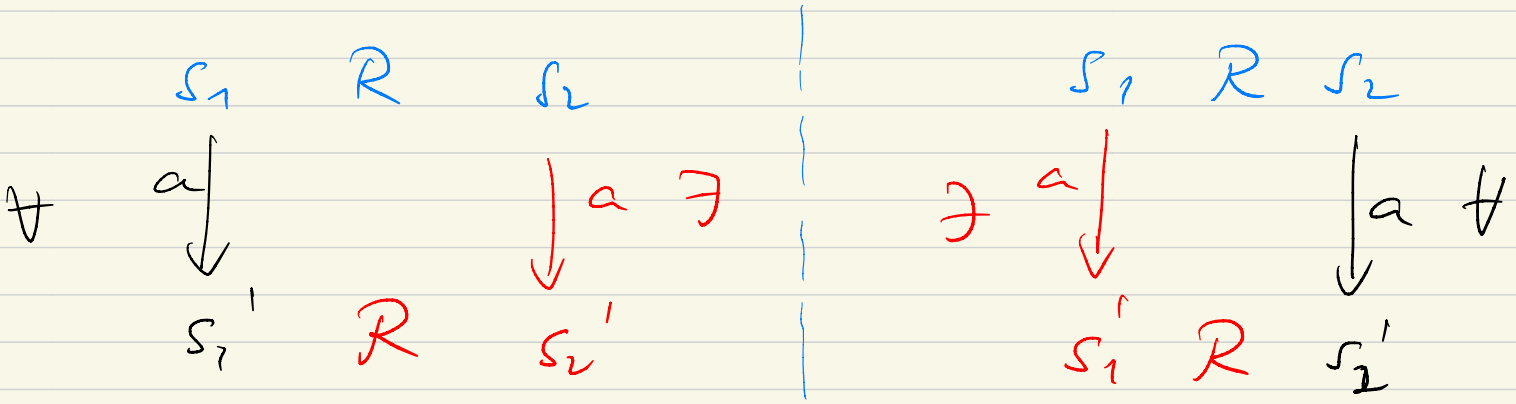
Cours 10/3/21

Sébastien Delpeuch · delpeuch.net · CC BY-NC-SA 4.0

Bisimulation $R \subseteq S_1 \times S_2$

$\rightsquigarrow$ comparaison de systèmes de transition (ST) : T_1, T_2

$T_i = (S_i, \Sigma, \delta_i, q_0^i)$



Ex : Bakery algorithm (exclusion mutuelle)

Proc 1 — Init : $x_1 = x_2 = 0$ — Proc 2

symm.

while true do

n_1 : $x_1 := x_2 + 1$

w_1 : wait until $(x_2 = 0$ OR $x_1 < x_2)$

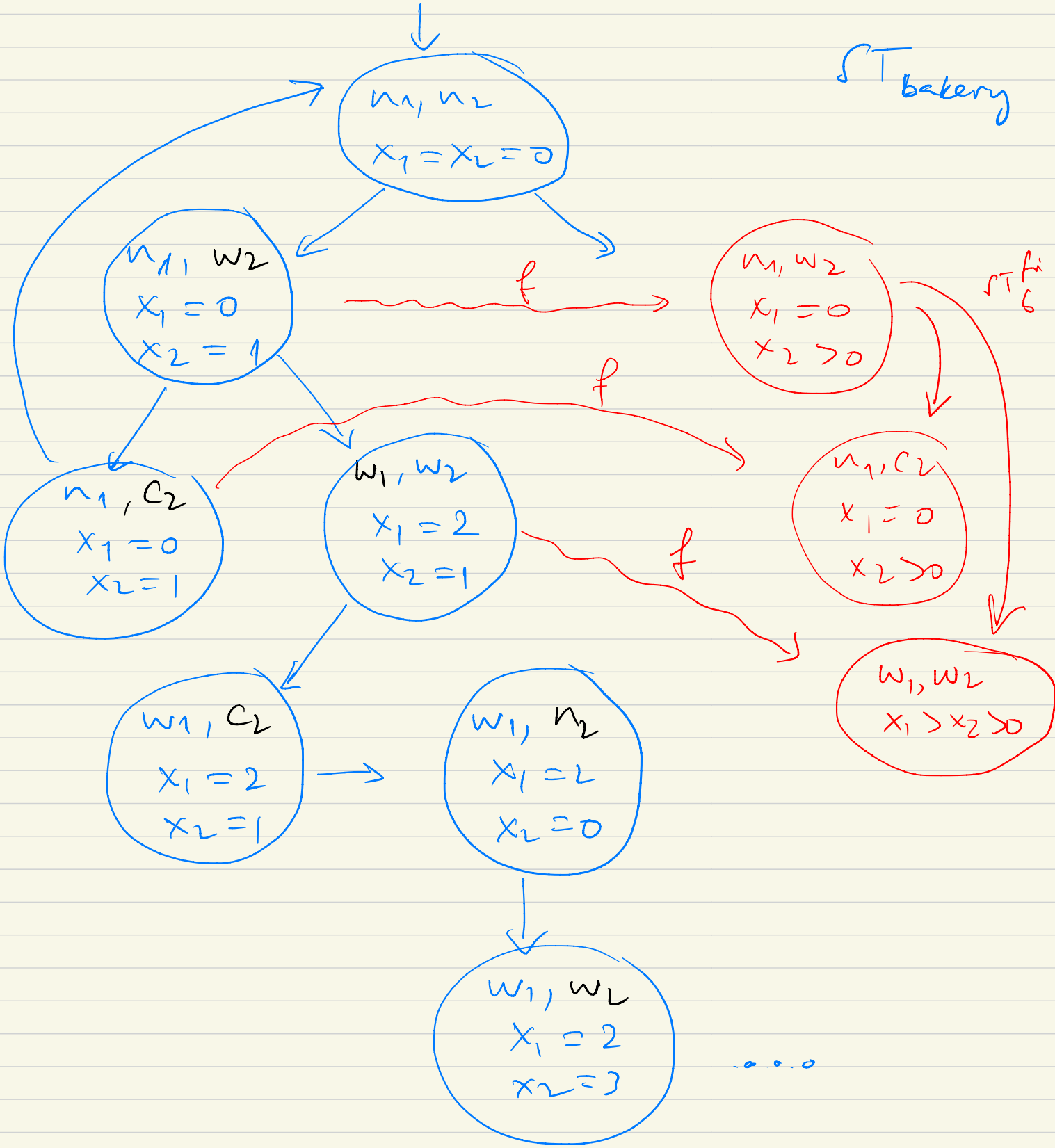
critical section

c_1 : $x_1 := 0$;

ok

Sébastien Delpeuch · delpeuch.net · CC BY-NC-SA 4.0

le ST associé est infini :



Sébastien Delpeuch · delpeuch.net · CC BY-NC-SA 4.0

Mais il ex. un ST fini ST^{fin}_{bakery}

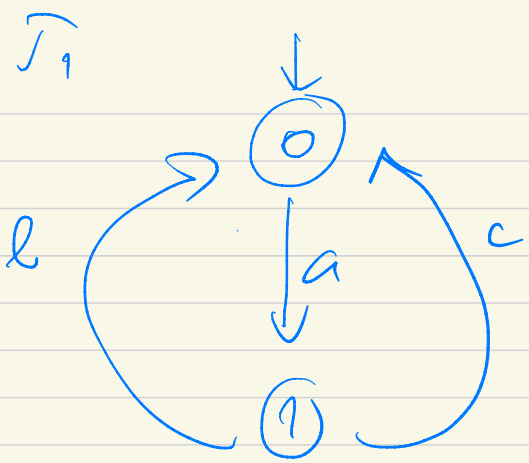
tq. $ST_{bakery} \sim_{bisim} ST^{fin}_{bakery}$

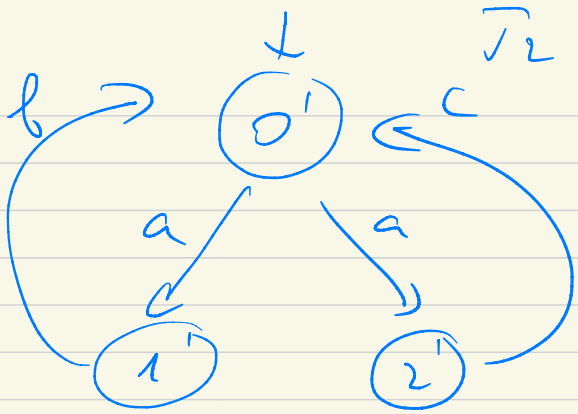
Etat $s = (l_1, l_2, x_1 = b_1, x_2 = b_2)$ de

ST_{bakery}, $l_i \in \{n_i, w_i, c_i\}$

$$f(s) = \begin{cases} (l_1, l_2, x_1 = 0, x_2 = 0) & \text{si } b_1 = b_2 = 0 \\ (l_1, l_2, x_1 = 0, x_2 > 0) & \text{si } b_1 = 0, b_2 > 0 \\ (l_1, l_2, x_1 > 0, x_2 = 0) & \text{si } b_1 > 0, b_2 = 0 \\ (l_1, l_2, x_1 > x_2 > 0) & \text{si } b_1 > b_2 > 0 \\ (l_1, l_2, x_2 > x_1 > 0) & \text{si } b_2 > b_1 > 0 \end{cases}$$

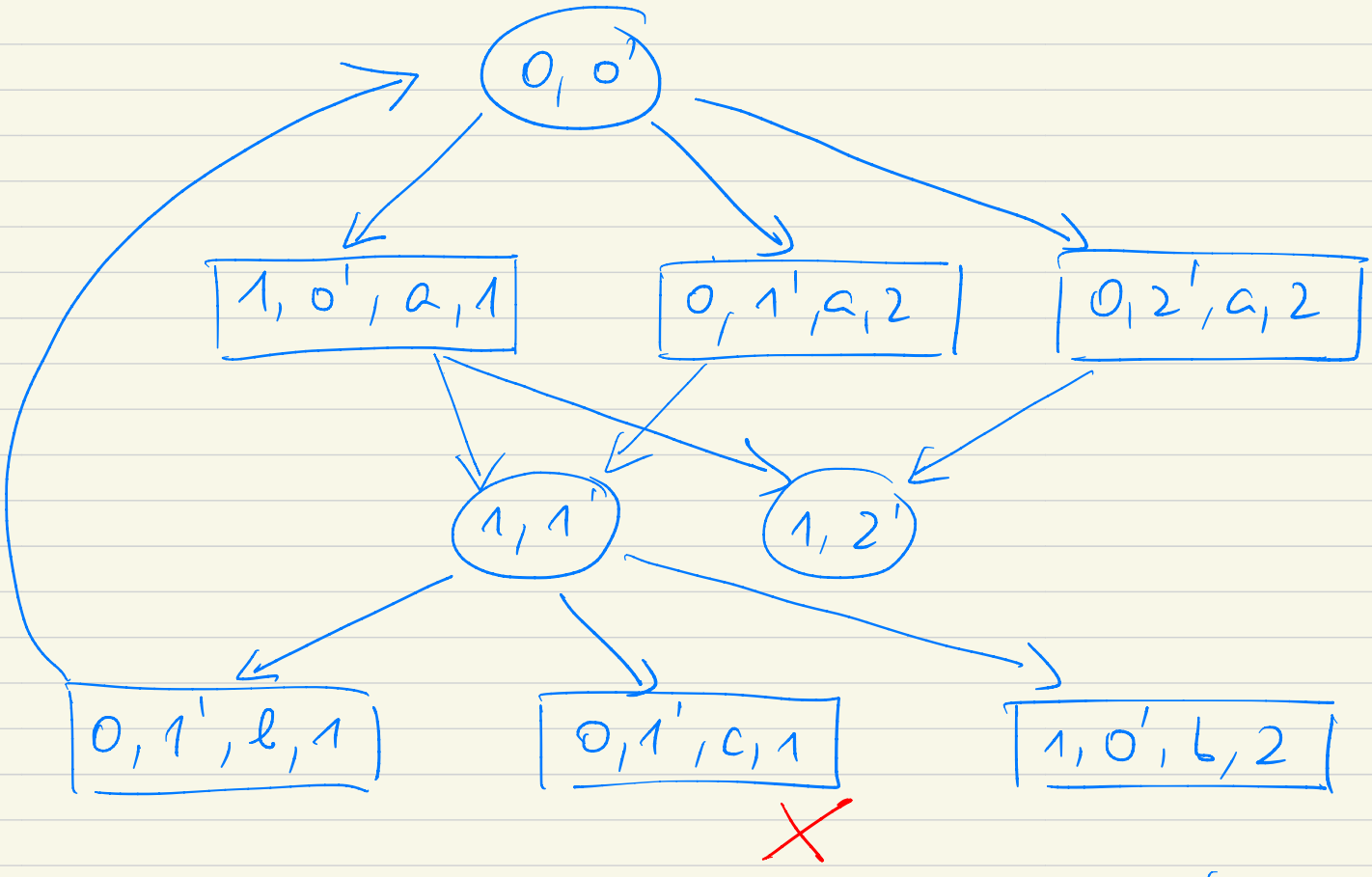
Sébastien Delpeuch · delpeuch.net · CC BY-NC-SA 4.0





$F_1 = \{0\}$ $\quad$ $F_2 = \{0'\}$

$L(\mathcal{T}_1) = L(\mathcal{T}_2) = (ab + ac)^*$



Spoiler gagné en essayant d'aller vers $\langle 0, 1', c, 1 \rangle$ ou $\langle 0, 2', b, 1 \rangle$

→ Jeu d'accessibilité

Sébastien Delpeuch · delpeuch.net · CC BY NC-SA 4.0

Ça donne un algo polynomial pour savoir si deux ST sont bisimulatrices.

[Il y a bcp d'algorithmes très efficaces pour la bisimulation.

Complexité : $O(|T_1| \cdot |T_2|)$ pour $|T_1|, |T_2| \leq n \rightarrow O(n \log n)$]

Sébastien Delpeuch · delpeuch.net · CC BY-NC-SA 4.0

Aujourd'hui : comment résoudre des jeux dont la cond. de victoire est une combinaison booléenne de cond. d'accessibilité :

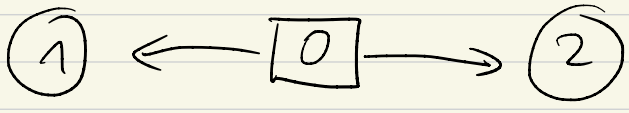
ex Po veut visiter F, tout en évitant B. $F, B \subseteq V$.

$Reach(F) \cap Avoid(B)$

Préparation

Q Etant donnés $F_1, F_2 \subseteq V$

?? $Attro(F_1 \cup F_2) \stackrel{?}{=} Attro(F_1) \cup Attro(F_2)$??

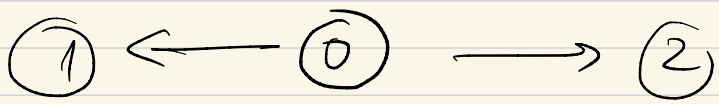
P 

$F_1 = \{1\}$
$F_2 = \{2\}$

$Attro(F_1) = \{1\}$
$F_2 = \{2\}$
$Attro(F_1 \cup F_2) = \{0, 1, 2\}$

Sébastien Delpeuch · delpeuch.net · CC BY-NC-SA 4.0

Rq $Attro(F_1) \cap Attro(F_2) \neq Attro(F_1 \cap F_2)$



$F_1 = \{1\}$

$F_2 = \{2\}$

$Attro(F_1) = \{0, 1\}$

$Attro(F_2) = \{0, 2\}$

$Attro(F_1 \cap F_2) = \emptyset$

ex $A = (V_0, V_1, E)$ $F_1, F_2 \subseteq V$

$Reach(F_1) \cap Reach(F_2)$

Une partie $\pi = (v_0, v_1 \ldots)$ est gagnée par P_0 s'il ex. i, j tq.

$v_i \in F_1$, $v_j \in F_2$

solution : construire une arène plus grande sur laquelle on se souvient si on a déjà visité F_1 (ou F_2).

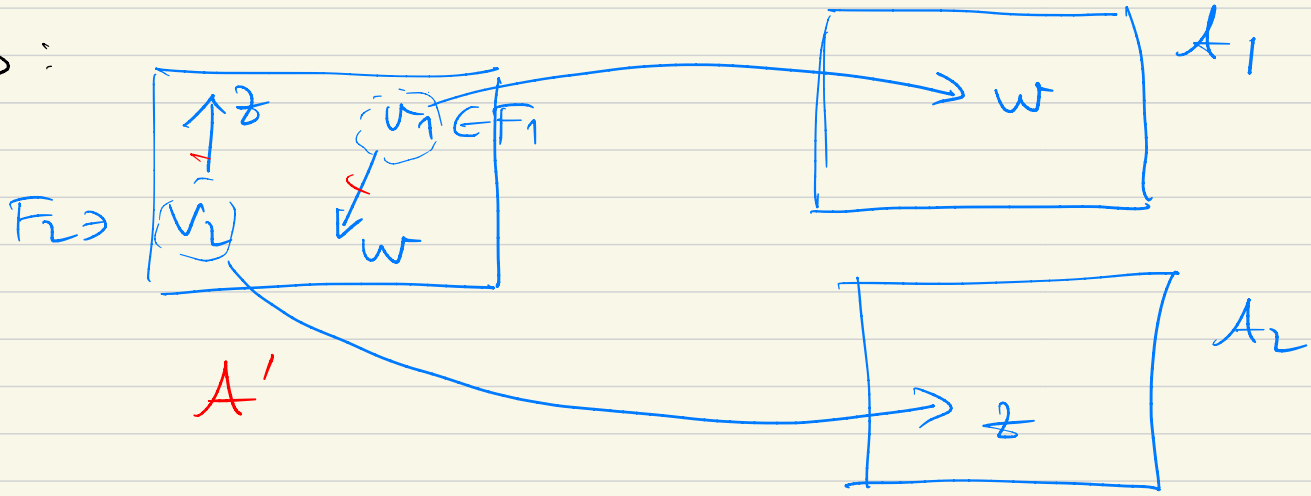
Et utiliser un jeu d'accessibilité sur l'arène plus grande.

→ réduction de jeu

Sébastien Delpeuch · delpeuch.net · CC BY-NC-SA 4.0

On suppose : $F_1 \cap F_2 = \emptyset$

$\mathcal{B}$:



A_1, A_2 : copies de A

A' : A où les arcs sortants de $F_1 \cup F_2$ sont redirigés vers A_1, resp. A_2

Sur $\mathcal{B}$ on joue jeu d'accessibilité

$$\mathrm{Reach}\left(\underbrace{F_2^{(1)}}_{\subseteq A_1} \cup \underbrace{F_1^{(2)}}_{\subseteq A_2}\right)$$

$\mathcal{B}$:

$$V' = V \cup (V \times \{1\}) \cup (V \times \{2\})$$

$$E' = \Big(E \cap \big(V \setminus (F_1 \cup F_2)\big) \times V \Big) \cup E^{(1)} \cup E^{(2)} \cup$$

$$\{(u, v^{(1)}) : u \in F_1, (u,v) \in E\} \cup$$

$$\{(u, v^{(2)}) : u \in F_2, (u,v) \in E\}$$

Sébastien Delpeuch · delpeuch.net · CC BY-NC-SA 4.0

$X = Attr_0(F_1^{(2)} \cup F_2^{(1)})$ ⟵ dans le jeu $\mathcal{B}$

$v \in X \cap V \iff$ dans le jeu original, Po gagne à partir de v pour la condition $Reach(F_1) \cap Reach(F_2)$

Deux jeux $\mathcal{G}_1$, $\mathcal{G}_2$. $\mathcal{G}_i = (A_i, Win_i)$ (arène)

$f : V(A_1) \to V(A_2)$.

f est <u>réduction de $\mathcal{G}_1$ vers $\mathcal{G}_2$</u> ssi pour tout $v \in V(A_1)$:

v gagnant pour Po par rapport à Win_1 ⟹

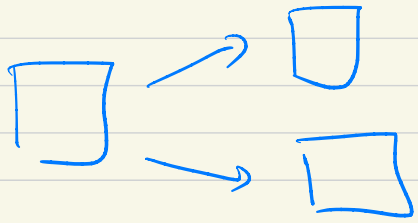
$f(v)$ gagnant pour Po par rapport à Win_2

Sébastien Delpeuch · delpeuch.net · CC BY-NC-SA 4.0

Dans l'exemple :

A_1 = arène initiale A

σ_0, σ_1 ↑ σ_0', σ_1'

A_2 = 

$f(v) = v$ → dans A'

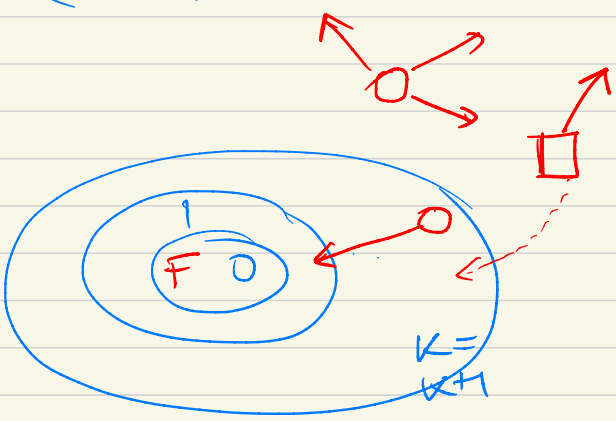
$|$

$V(A)$

En résolvant le jeu (A_2, Win_2) on obtient une stratégie gagnante pour P_0, (à partir des sommets gagnants)

$(A_2, Win_2) \longrightarrow$ W_0', W_1' } régions gagnantes

σ_0', σ_1' stratégies gagnantes



σ_0' : strat. d'attracteur.

σ_1' : strat. de piège

Sébastien Delpeuch · delpeuch.net · CC BY-NC-SA 4.0

(A_1, W_{in_1}) $\xrightarrow{f}$ (A_2, W_{in_2})

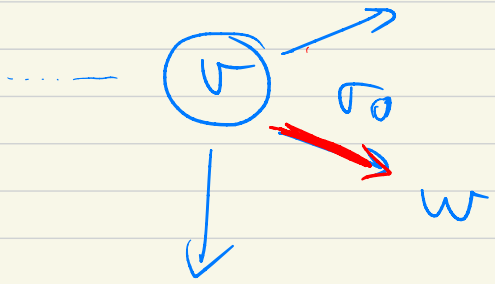
W_0, W_1 $\longleftarrow$ W_0', W_1'

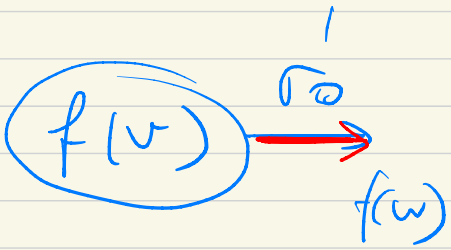
τ_0, τ_1 τ_0', τ_1'

$$W_0 = \{ v \in V(A_1) : f(v) \in W_0' \}$$

$$W_1 = \{ v \in V(A_1) : f(v) \in W_1' \}$$

τ_0





$\tau_0(v) = w$ si $\tau_0'(f(v)) = f(w)$

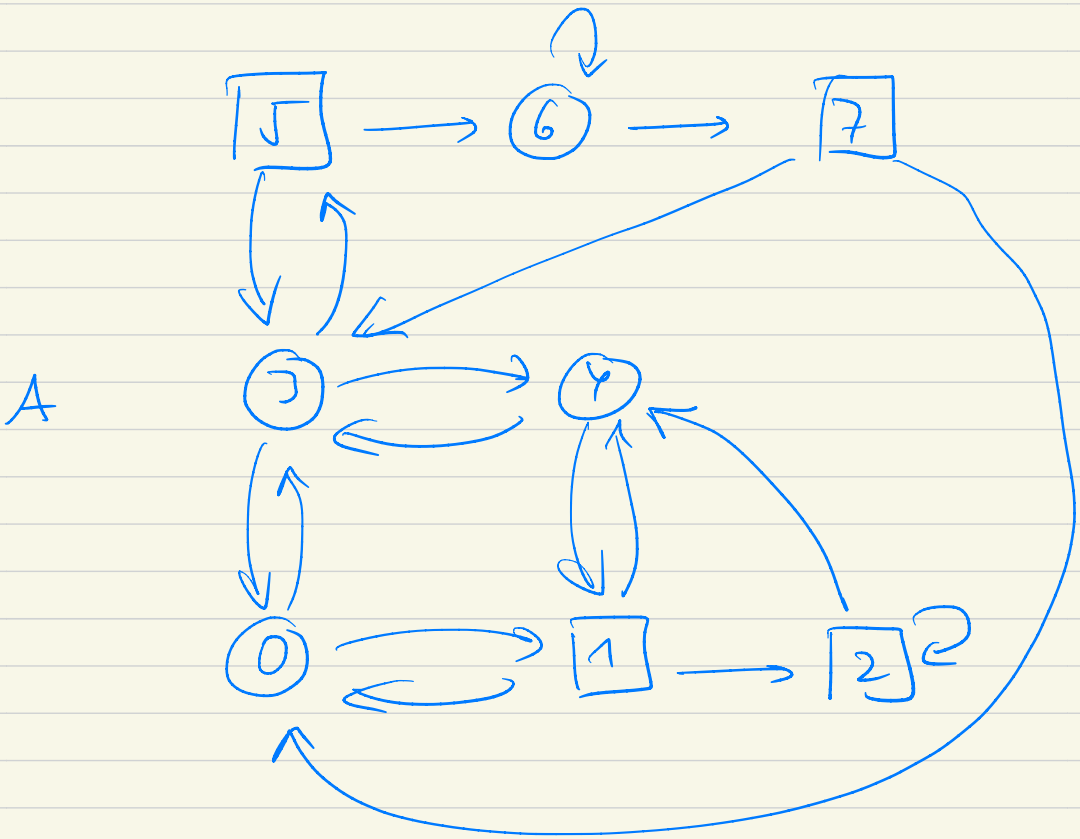
Sébastien Delpeuch · delpeuch.net · CC BY-NC-SA 4.0

$Reach(3,4) \cap Reach(0,6)$

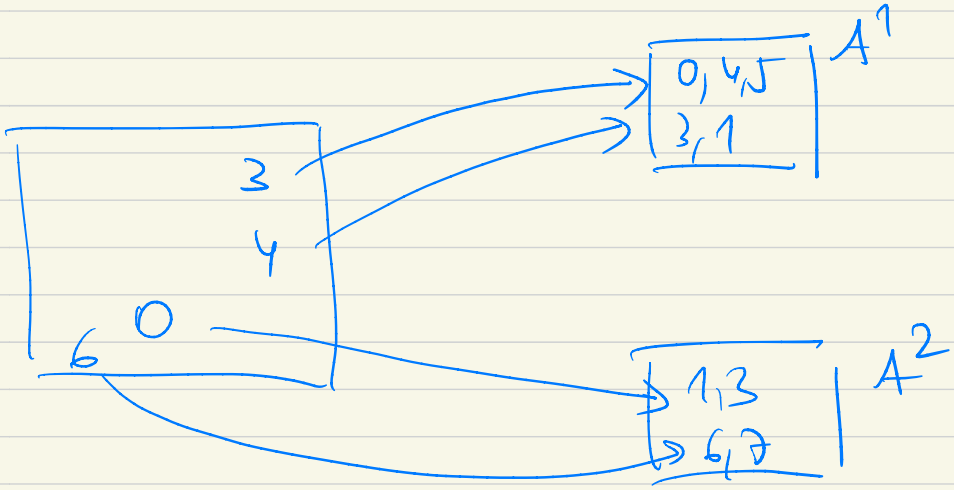
↓

σ_0 utilise de la mémoire

$7 \in W_0$



$Reach(\{3,4\}) \cap Reach(\{0,6\})$



$A' \cup A' \cup A^2$

B

A'

↓

$\sigma'_0((3,1)) = (0,1)$

Dès que P_0 attend 3

→ bit de mémoire 1

$\sigma_0(3,1) = 0$

Sébastien Delpeuch · delpeuch.net · CC BY-NC-SA 4.0

Rq : dans le jeu B : P_0 a stratégie sans mémoire (strat. d'attracteur). σ_0'

$\sigma_0' \to \sigma_0$ est une strat. gagnante avec mémoire

$\sigma_0 : V_0 \times \{0,1,2\} \to V$

- ni F_1, ni F_2
- F_1, pas F_2
- F_2, pas F_1

$\sigma_0'(v) = (w, 1)$, $v \in A'$

$\sigma_0(v, 0) = w$

↓ mémoire

update $0 \to 1$

Sébastien Delpeuch · delpeuch.net · CC BY-NC-SA 4.0

Stratégies (pour P_0) :

$$\sigma_0 : \underbrace{V^*}_{\text{historique}} V_0 \to V$$

Stratégies sans mémoire :

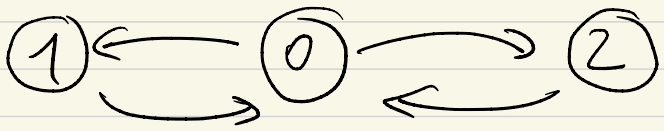
$$\sigma_0 : V_0 \to V$$

Stratégies avec mémoire finie M

$$\sigma_0 : V_0 \times M \to V$$
$$\text{update} : V_0 \times M \to M$$

Q Est-ce que P_0 a besoin de mémoire pour gagner un jeu $\text{Reach}(F_1) \cap \text{Reach}(F_2)$?

Oui :



$F_1 = \{1\}$

$F_2 = \{2\}$

mémoire $0, 1, 2$

$\sigma_0(0,0) = 1$, $\text{upd}(0,0) = 1$

$\sigma_0(0,1) = 2$, $\text{upd}(0,1) = 2$

Sébastien Delpeuch · delpeuch.net · CC BY-NC-SA 4.0

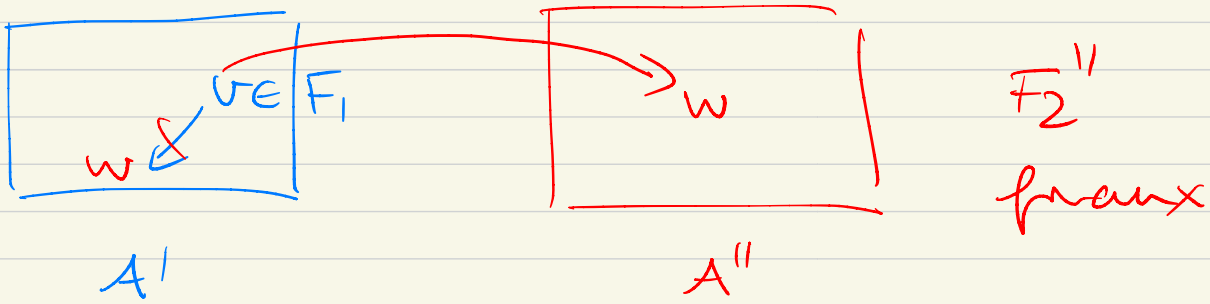
$F_1 \cap F_2 = \emptyset$

Win : voir F_1, et ensuite F_2

$\pi = v_0, v_1, \ldots$ est gagnante pour

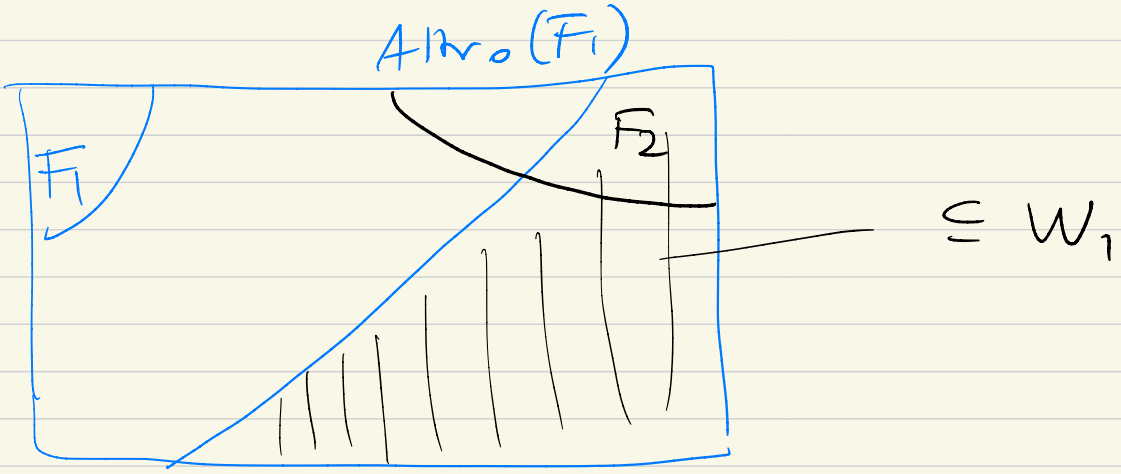
P_0 ssi il ex. $i < j$ tq.

$v_i \in F_1$, $v_j \in F_2$



P_0 a toujours besoin de mémoire

Mais : est-ce vrai aussi pour P_1 ?



Sébastien Delpeuch · delpeuch.net · CC BY-NC-SA 4.0