduler as a random sequence $S = (S_t)_{t\in\mathbb{N}}$ where $S_t = P_i$ if the thread $P_i$ takes a step (read or write operation) at time $t$. Let $X = (X_\tau)_{\tau\in\mathbb{N}}$ be the corresponding trace in the reconstruction graph of Figure 1a.

**Proposition 1.** If $S$ is Markovian with lag $l$, i.e., the conditional probabilities satisfy

$$\mathbb{P}(S_{t+1}|S_t \ldots S_0) = \mathbb{P}(S_{t+1}|S_t \ldots S_{t-l+1})$$

then $X$ is Markovian with lag $\lfloor l/2 \rfloor$.

**Proof.** Without loss of generality, we assume that the lag of $S$ is even, $l = 2\lambda$. We first introduce an auxiliary process $Z = (Z_\tau)_{\tau\in\mathbb{N}}$ with values in $\mathbb{Z}/2$. In addition, we add the relations

$$Z_0 = 0, \quad Z_{\tau+1} = Z_\tau + \mathbb{I}(S_{2\tau} \neq S_{2\tau+1})$$

\begin{align*}
\mathbb{P}(S_{t+1}|S_t \ldots S_0) &= \mathbb{P}(Z_{t+1}|Z_t \ldots Z_0) \\
&= \mathbb{P}(Z_{t+1}|Z_t \ldots Z_{t-l+1}) \\
&= \mathbb{P}(Z_{t+1}|Z_t \ldots Z_{t-l+1})
\end{align*}
where \( I \) is the indicator function. Intuitively, \( Z_\tau = 0 \) (resp. \( Z_\tau = 1 \)) means that threads \( P_0 \) and \( P_1 \) are both about to write (resp. read) at time \( 2\tau \). If \( S_{2\tau} = S_{2\tau+1} \) then, at time \( 2\tau + 2 \), threads \( P_0 \) and \( P_1 \) are again about to write (resp. read); otherwise, they are about to read (resp. write).

We define another process \( Y = (Y_\tau)_{\tau \in \mathbb{N}} \) where \( Y_\tau \) denotes the triple \((Z_\tau, S_\tau, S_{\tau+1})\). By construction, the process \( Y \) has the Markov property. Indeed, \( Z_{\tau+1} \) depends only on \( Y_\tau \), and the couple \((S_{2\tau+2}, S_{2\tau+3})\) depends only on \( S_{2\tau+1} \) down to \( S_{2\tau+1-\epsilon} \). In particular, \( Y \) has lag \( \lambda \).

We have, for all \( \tau \), \( X_\tau = f(Y_\tau) \) where \( f \) is defined by

\[
\begin{align*}
(0, P_0, P_1) &\mapsto [w_0 | w_1] \\
(0, P_1, P_0) &\mapsto [w_0 | w_1] \\
(0, P_0, P_0) &\mapsto [w_0 \cdot r_0] \\
(0, P_1, P_1) &\mapsto [w_1 \cdot r_1] \\
(1, P_0, P_1) &\mapsto [r_0 | r_1] \\
(1, P_1, P_0) &\mapsto [r_0 | r_1] \\
(1, P_0, P_0) &\mapsto [r_0 \cdot w_0] \\
(1, P_1, P_1) &\mapsto [r_1 \cdot w_1]
\end{align*}
\]

In particular, \( X \) is Markovian with lag \( \lambda = 1/2 \).

**Proposition 2.** If \( S \) is Markovian with lag \( l \) and if for all \( t \in \mathbb{N} \) and \( i \in \{0, 1\} \) we have

\[ \mathbb{P}(S_{t+1} = P_i | S_t \ldots) < 1, \]

then \( X \) is a non-degenerate Markov process, i.e. for all \( t \in \mathbb{N} \) and all possible states \( X \) the transition probabilities of the process \( X \) verify

\[ \mathbb{P}(X_{t+1} = x | X_t \ldots) < 1. \]

**Proof.** Let us consider the same notations and definitions as above for \( Z, Y \) and \( f \). We fix a path \((x_t)_{t \in \mathbb{N}}\). We denote by \( y_t \) the set \( f^{-1}(x_t) \). We define \( \pi_Z(z, s, s') = z \) and \( \pi_S(z, s, s') = (s, s') \).

\[
\mathbb{P}(X_{t+1} = x_{t+1} | X_t = x_t \ldots) = \mathbb{P}(Y_{t+1} = y_{t+1} | Y_t = y_t \ldots)
\]

\[ = \mathbb{P}
\begin{pmatrix}
Z_{t+1} \in \pi_Z y_{t+1} \\
(S_{2t+2}, S_{2t+3}) \in \pi_S y_{t+1} \\
S_{2t}, S_{2t+1} \in \pi_S y_t \ldots \\
\end{pmatrix}
\]

Note that each \( Z_t, t > 0 \) is entirely determined by the sequence \( S_0, \ldots, S_{2t-1} \) and that \( Z_0 = 0 \). Hence we can remove them from the conditional part.

\[
\mathbb{P}(X_{t+1} = x_{t+1} | X_t = x_t \ldots) = \mathbb{P}
\begin{pmatrix}
Z_{t+1} \in \pi_Z y_{t+1} \\
(S_{2t+2}, S_{2t+3}) \in \pi_S y_{t+1} \\
S_{2t}, S_{2t+1} \in \pi_S y_t \ldots \\
\end{pmatrix}_{W}
\leq \mathbb{P}((S_{2t+2}, S_{2t+3}) \in \pi_S y_{t+1} | W)
\]

We now distinguish two cases. Either \( |\pi_S y_{t+1}| = 1 \), or \( \pi_S y_{t+1} = \{(P_0, P_1), (P_1, P_0)\} \). In the first case, \( \pi_S y_{t+1} = \{(P_i, P_j)\} \) and we have:

\[
\mathbb{P}((S_{2t+2}, S_{2t+3}) \in \pi_S y_{t+1} | W) = \frac{\mathbb{P}(S_{2t+2} = P_i | W) \cdot \mathbb{P}(S_{2t+3} = P_j | W, S_{2t+2} = P_i)}{< 1} < 1.
\]
In the second case, $\pi_{Syt+1} = \{(P_0, P_1), (P_1, P_0)\}$. 

$$
P((S_{2t+2}, S_{2t+3}) \in \pi_{Syt+1}|W) = P(S_{2t+2} = P_0|W) \cdot P(S_{2t+3} = P_1|W, S_{2t+2} = P_0)
+ \frac{P(S_{2t+2} = P_1|W) \cdot P(S_{2t+3} = P_0|W, S_{2t+2} = P_1)}{1 - P(S_{2t+2} = P_0|W)}
< 1
\]