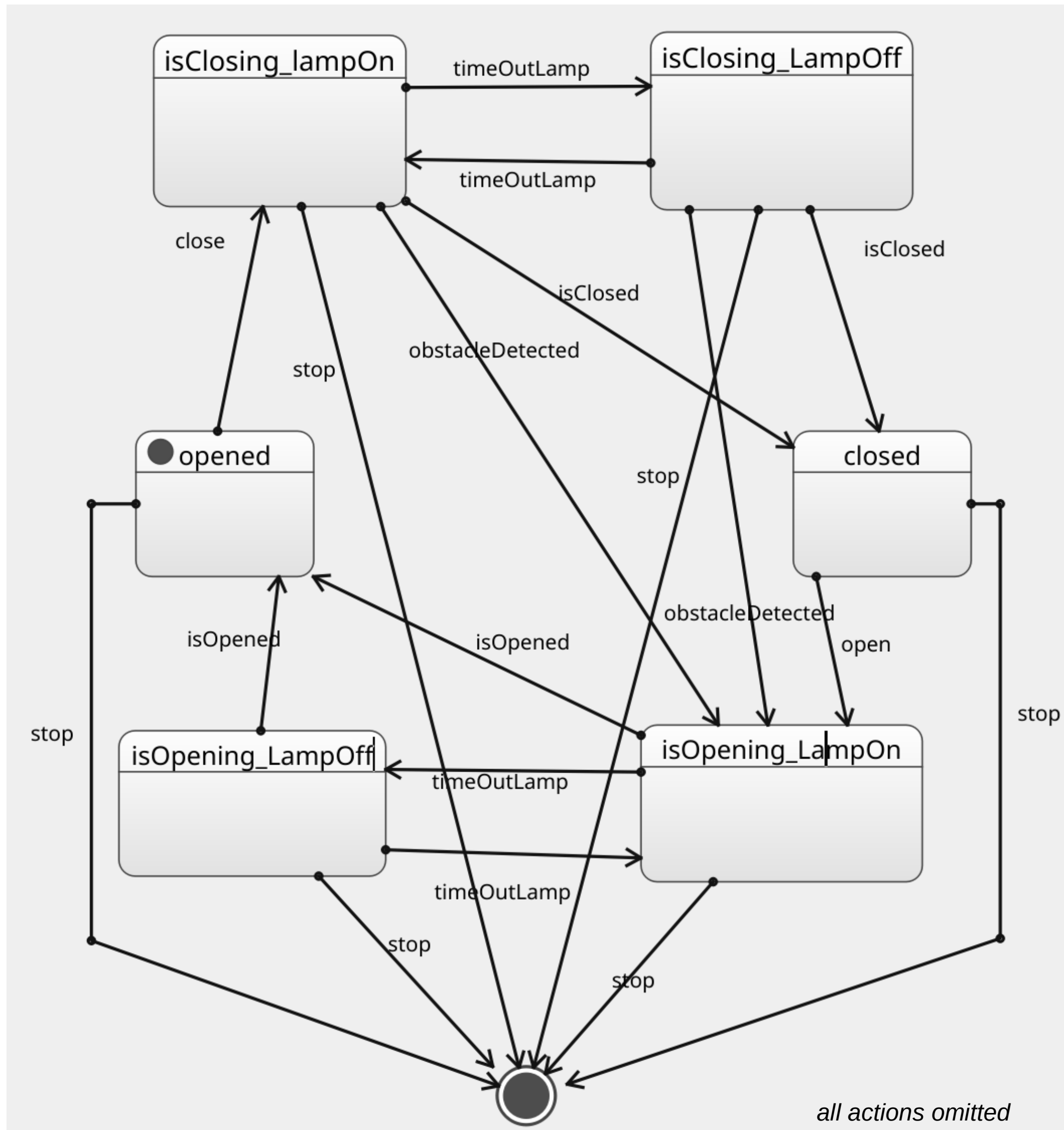
Finite State Machine

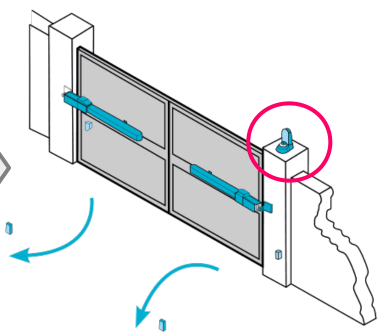
composition et mise à plat

composition et mise à plat: pourquoi ?

- D’un côté….

Running Example





wasn’t it supposed to help ?

State Charts

statecharts = state-diagrams + depth

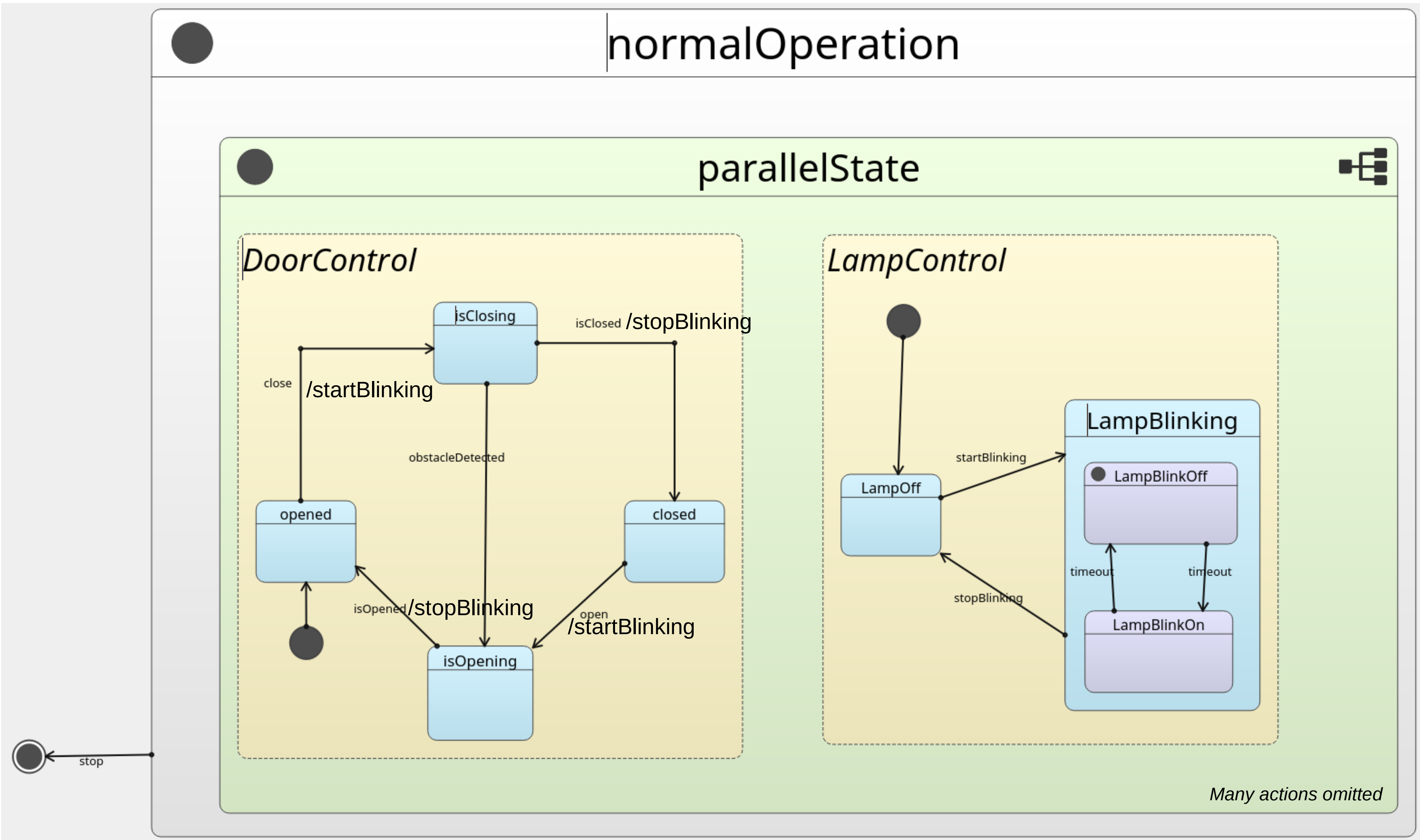
+ orthogonality + broadcast-communication.

David Harel
Statecharts: A visual formalism for complex systems
Science of computer programming 8 (3), 231-274
1987

State Charts

statecharts = state-diagrams + depth

+ orthogonality + broadcast-communication.

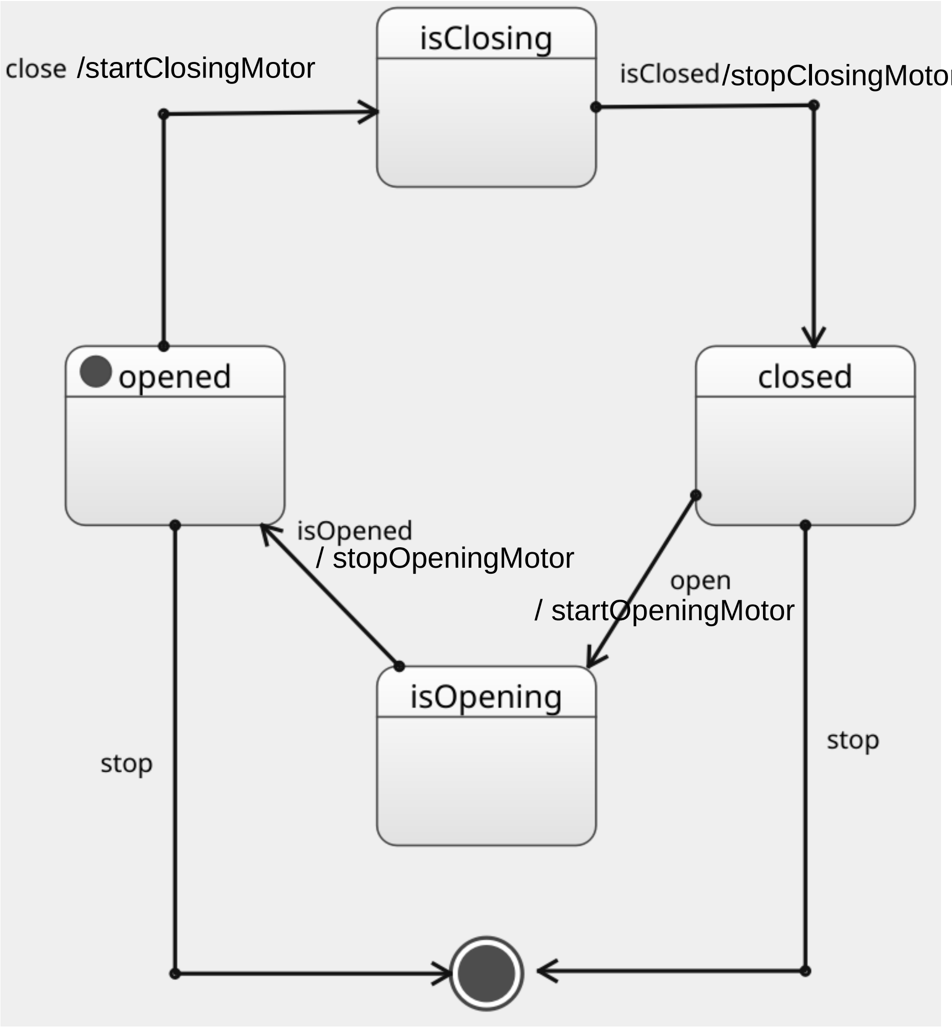


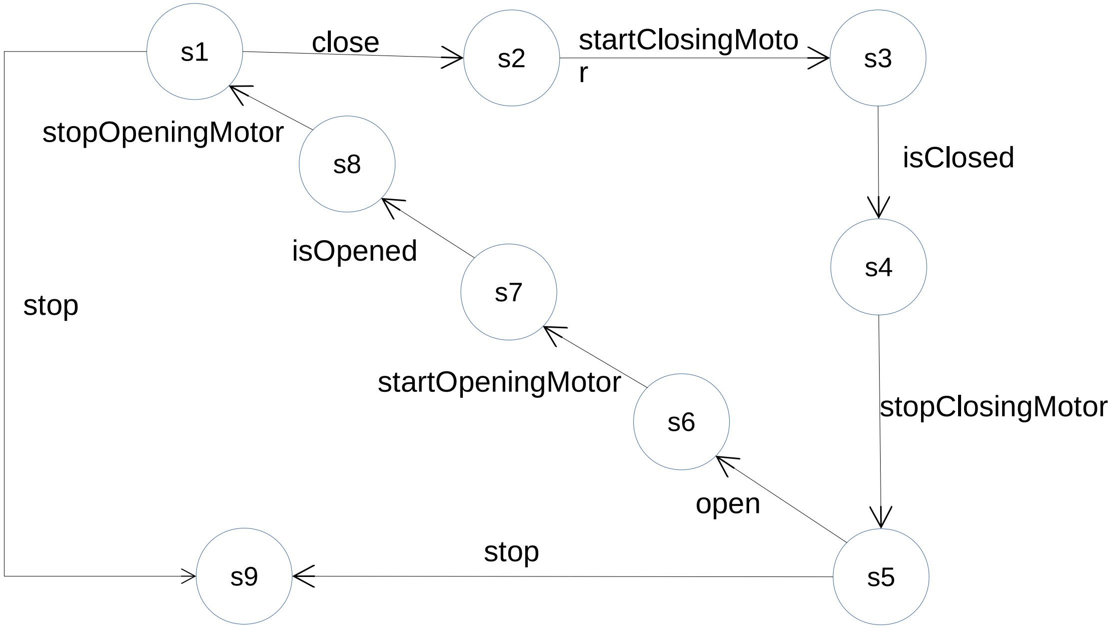
composition et mise à plat: pourquoi ?

- D’un autre côté….

V&V ?

- ensemble de chemins d'exécutions finis ?
  - Énumération de l'espace d'état (**habituellement un graphe orienté d'une forme particulière : *Labelled Transition system* ou *Kripke structure***)





Plus les actions *onEnter* et *onExit !*

composition et mise à plat: but

- La composition (aussi appelé produit) d'automate permet de n'obtenir qu'un seul transducteur à partir de plusieurs transducteur en parallèle. Il existe différentes manières (i.e., sémantiques) des les composer :
  - Produit synchrone (noté **X** dans la suite)
  - Produit asynchrone (produit parallèle, noté **||** dans la suite)
- La mise à plat (*flattening*) permet de "gommer" les états hiérarchiques tout en *gardant le même comportement* (notion que nous n'aurons pas le temps de voir, mais proche de l'équivalence de langages)

Produit Synchrone[1]

- Soit 2 transducers:
  - A1 = <$\mathcal{Q}1$, $q1_0$, $\mathcal{F}1$, $\Sigma 1_I$, $\Sigma 1_O$, $\delta 1$ >
  - A2 = <$\mathcal{Q}2$, $q2_0$, $\mathcal{F}2$, $\Sigma 2_I$, $\Sigma 2_O$, $\delta 2$ >

[1] différentes variantes mineures existent dans la littérature selon la formalisation de l’automate

Produit Synchrone[1]

- Soit 2 transducers:
  - A1 = <$\mathcal{Q}1$, $q1_0$, $\mathcal{F}1$, $\Sigma1_I$, $\Sigma1_O$, $\delta1$ >
  - A2 = <$\mathcal{Q}2$, $q2_0$, $\mathcal{F}2$, $\Sigma2_I$, $\Sigma2_O$, $\delta2$ >
- A_{res} = A1 x A2 = <$\mathcal{Q}$, q_0, $\mathcal{F}$, Σ_I, Σ_O, δ > tel que

[1] différentes variantes mineures existent dans la littérature selon la formalisation de l’automate

Produit Synchrone[1]

- Soit 2 transducers:
  - A1 = $< \mathcal{Q}1, q1_0, \mathcal{F}1, \Sigma 1_I, \Sigma 1_O, \delta 1 >$
  - A2 = $< \mathcal{Q}2, q2_0, \mathcal{F}2, \Sigma 2_I, \Sigma 2_O, \delta 2 >$
- A_{res} = A1 x A2 = $< \mathcal{Q}, q_0, \mathcal{F}, \Sigma_I, \Sigma_O, \delta >$ tel que
  - $\mathcal{Q} \subseteq \mathcal{Q}1 \otimes \mathcal{Q}2$ (produit cartésien)

[1] différentes variantes mineures existent dans la littérature selon la formalisation de l'automate

Produit Synchrone[1]

- Soit 2 transducers:
  - A1 = <$\mathcal{Q}1$, $q1_0$, $\mathcal{F}1$, $\Sigma 1_I$, $\Sigma 1_O$, $\delta 1$ >
  - A2 = <$\mathcal{Q}2$, $q2_0$, $\mathcal{F}2$, $\Sigma 2_I$, $\Sigma 2_O$, $\delta 2$ >
- A_{res} = A1 x A2 = <$\mathcal{Q}$, q_0, $\mathcal{F}$, Σ_I, Σ_O, δ > tel que
  - $\mathcal{Q} \subseteq \mathcal{Q}1 \otimes \mathcal{Q}2$ (produit cartésien)
  - $q_0 = <q1_0, q2_0>$

[1] différentes variantes mineures existent dans la littérature selon la formalisation de l'automate

Produit Synchrone[1]

- Soit 2 transducers:
  - A1 = $<\mathcal{Q}1\ , q1_0, \mathcal{F}1\ , \Sigma 1_I, \Sigma 1_O, \delta 1 >$
  - A2 = $<\mathcal{Q}2\ , q2_0, \mathcal{F}2\ , \Sigma 2_I, \Sigma 2_O, \delta 2 >$
- A_{res} = A1 x A2 = $<\mathcal{Q}\ , q_0, \mathcal{F}\ , \Sigma_I, \Sigma_O, \delta >$ tel que
  - $\mathcal{Q} \subseteq \mathcal{Q}1 \otimes \mathcal{Q}2$ (produit cartésien)
  - $q_0 = <q1_0 , q2_0>$
  - $\mathcal{F} \subseteq \mathcal{F}1 \otimes \mathcal{F}2$

[1] différentes variantes mineures existent dans la littérature selon la formalisation de l'automate

Produit Synchrone[1]

- Soit 2 transducers:
  - A1 = $<\mathcal{Q}1, q1_0, \mathcal{F}1, \Sigma1_I, \Sigma1_O, \delta1>$
  - A2 = $<\mathcal{Q}2, q2_0, \mathcal{F}2, \Sigma2_I, \Sigma2_O, \delta2>$
- A_{res} = A1 x A2 = $<\mathcal{Q}, q_0, \mathcal{F}, \Sigma_I, \Sigma_O, \delta>$ tel que
  - $\mathcal{Q} \subseteq \mathcal{Q}1 \otimes \mathcal{Q}2$ (produit cartésien)
  - $q_0 = <q1_0, q2_0>$
  - $\mathcal{F} \subseteq \mathcal{F}1 \otimes \mathcal{F}2$
  - $\Sigma_I \subseteq \Sigma1_I \cap \Sigma2_I$

[1] différentes variantes mineures existent dans la littérature selon la formalisation de l'automate

Produit Synchrone[1]

- Soit 2 transducers:
  - A1 = < $\mathcal{Q}1$, $q1_0$, $\mathcal{F}1$, $\Sigma 1_I$, $\Sigma 1_O$, $\delta 1$ >
  - A2 = < $\mathcal{Q}2$, $q2_0$, $\mathcal{F}2$, $\Sigma 2_I$, $\Sigma 2_O$, $\delta 2$ >
- A_{res} = A1 x A2 = < $\mathcal{Q}$, q_0, $\mathcal{F}$, Σ_I, Σ_O, δ > tel que
  - $\mathcal{Q} \subseteq \mathcal{Q}1 \otimes \mathcal{Q}2$ (produit cartésien)
  - $q_0 = <q1_0 ,q2_0>$
  - $\mathcal{F} \subseteq \mathcal{F}1 \otimes \mathcal{F}2$
  - $\Sigma_I \subseteq \Sigma 1_I \cap \Sigma 2_I$
  - $\Sigma_O \subseteq \Sigma 1_O \cup \Sigma 2_O$

[1] différentes variantes mineures existent dans la littérature selon la formalisation de l'automate

Produit Synchrone[1]

- Soit 2 transducers:
  - A1 = $< \mathcal{Q}1, q1_0, \mathcal{F}1, \Sigma 1_I, \Sigma 1_O, \delta 1 >$
  - A2 = $< \mathcal{Q}2, q2_0, \mathcal{F}2, \Sigma 2_I, \Sigma 2_O, \delta 2 >$
- A_{res} = A1 x A2 = $< \mathcal{Q}, q_0, \mathcal{F}, \Sigma_I, \Sigma_O, \delta >$ tel que
  - $\mathcal{Q} \subseteq \mathcal{Q}1 \otimes \mathcal{Q}2$ (produit cartésien)
  - $q_0 = < q1_0, q2_0 >$
  - $\mathcal{F} \subseteq \mathcal{F}1 \otimes \mathcal{F}2$
  - $\Sigma_I \subseteq \Sigma 1_I \cap \Sigma 2_I$
  - $\Sigma_O \subseteq \Sigma 1_O \cup \Sigma 2_O$
  - $\delta = \delta 1 \wedge \delta 2$ s.t. $\delta(<q1_1, q2_1>, i, o, <q1_2, q2_2>) :=$

[1] différentes variantes mineures existent dans la littérature selon la formalisation de l'automate

Produit Synchrone[1]

- Soit 2 transducers:
  - A1 = $<\mathcal{Q}1, q1_0, \mathcal{F}1, \Sigma 1_I, \Sigma 1_O, \delta 1>$
  - A2 = $<\mathcal{Q}2, q2_0, \mathcal{F}2, \Sigma 2_I, \Sigma 2_O, \delta 2>$
- A_{res} = A1 x A2 = $<\mathcal{Q}, q_0, \mathcal{F}, \Sigma_I, \Sigma_O, \delta>$ tel que
  - $\mathcal{Q} \subseteq \mathcal{Q}1 \otimes \mathcal{Q}2$ (produit cartésien)
  - $q_0 = <q1_0, q2_0>$
  - $\mathcal{F} \subseteq \mathcal{F}1 \otimes \mathcal{F}2$
  - $\Sigma_I \subseteq \Sigma 1_I \cap \Sigma 2_I$
  - $\Sigma_O \subseteq \Sigma 1_O \cup \Sigma 2_O$
  - $\delta = \delta 1 \wedge \delta 2$ s.t. $\delta(<q1_1, q2_1>, i, o, <q1_2, q2_2>) :=$

$$\begin{cases} \delta 1(q1_1, i, o1, q1_2) \wedge \delta 2(q2_1, i, o2, q2_2) & \text{if } \delta 1(q1_1, i, o1, q1_2) \text{ and } \delta 2(q2_1, i, o2, q2_2) \text{ defined, where } o = o1 \cup o2 \\ \text{undefined} & \text{otherwise} \end{cases}$$

Where $q1_1 \in \mathcal{Q}1,\ q1_2 \in \mathcal{Q}1,\ q2_1 \in \mathcal{Q}2,\ q2_2 \in \mathcal{Q}2,\ i1 \in \Sigma 1_I,\ i2 \in \Sigma 2_I,\ o1 \in \Sigma 1_O,\ o2 \in \Sigma 2_O$

[1] différentes variantes mineures existent dans la littérature selon la formalisation de l'automate

Produit Synchrone[1]

- Soit 2 transducers:
  - A1 = $<\mathcal{Q}1\ , q1_0, \mathcal{F}1\ , \Sigma 1_I, \Sigma 1_O, \delta 1>$
  - A2 = $<\mathcal{Q}2\ , q2_0, \mathcal{F}2\ , \Sigma 2_I, \Sigma 2_O, \delta 2>$
- A_{res} = A1 x A2 = $<\mathcal{Q}\ , q_0, \mathcal{F}\ , \Sigma_I, \Sigma_O, \delta>$ tel que
  - $\mathcal{Q} \subseteq \mathcal{Q}1 \otimes \mathcal{Q}2$ (produit cartésien)
  - $q_0 = <q1_0, q2_0>$
  - $\mathcal{F} \subseteq \mathcal{F}1 \otimes \mathcal{F}2$
  - $\Sigma_I \subseteq \Sigma 1_I \cap \Sigma 2_I$
  - $\Sigma_O \subseteq \Sigma 1_O \cup \Sigma 2_O$
  - $\delta = \delta 1 \wedge \delta 2$ s.t. $\delta(<q1_1, q2_1>, i, o, <q1_2, q2_2>) :=$

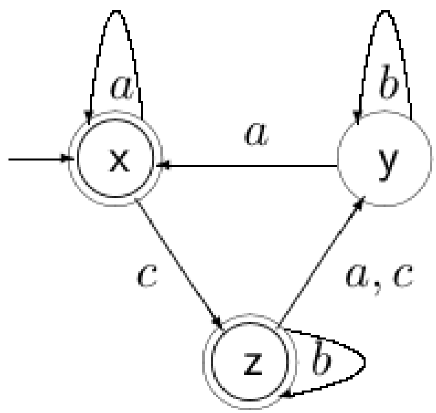
$$\begin{cases} \delta 1(q1_1, i, o1, q1_2) \wedge \delta 2(q2_1, i, o2, q2_2) & \text{if } \delta 1(q1_1, i, o1, q1_2) \text{ and } \delta 2(q2_1, i, o2, q2_2) \text{ defined, where } o = o1 \cup o2 \\ \text{undefined} & \text{otherwise} \end{cases}$$

Si la FSM est plus compliquée,(e.g., avec des conditions booléennes), il faut bien sûr le prendre en compte (en faisant la conjonction des conditions)

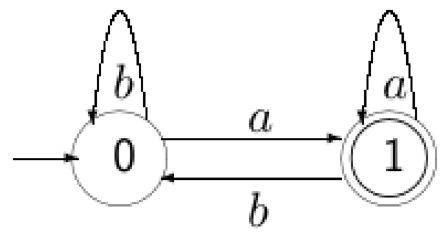
Where $q1_1 \in \mathcal{Q}1,\ q1_2 \in \mathcal{Q}1,\ q2_1 \in \mathcal{Q}2,\ q2_2 \in \mathcal{Q}2,\ i1 \in \Sigma 1_I,\ i2 \in \Sigma 2_I,\ o1 \in \Sigma 1_O,\ o2 \in \Sigma 2_O$

[1] différentes variantes mineures existent dans la littérature selon la formalisation de l'automate

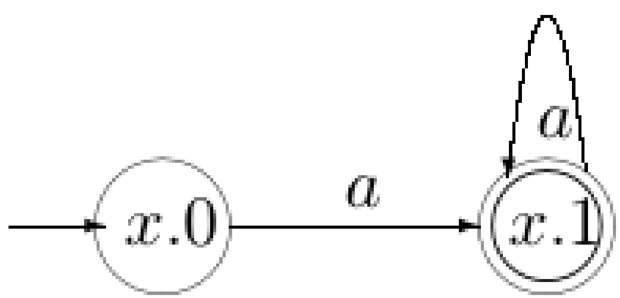
Produit Synchrone: simplified example



A



B



Produit Asynchrone

- Soit 2 transducers:
  - A1 = $<\mathcal{Q}1\ ,\ q1_0,\ \mathcal{F}1\ ,\ \Sigma 1_I,\ \Sigma 1_O,\ \delta 1 >$
  - A2 = $<\mathcal{Q}2\ ,\ q2_0,\ \mathcal{F}2\ ,\ \Sigma 2_I,\ \Sigma 2_O,\ \delta 2 >$
- A_{res} = A1 x A2 = $<\mathcal{Q}\ ,\ q_0,\ \mathcal{F}\ ,\ \Sigma_I,\ \Sigma_O,\ \delta >$ tel que
  - $\mathcal{Q} \subseteq \mathcal{Q}1 \otimes \mathcal{Q}2$ (produit cartésien)
  - $q_0 = <q1_0, q2_0>$
  - $\mathcal{F} \subseteq \mathcal{F}1 \otimes \mathcal{F}2$
  - $\Sigma_I \subseteq \Sigma 1_I \cup \Sigma 2_I$
  - $\Sigma_O \subseteq \Sigma 1_O \cup \Sigma 2_O$
  - $\delta = \delta 1 \vee \delta 2$ s.t. $\delta(<q1_1, q2_1>, i, *, <*, *>) :=$

$$\begin{cases} \delta(<q1_1, q2_1>, i, o, <q1_2, q2_2>) & \text{if } \delta 1(q1_1, i, o1, q1_2) \text{ and } \delta 2(q2_1, i, o2, q2_2) \text{ defined, where } o = o1 \cup o2 \\ \text{undefined} & \text{otherwise} \end{cases}$$

Where $q1_1 \in \mathcal{Q}1,\ q1_2 \in \mathcal{Q}1,\ q2_1 \in \mathcal{Q}2,\ q2_2 \in \mathcal{Q}2,\ i1 \in \Sigma 1_I,\ i2 \in \Sigma 2_I,\ o1 \in \Sigma 1_O,\ o2 \in \Sigma 2_O$

Produit Asynchrone

- Soit 2 transducers:
  - A1 = $<Q1\ , q1_0, \mathcal{F}1\ , \Sigma1_I, \Sigma1_O, \delta1>$
  - A2 = $<Q2\ , q2_0, \mathcal{F}2\ , \Sigma2_I, \Sigma2_O, \delta2>$
- A_{res} = A1 x A2 = $<Q\ , q_0, \mathcal{F}\ , \Sigma_I, \Sigma_O, \delta>$ tel que
  - $Q \subseteq Q1 \otimes Q2$ (produit cartésien)
  - $q_0 = <q1_0, q2_0>$
  - $\mathcal{F} \subseteq \mathcal{F}1 \otimes \mathcal{F}2$
  - $\Sigma_I \subseteq \Sigma1_I \cup \Sigma2_I$
  - $\Sigma_O \subseteq \Sigma1_O \cup \Sigma2_O$
  - $\delta = \delta1 \vee \delta2$ s.t. $\delta(<q1_1, q2_1>, i, {}^*, <{}^*, {}^*>) :=$

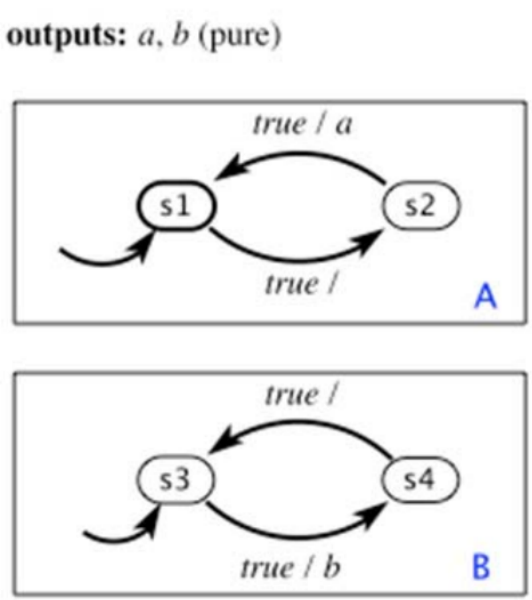
$$
\begin{cases}
\delta(<q1_1, q2_1>, i, o, <q1_2, q2_2>) & \text{if } \delta1(q1_1, i, o1, q1_2) \text{ and } \delta2(q2_1, i, o2, q2_2) \text{ defined, where } o = o1 \cup o2 \\
\delta(<q1_1, q2_1>, i, o2, <q1_1, q2_2>) & \text{if } i \notin \Sigma1_I \text{ and } \delta2(q2_1, i, o2, q2_2) \text{ defined} \\
\delta(<q1_1, q2_1>, i, o1, <q1_2, q2_1>) & \text{if } \delta1(q1_1, i, o1, q1_2) \text{ defined and } i \notin \Sigma2_I \\
\text{undefined} & \text{otherwise}
\end{cases}
$$

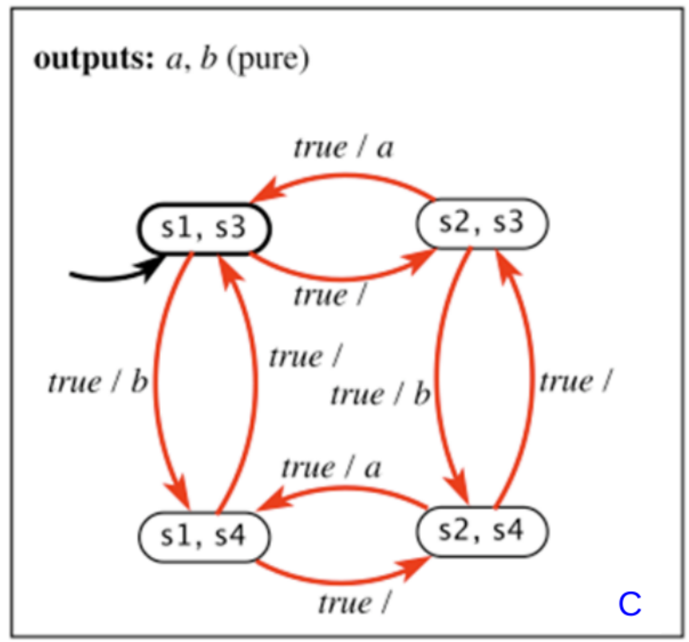
Where $q1_1 \in Q1,\ q1_2 \in Q1,\ q2_1 \in Q2,\ q2_2 \in Q2,\ i1 \in \Sigma1_I,\ i2 \in \Sigma2_I,\ o1 \in \Sigma1_O,\ o2 \in \Sigma2_O$

Produit Asynchrone: simplified example

Asynchronous Composition

$$S_C \subseteq S_A \times S_B$$





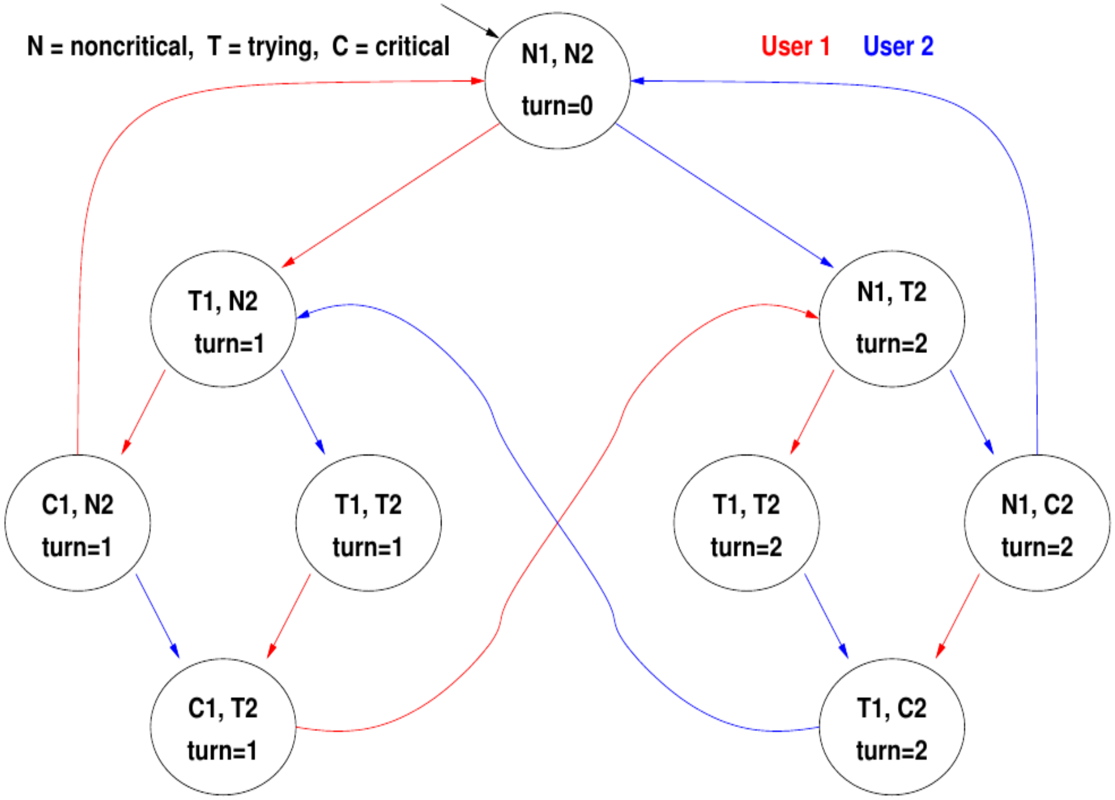
Asynchronous composition using interleaving semantics

Note that now all states are reachable.

Utilisation de la composition

- En Model Checking

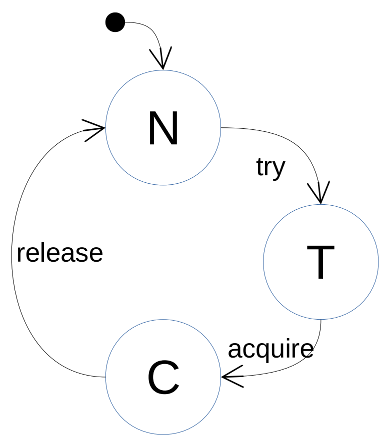
  Permet de passer d'automates parallèles, à une structure plus proche d'une structure adaptée (LTS, kripke)

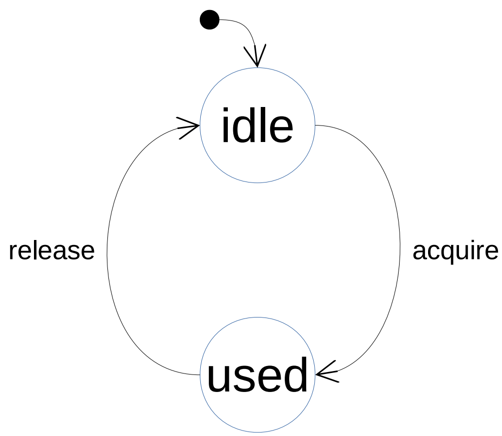


$$\models \Box (T_1 \Rightarrow \Diamond C_1) \ ?$$

Utilisation de la composition

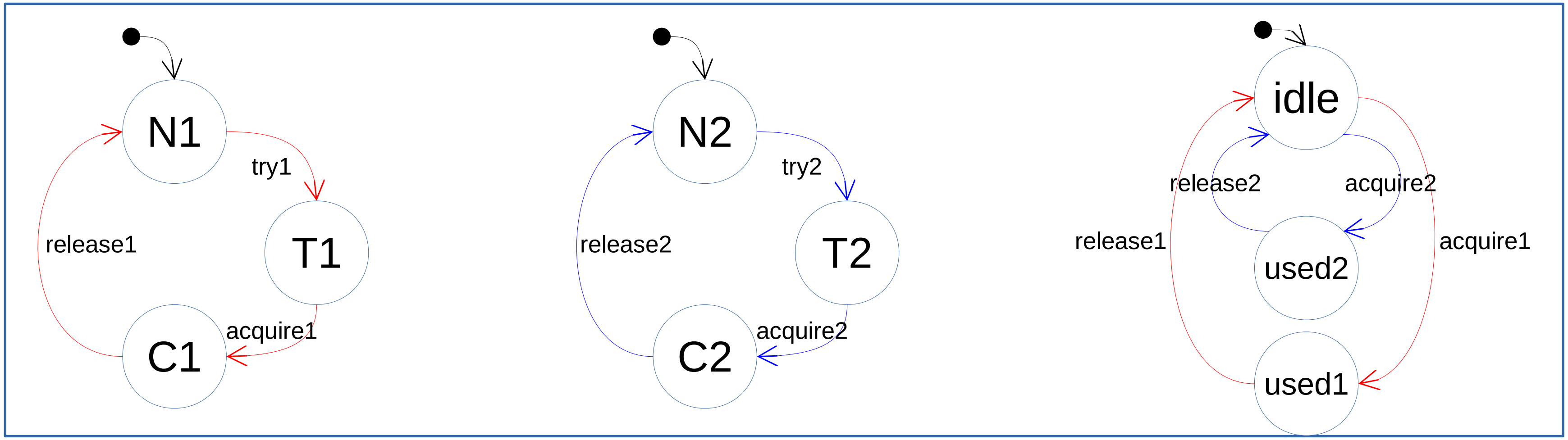
- En Model Checking





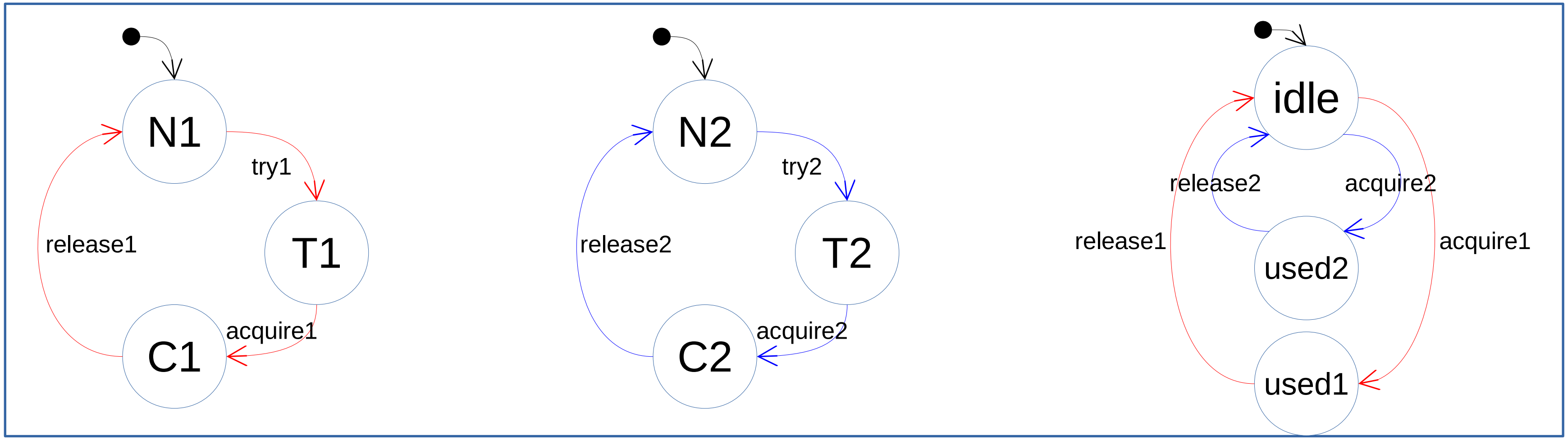
Utilisation de la composition

- En Model Checking



Utilisation de la composition

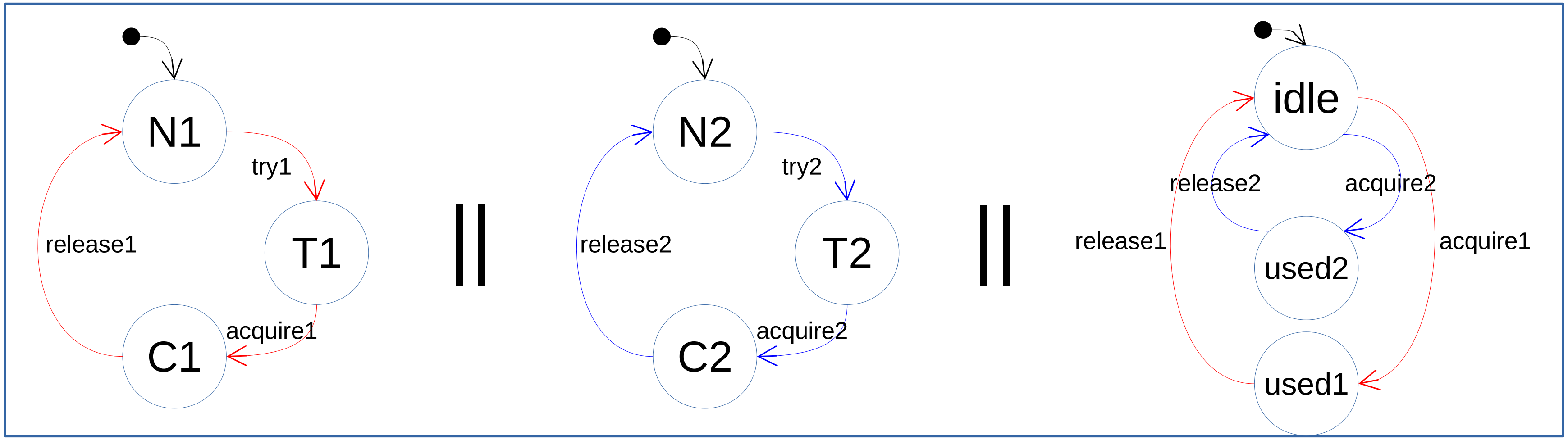
- En Model Checking



Simplification uniquement pour faciliter la lecture par la suite !

Utilisation de la composition

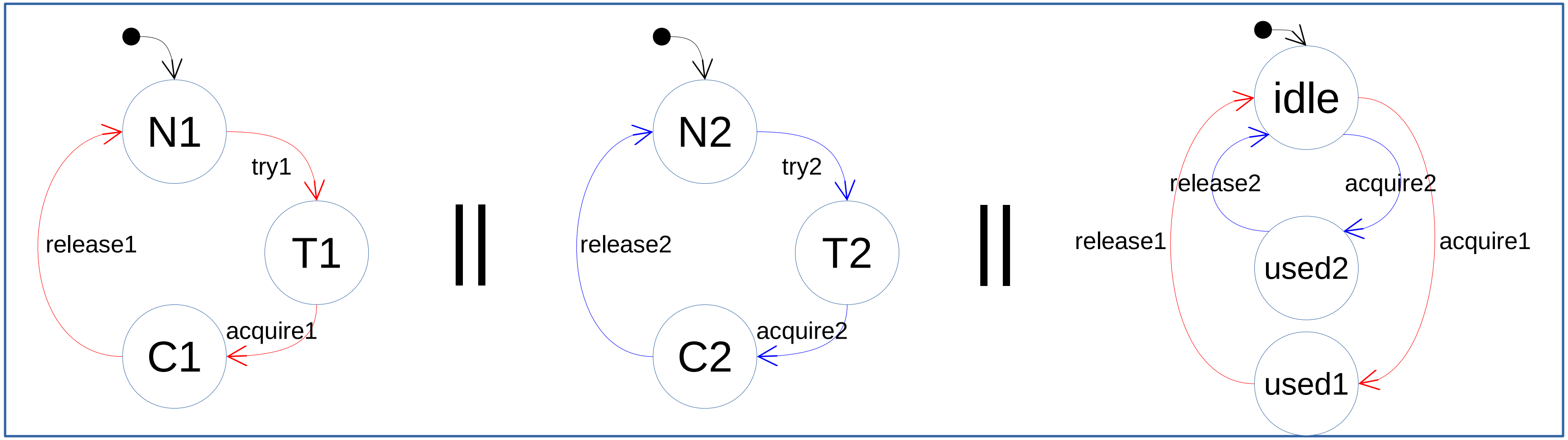
- En Model Checking



→ Le produit asynchrone nous donne l'automate au comportement équivalent...

Utilisation de la composition

- En Model Checking

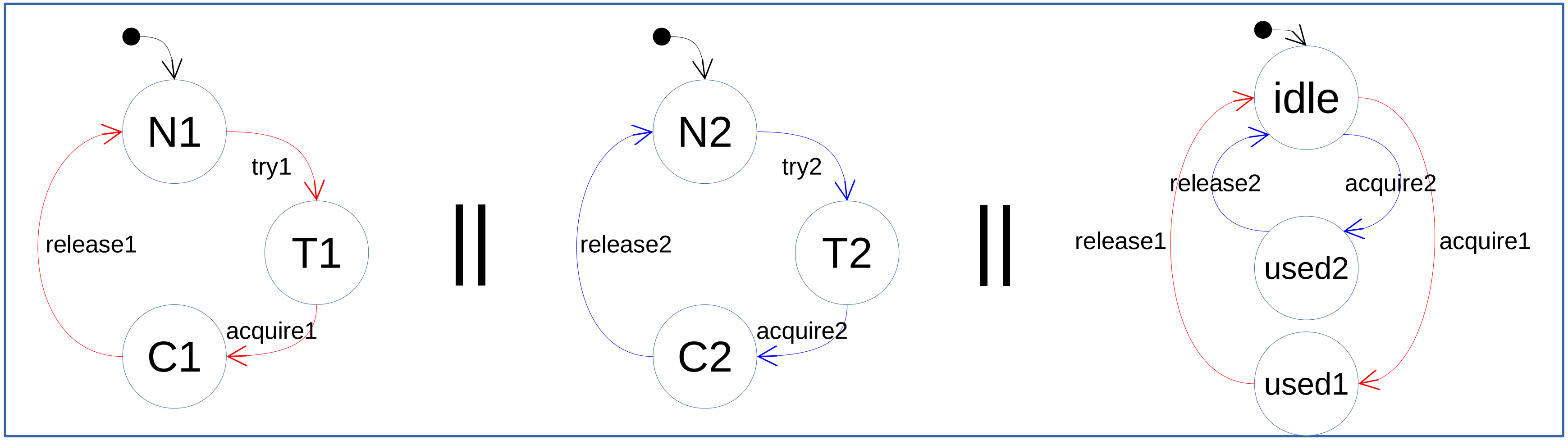


$$\models \Box(\textbf{try1} \Rightarrow \Diamond\,\textbf{acquire1})\ ?$$

→ Le produit asynchrone nous donne l’automate au comportement équivalent...... Que l’on va pouvoir questionner.

Utilisation de la composition

- En Model Checking



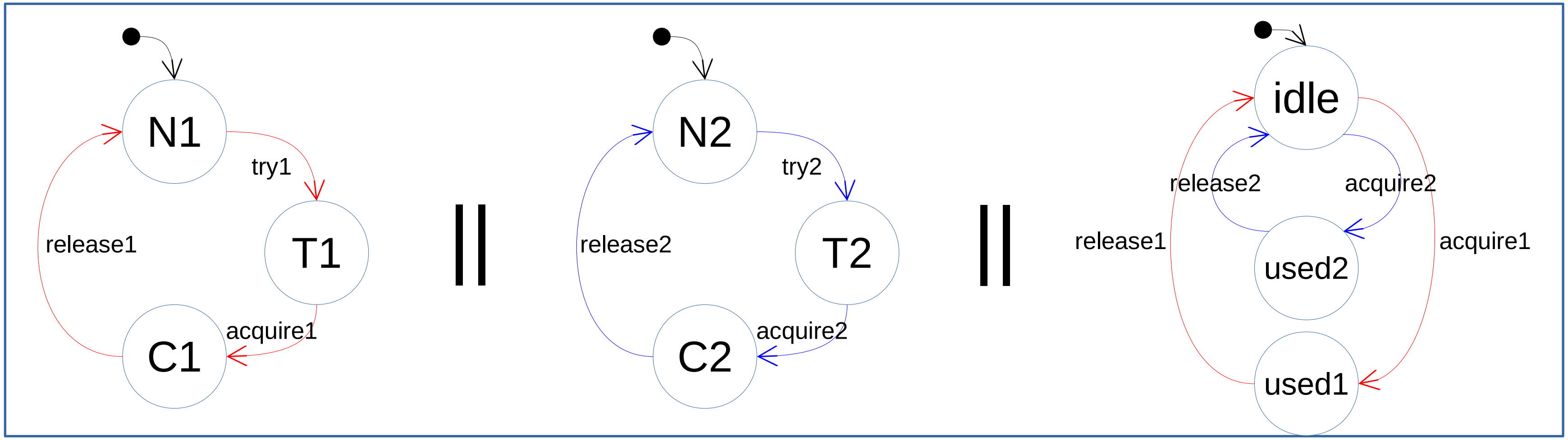
$$\models \Box(\textbf{try1} \Rightarrow \Diamond \textbf{acquire1}) \ ?$$

Safety or liveness ?

→ Le produit asynchrone nous donne l'automate au comportement équivalent…… Que l'on va pouvoir questionner.

Utilisation de la composition

- En Model Checking



$$\models \Box(\mathbf{try1} \Rightarrow \Diamond\, \mathbf{acquire1}) \ ?$$

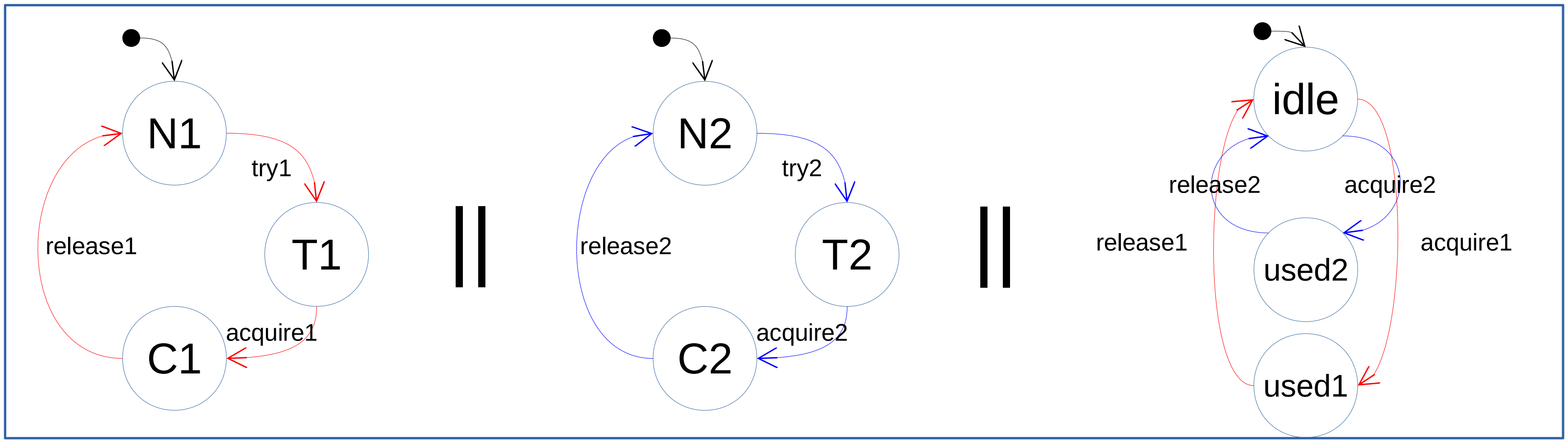
Safety or **liveness**

Something good eventually happens….

→ Le produit asynchrone nous donne l’automate au comportement équivalent…… Que l’on va pouvoir questionner.

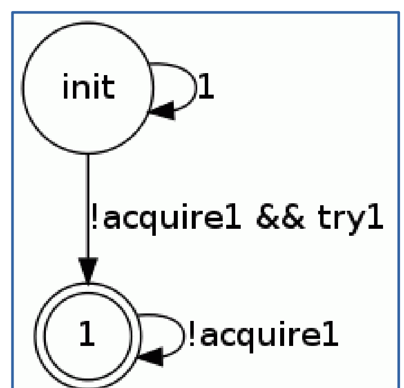
Utilisation de la composition

- En Model Checking



X

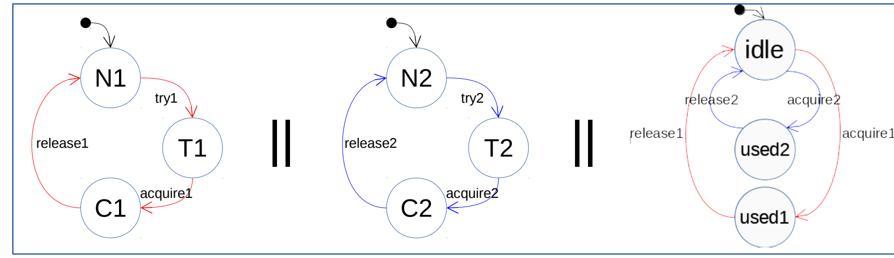
$$\neg \square \left(\mathbf{try1} \Rightarrow \lozenge \, \mathbf{acquire1} \right) \equiv$$



→ Le produit synchrone d'automate ne gardera que les chemins problématiques. Si l'automate résultant possède un cycle passant par un état acceptant, la propriété n'est pas vérifiée.

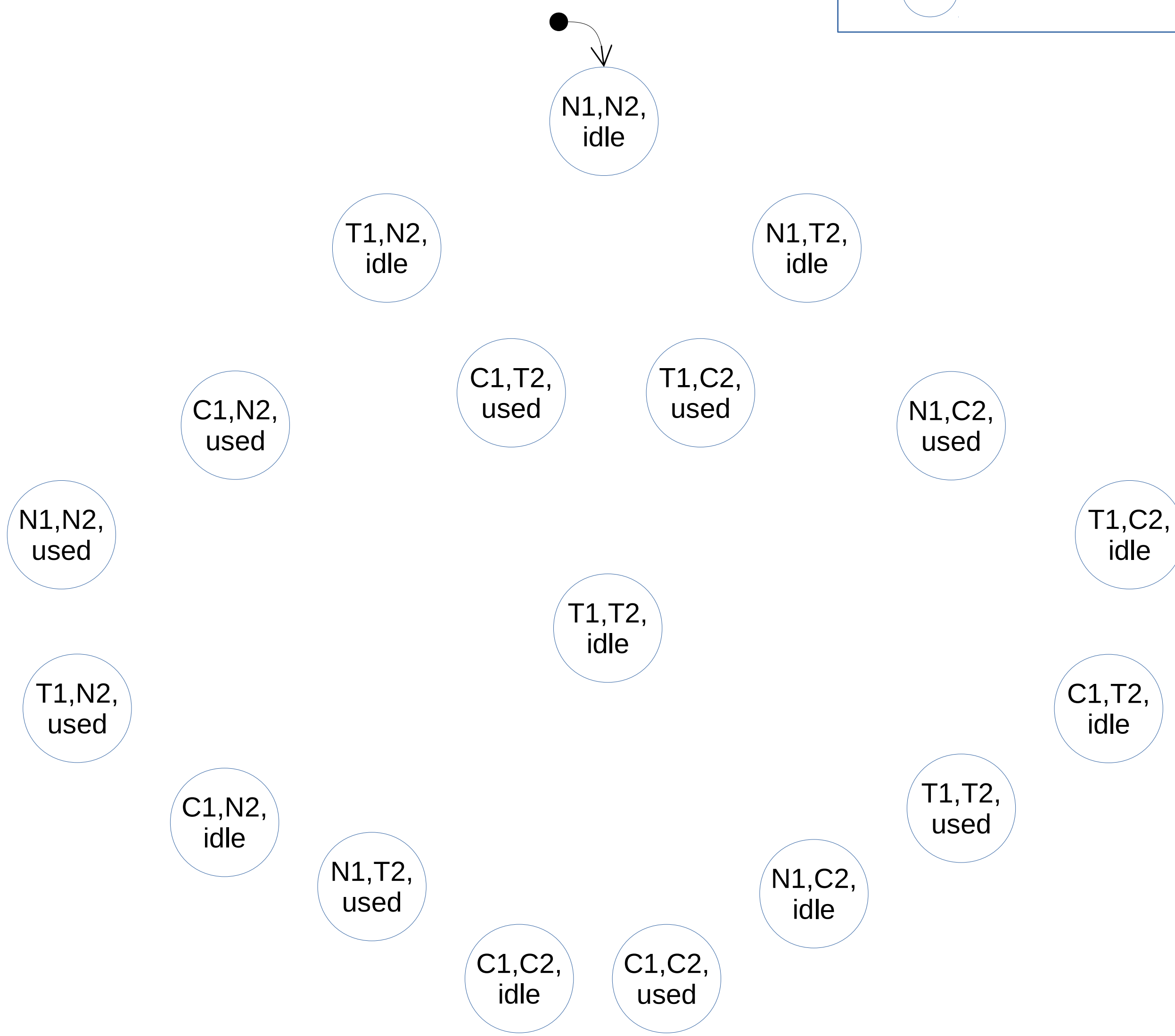
Utilisation de la composition

- En Model Checking



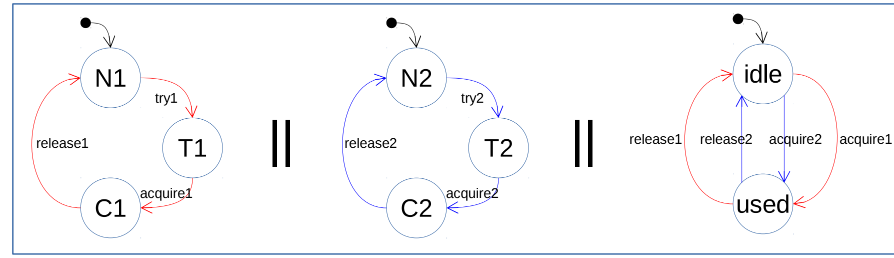
$Q = Q1 \otimes Q2$

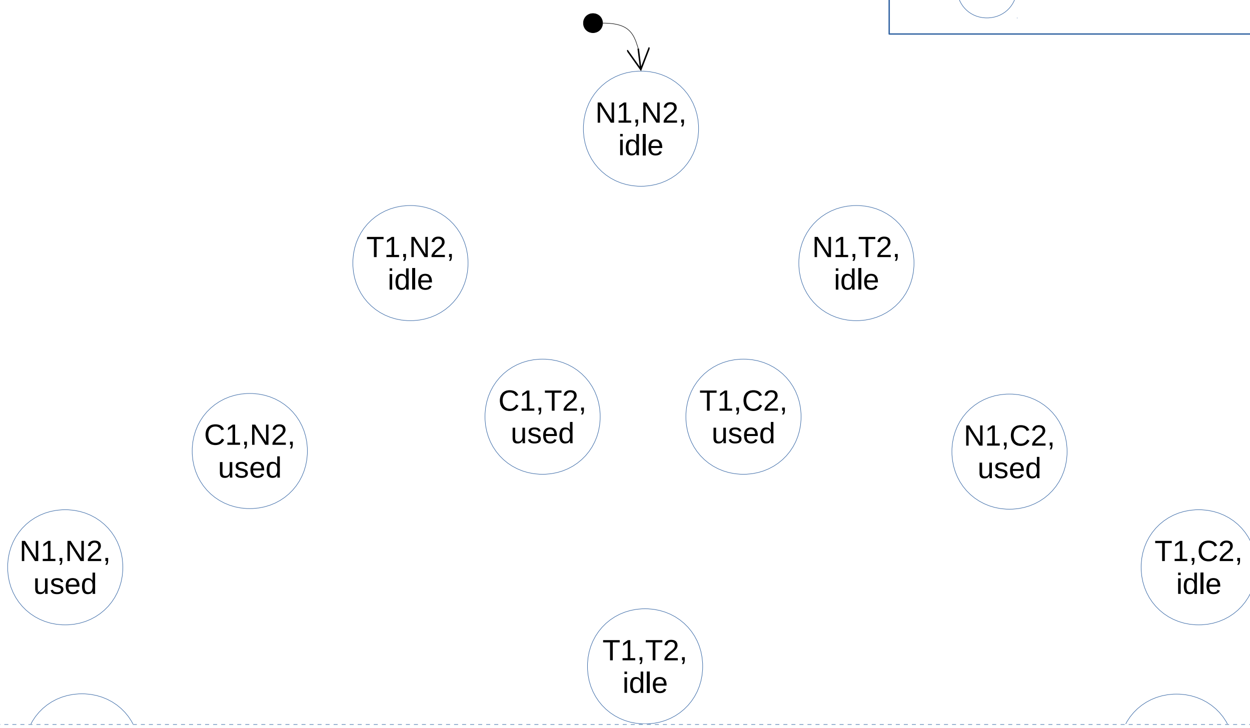
$q_0 = <q1_0, q2_0>$



Utilisation de la composition

- En Model Checking



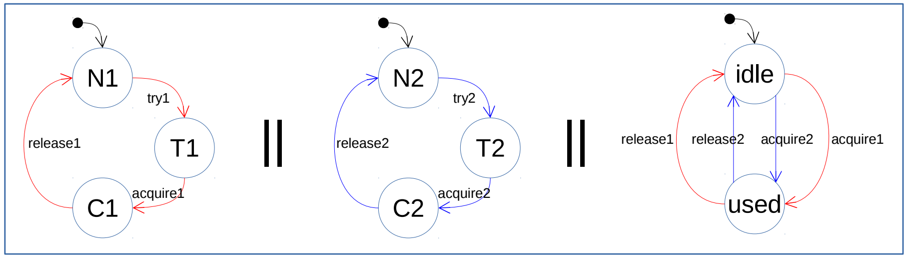


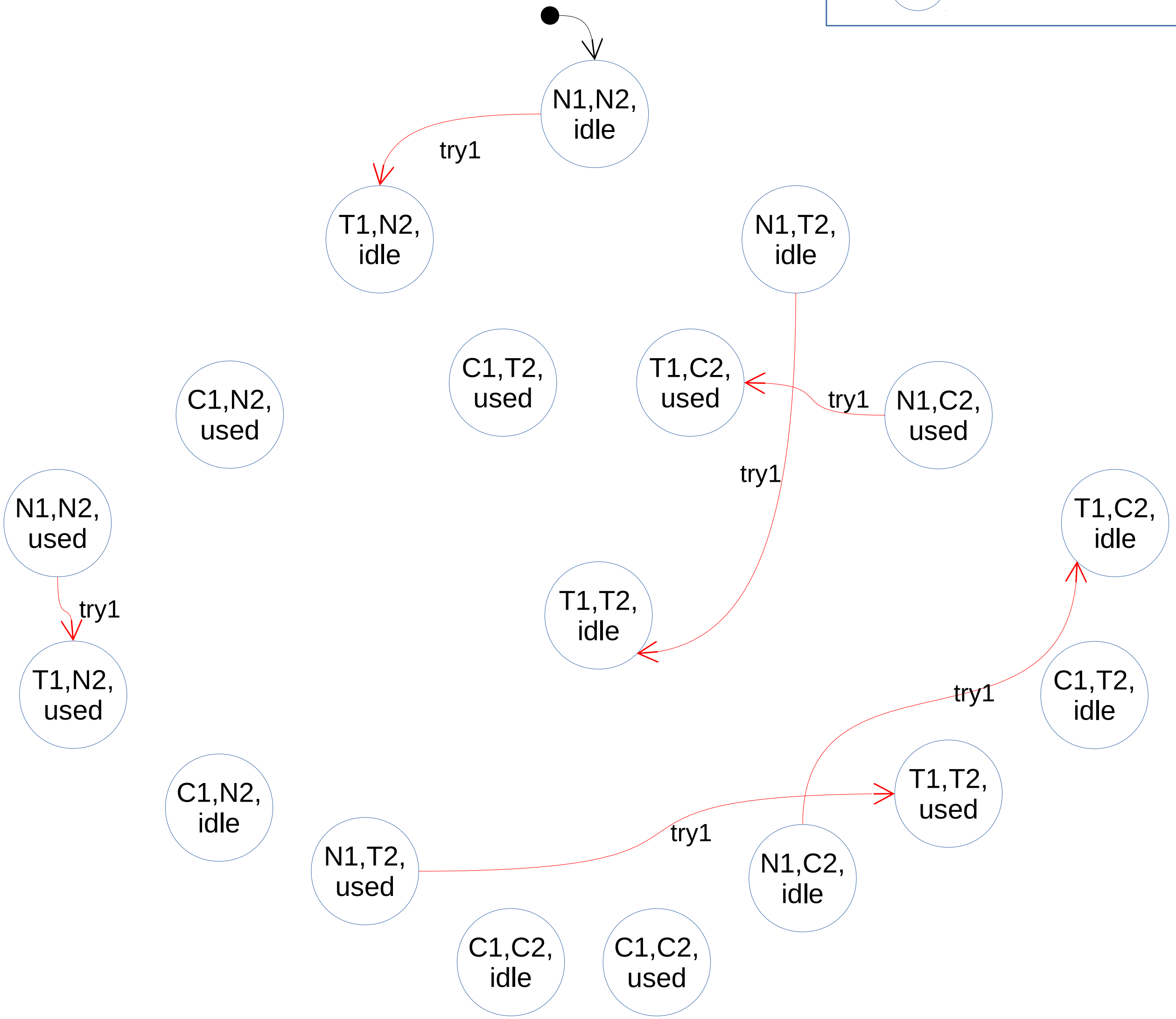
$\delta = \delta 1 \cup \delta 2$ s.t. $\delta(<q1_1, q2_1>, i, *, <*,*>) :=$

$$
\begin{cases}
\delta(<q1_1, q2_1>, i, o, <q1_2, q2_2>) & \text{if } \delta 1(q1_1, i, o1, q1_2) \text{ and } \delta 2(q2_1, i, o2, q2_2) \text{ defined, where } o = o1 \cup o2 \\
\delta(<q1_1, q2_1>, i, o2, <q1_1, q2_2>) & \text{if } i \notin \Sigma 1_I \text{ and } \delta 2(q2_1, i, o2, q2_2) \text{ defined} \\
\delta(<q1_1, q2_1>, i, o1, <q1_2, q2_1>) & \text{if } \delta 1(q1_1, i, o1, q1_2) \text{ defined and } i \notin \Sigma 2_I \\
\text{undefined} & \text{otherwise}
\end{cases}
$$

Utilisation de la composition

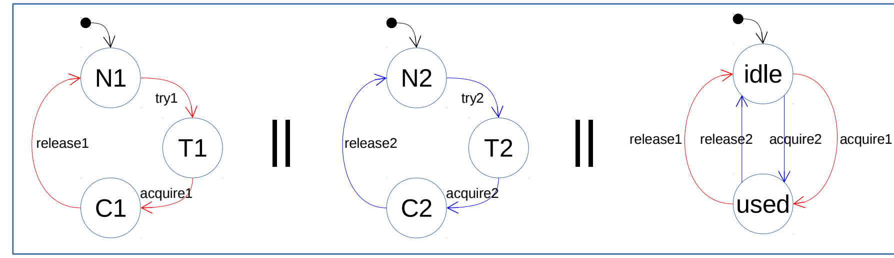
- En Model Checking

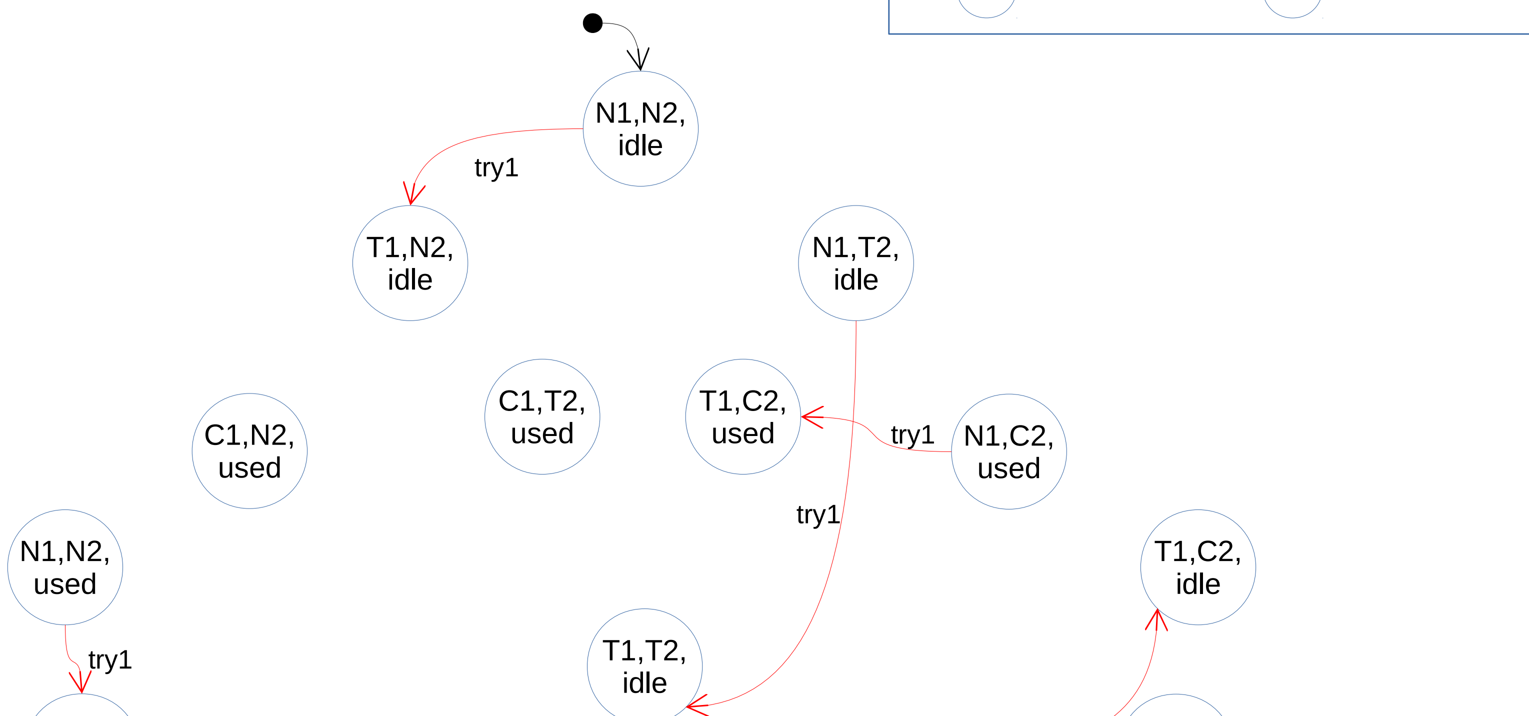




Utilisation de la composition

- En Model Checking



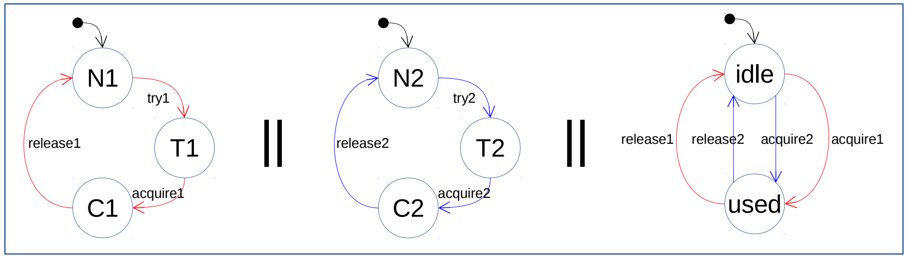


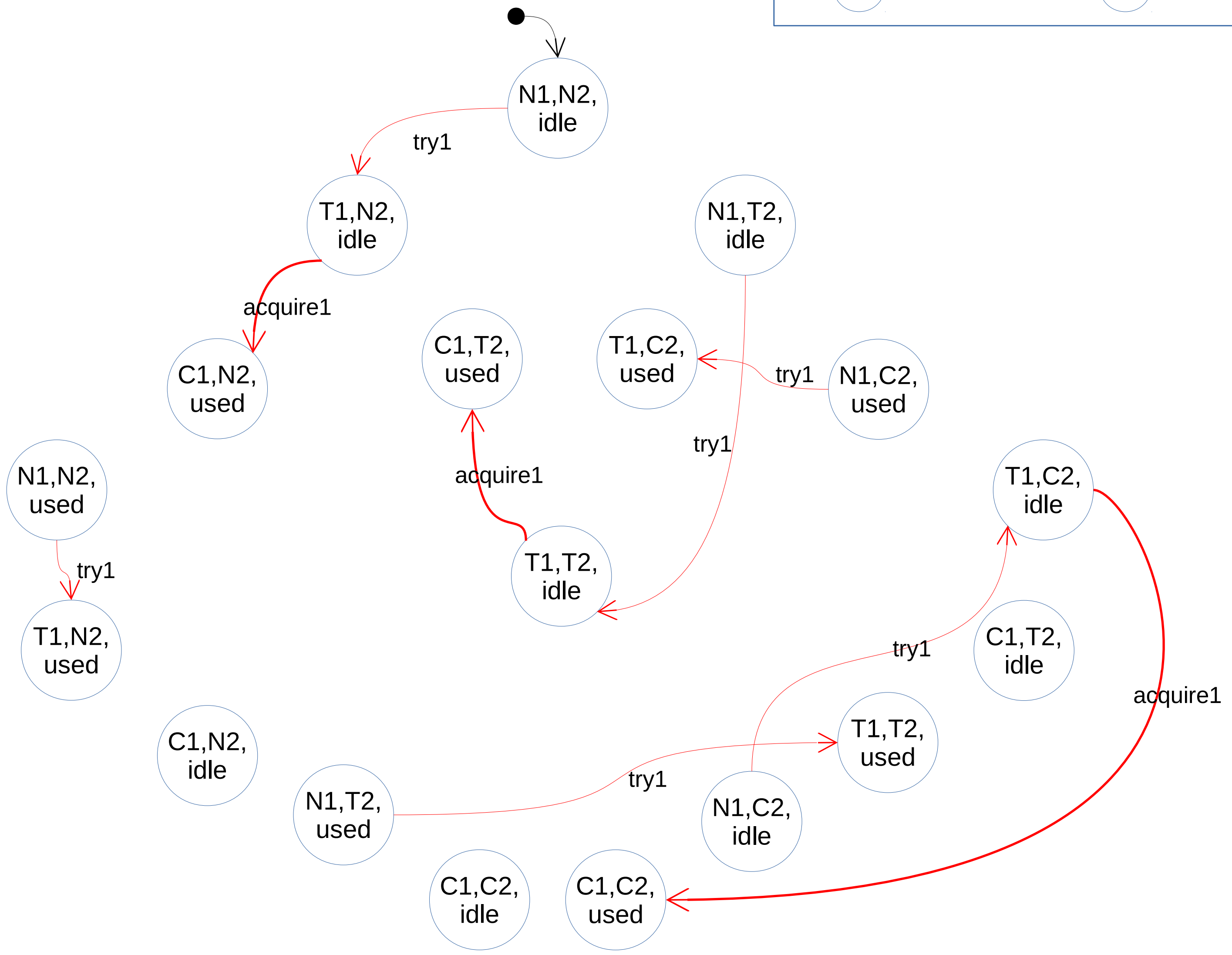
$$\delta = \delta 1 \cup \delta 2 \text{ s.t. } \delta(<q1_1, q2_1>, i, *, <*,*>) :=$$

$$\begin{cases} \delta(<q1_1, q2_1>, i, o, <q1_2, q2_2>) \text{ if } \delta 1(q1_1, i, o1, q1_2) \text{ and } \delta 2(q2_1, i, o2, q2_2) \text{ defined, where } o = o1 \cup o2 \\ \delta(<q1_1, q2_1>, i, o2, <q1_1, q2_2>) \text{ if } i \notin \Sigma 1_I \text{ and } \delta 2(q2_1, i, o2, q2_2) \text{ defined} \\ \delta(<q1_1, q2_1>, i, o1, <q1_2, q2_1>) \text{ if } \delta 1(q1_1, i, o1, q1_2) \text{ defined and } i \notin \Sigma 2_I \\ \text{undefined} \qquad \text{otherwise} \end{cases}$$

Utilisation de la composition

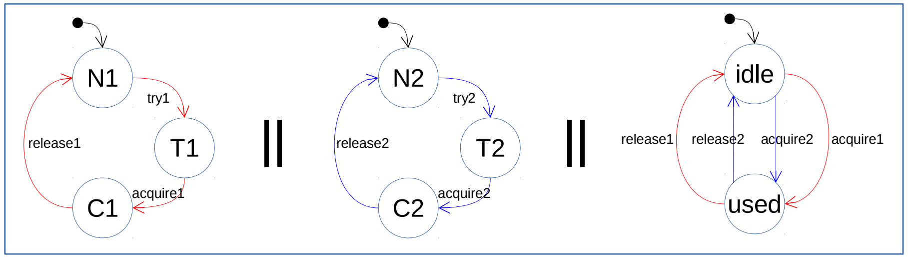
- En Model Checking

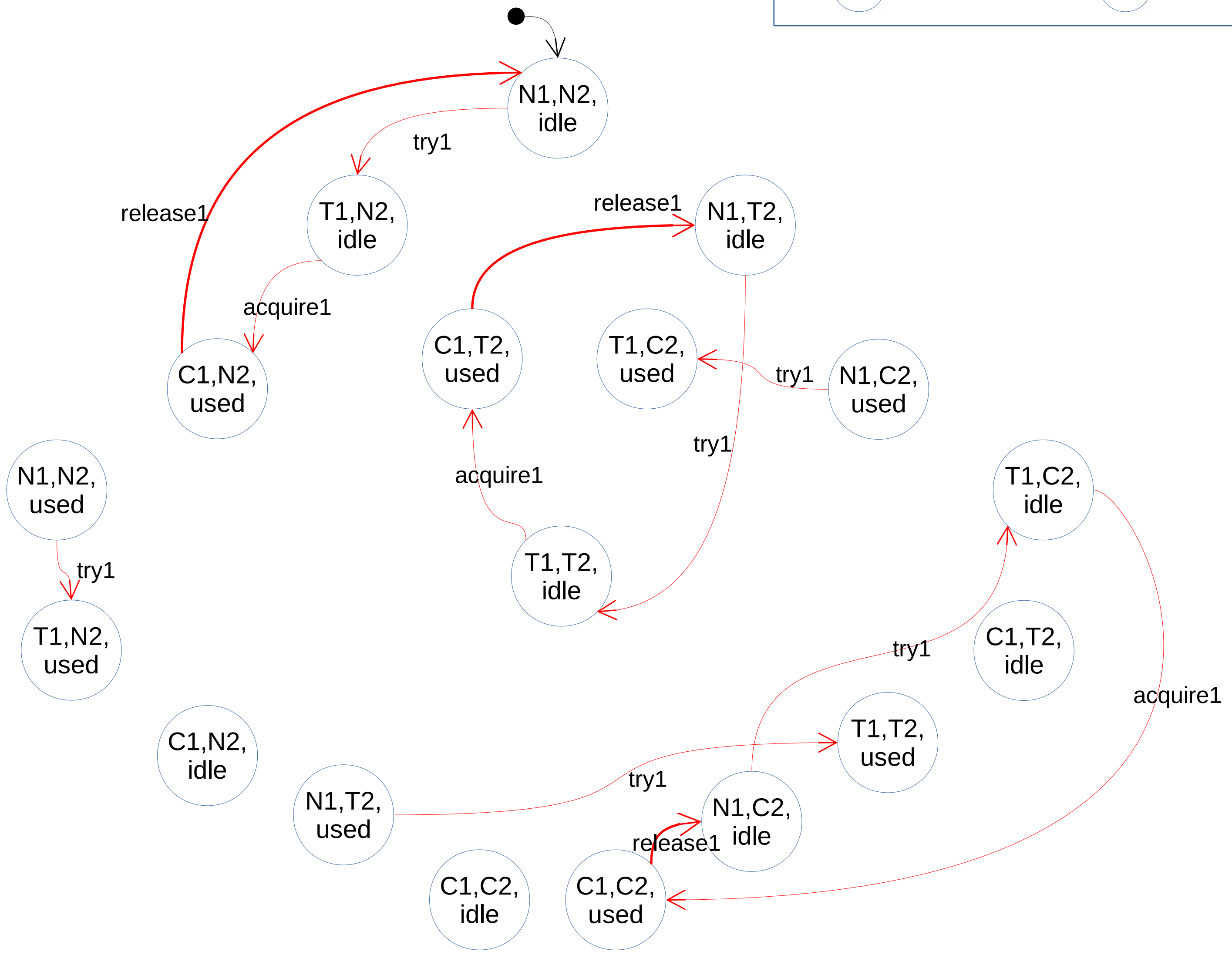




Utilisation de la composition

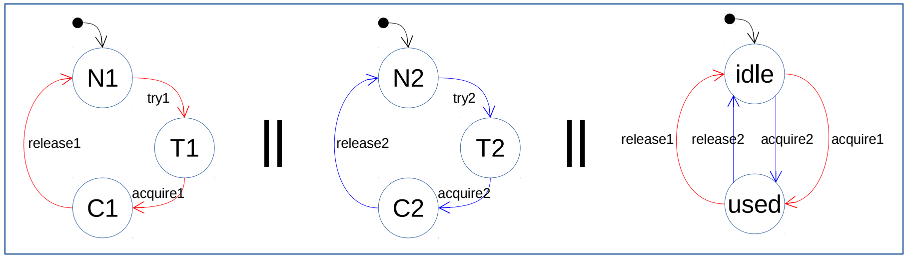
- En Model Checking

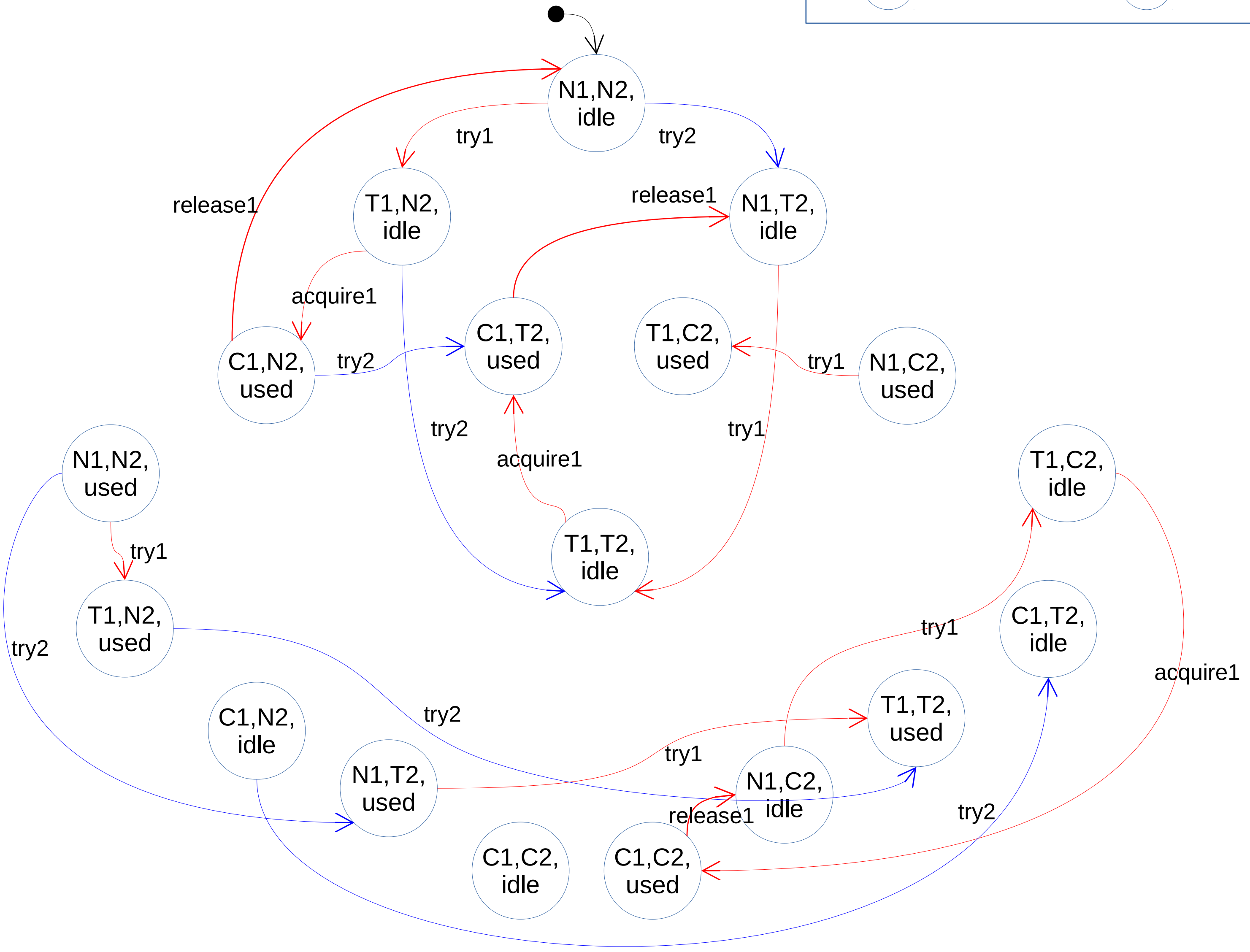




Utilisation de la composition

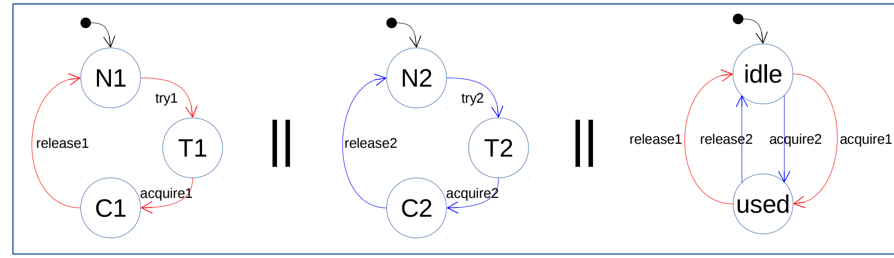
- En Model Checking

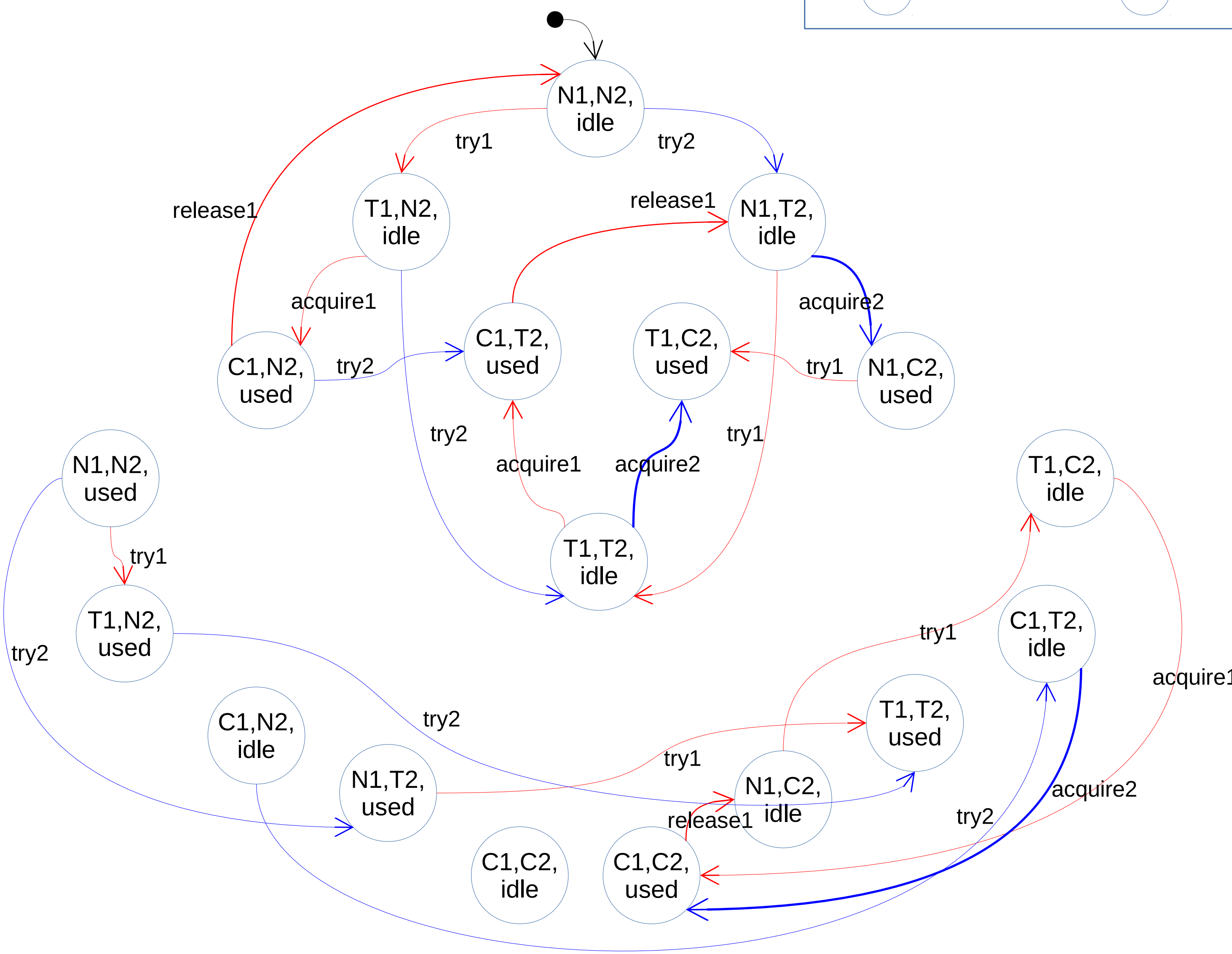




Utilisation de la composition

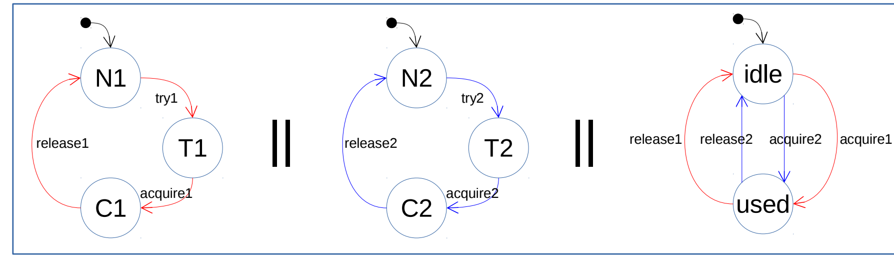
- En Model Checking

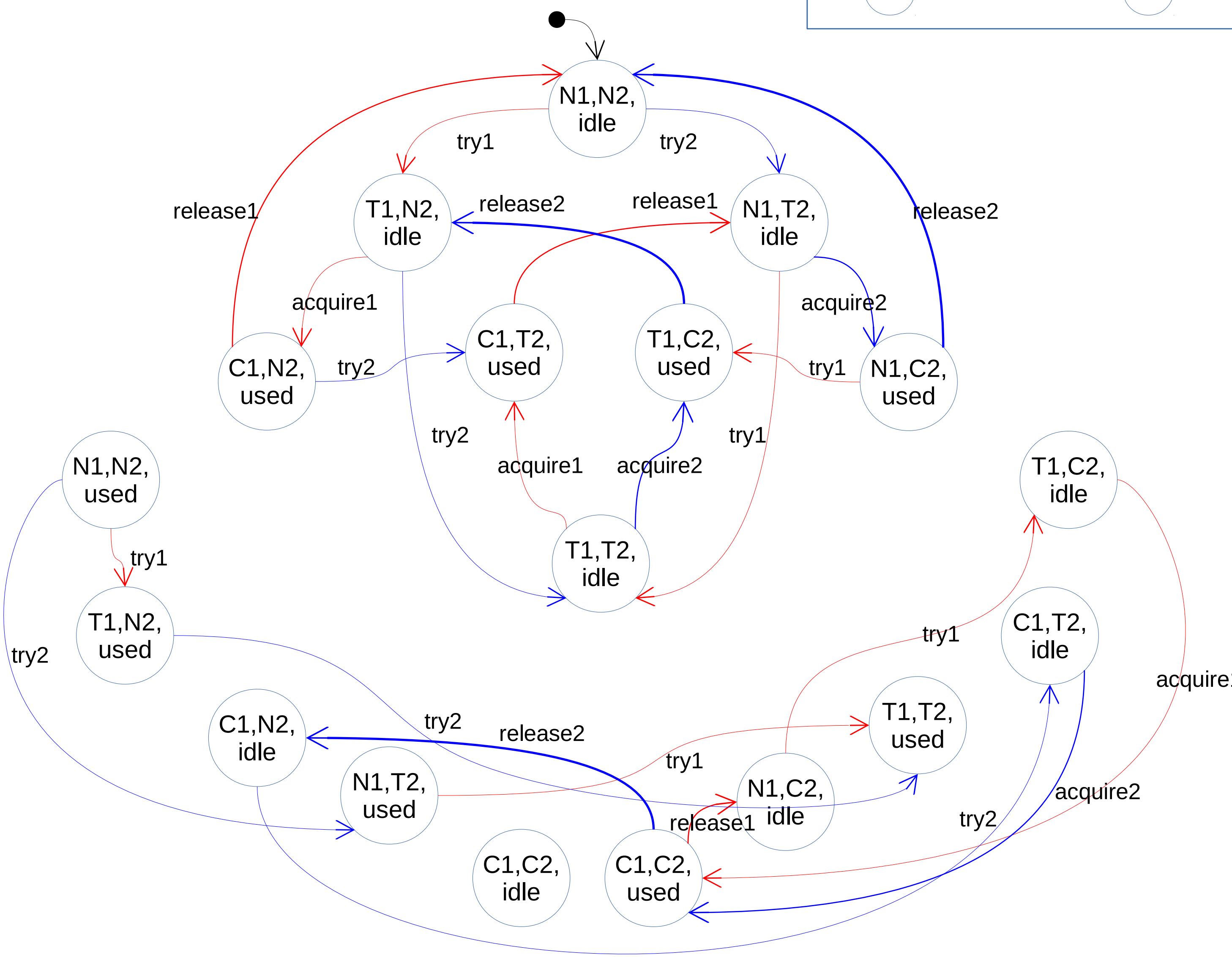




Utilisation de la composition

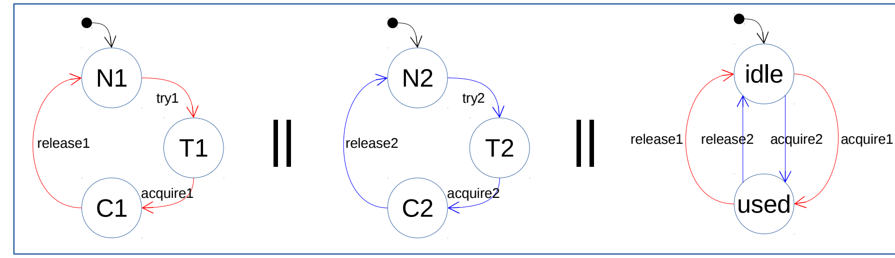
- En Model Checking

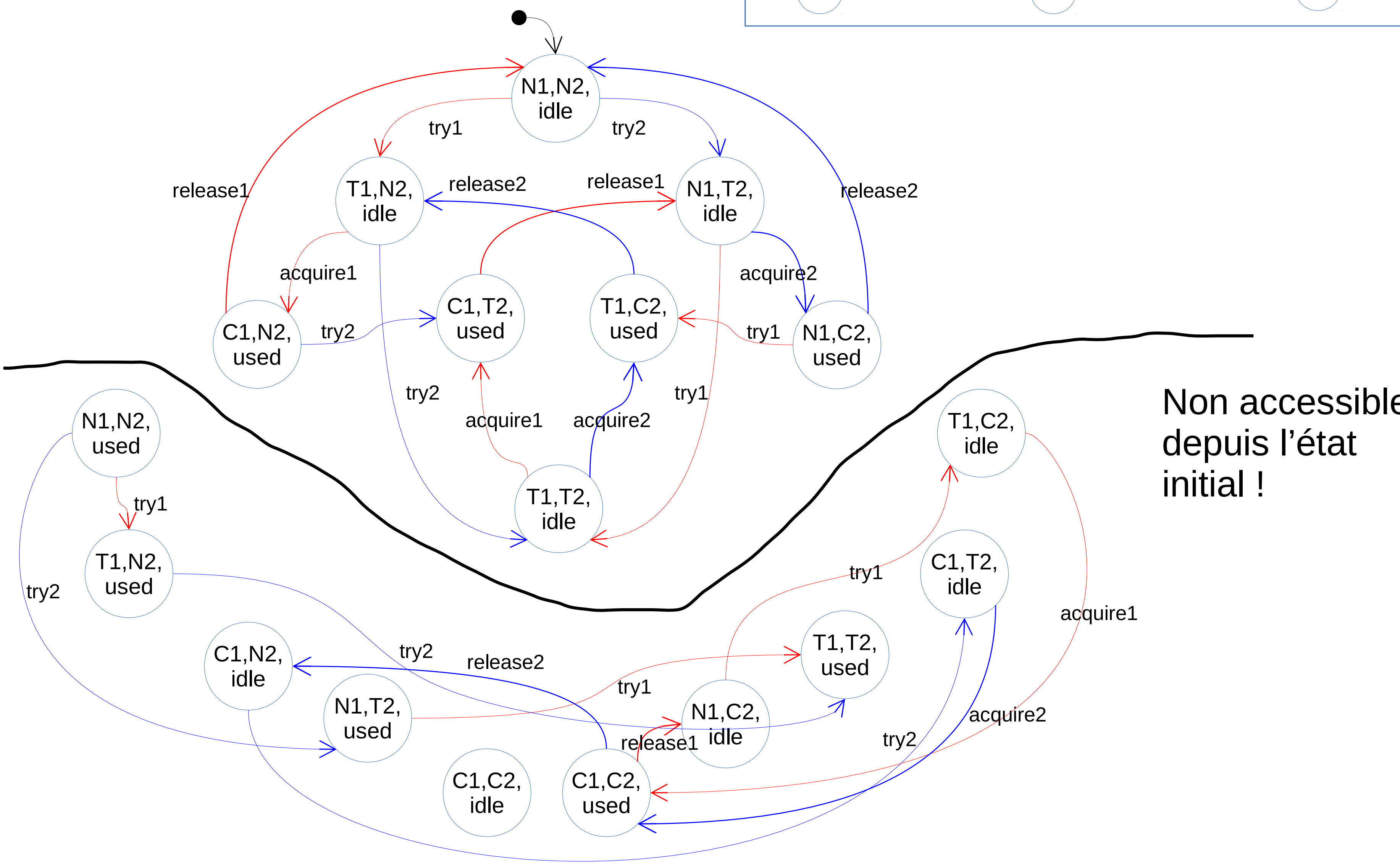




Utilisation de la composition

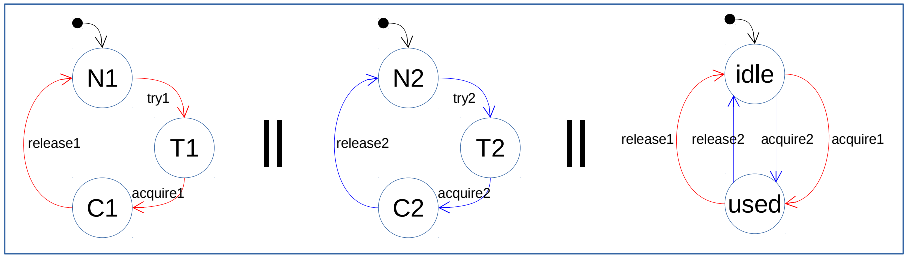
- En Model Checking

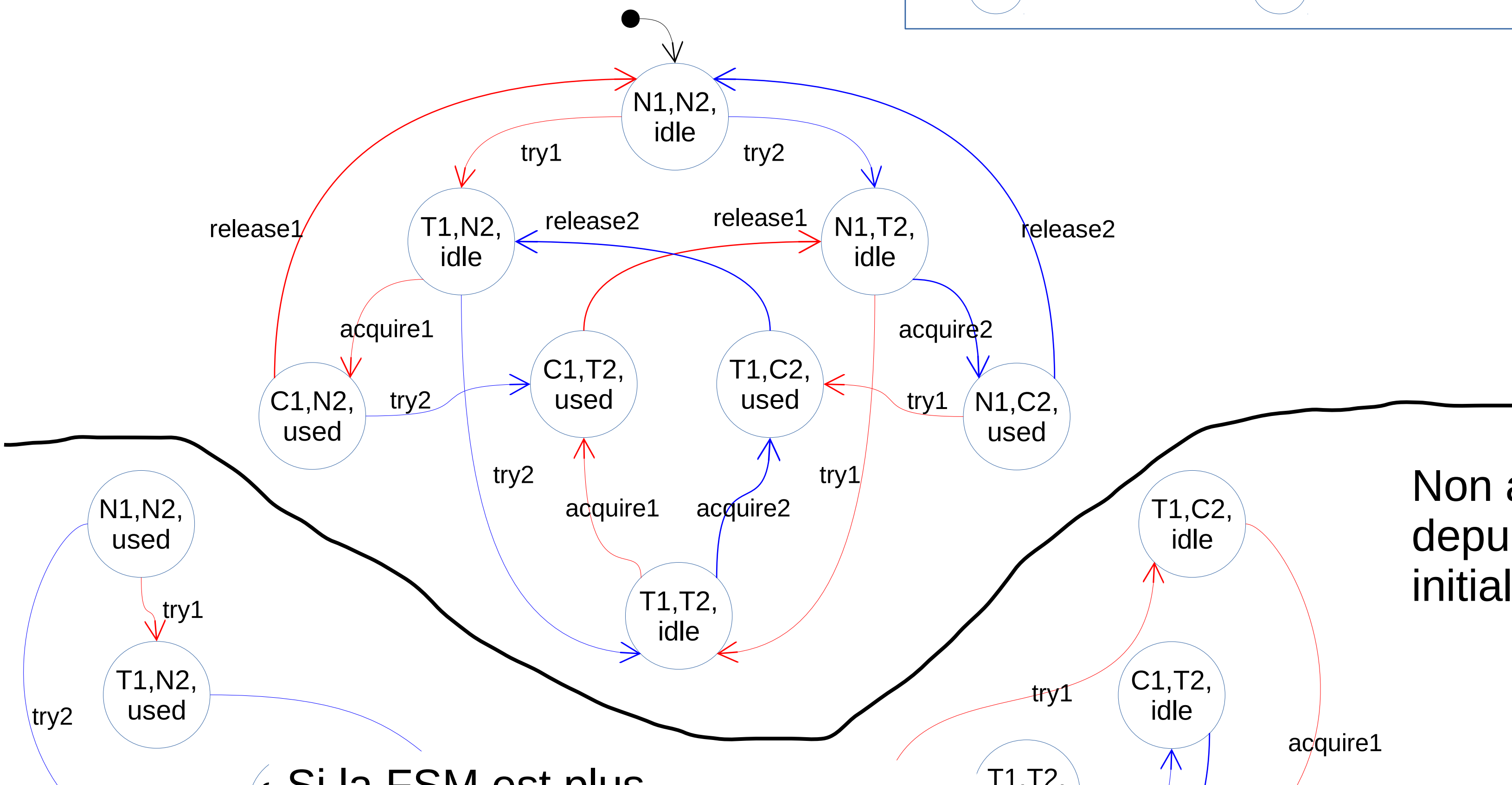




Utilisation de la composition

- En Model Checking





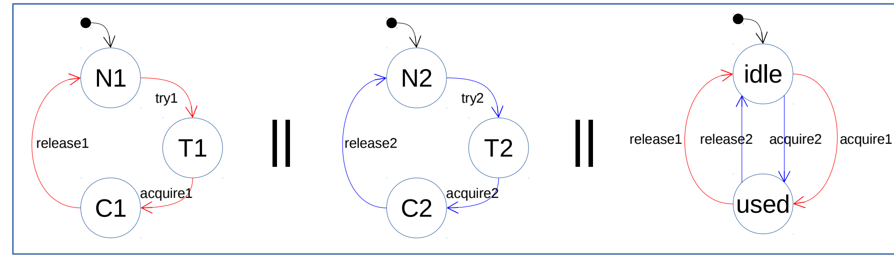
Non accessible depuis l'état initial !

Si la FSM est plus compliquée,(e.g., avec des conditions booléennes), il faut bien sûr le prendre en compte (en faisant la conjonction des conditions)

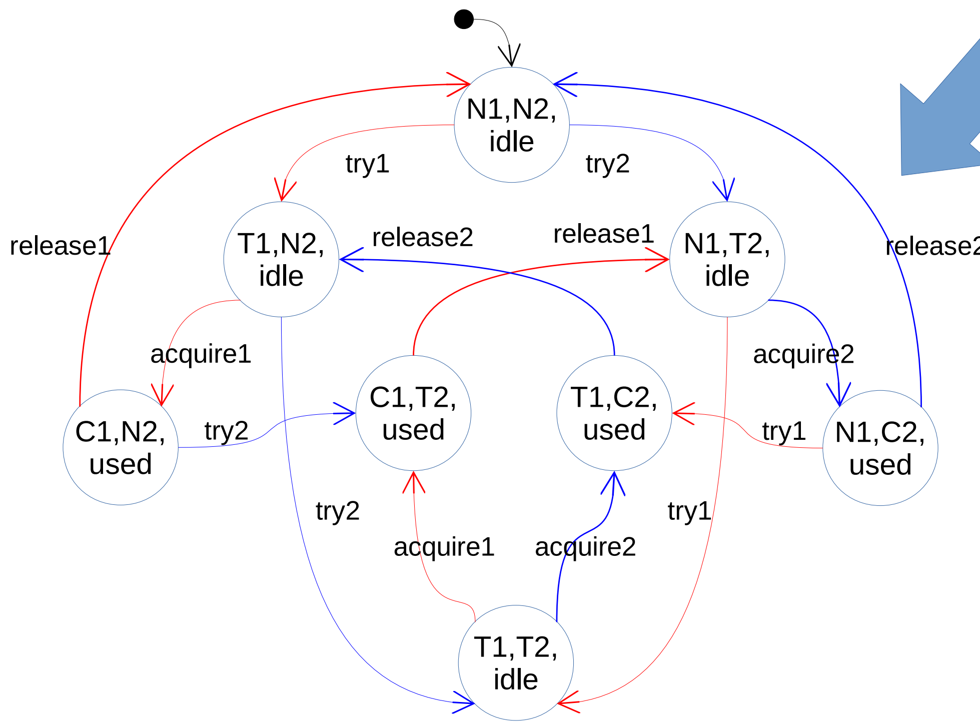
=> La simplication peut devenir compliquée…

Utilisation de la composition

- En Model Checking

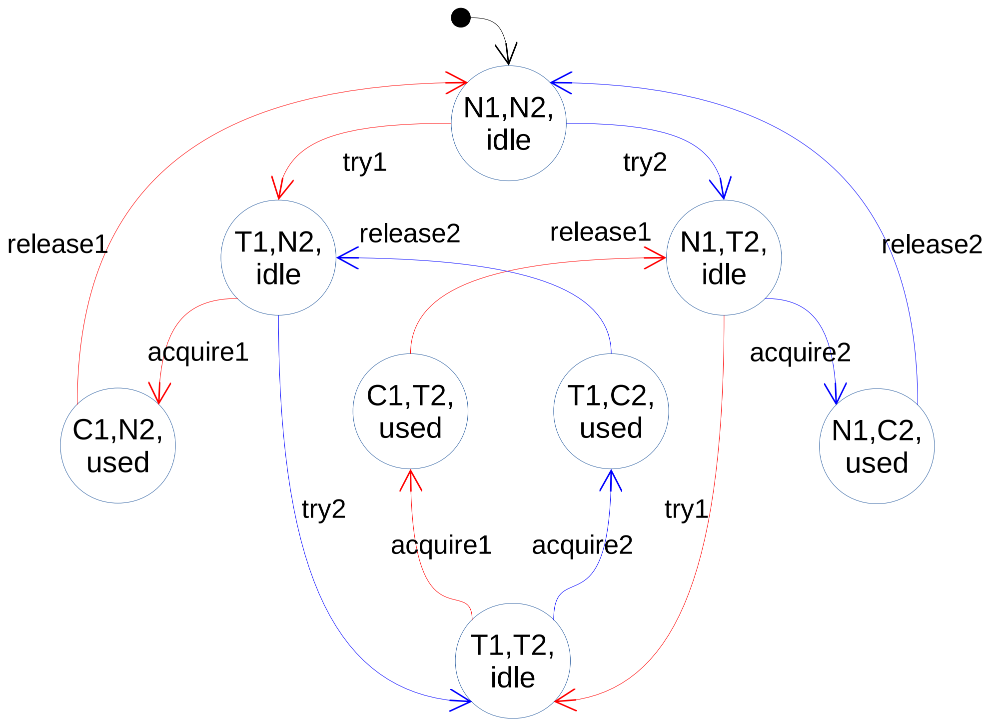


Équivalent

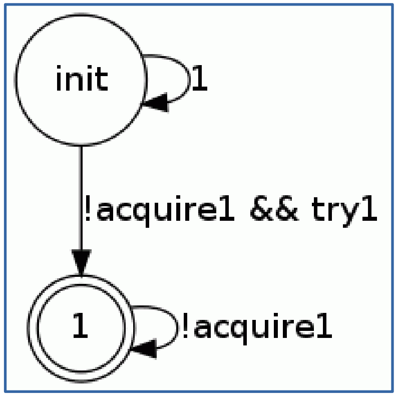


Utilisation de la composition

- En Model Checking

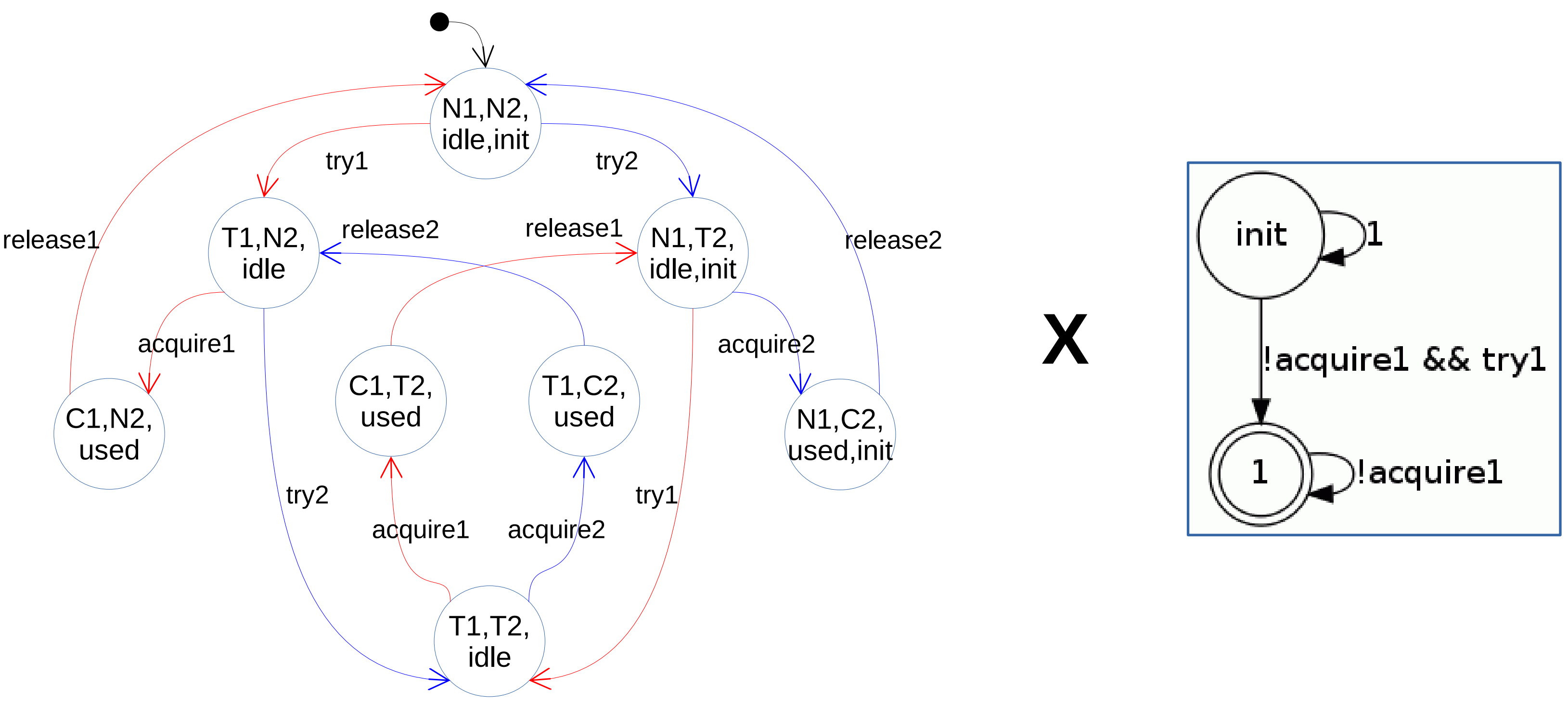


X



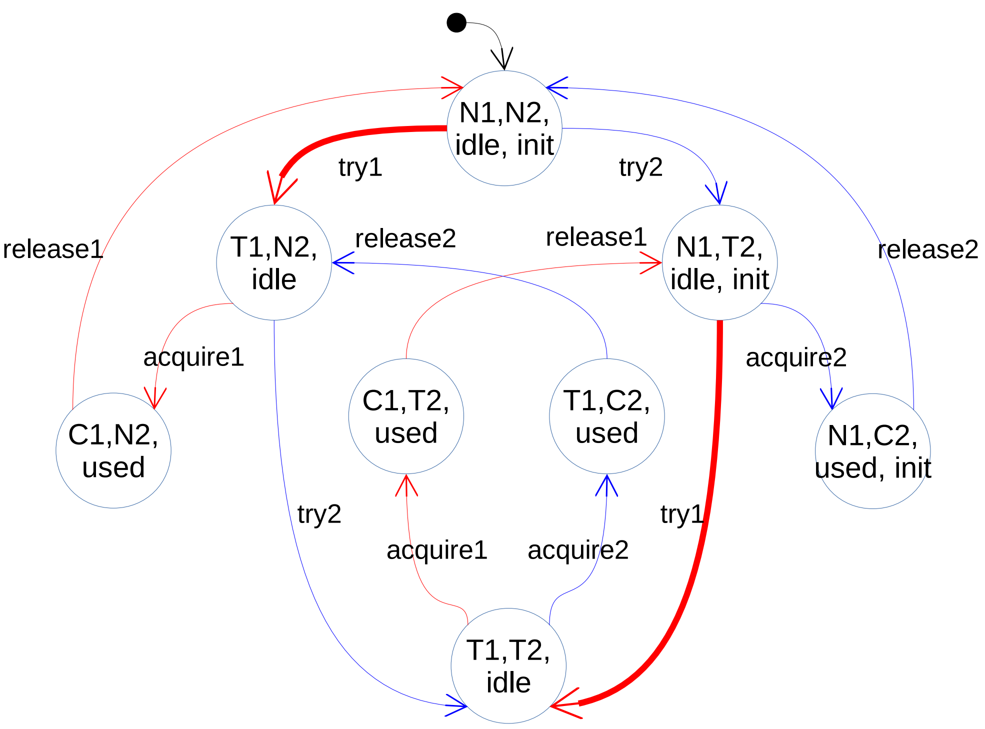
Utilisation de la composition

- En Model Checking

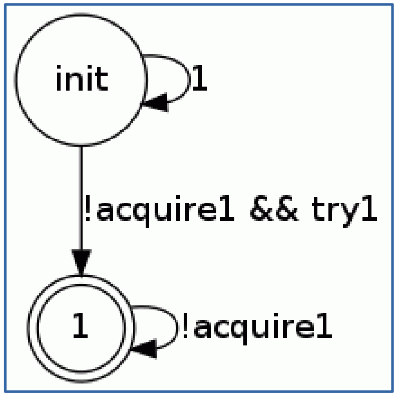


Utilisation de la composition

- En Model Checking

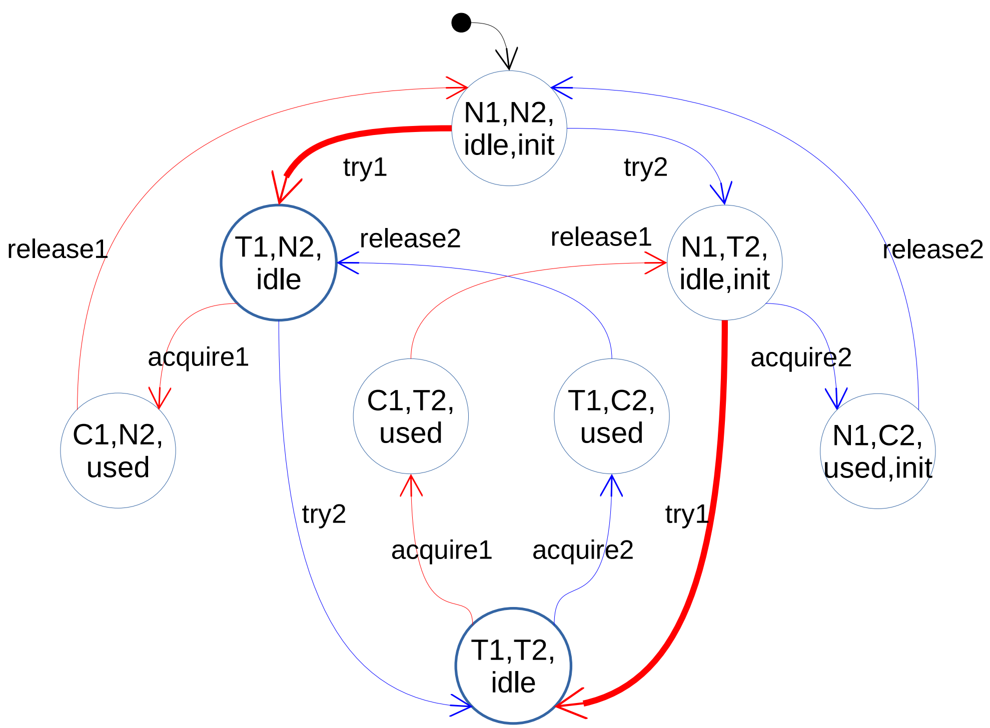


X

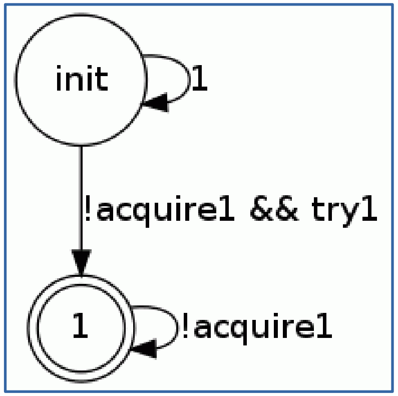


Utilisation de la composition

- En Model Checking

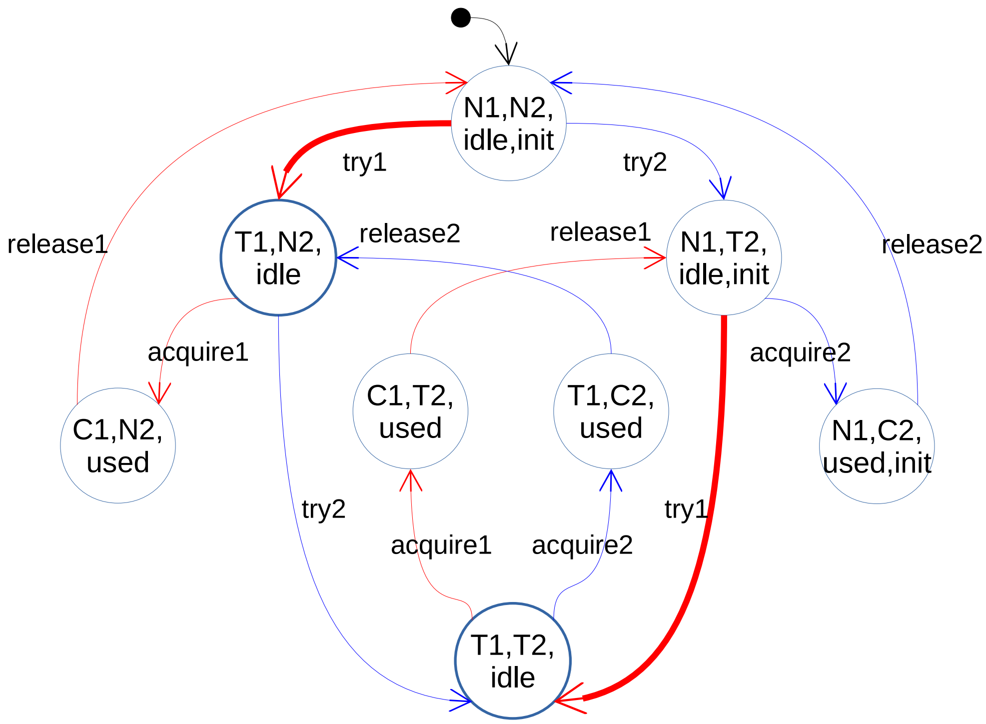


X

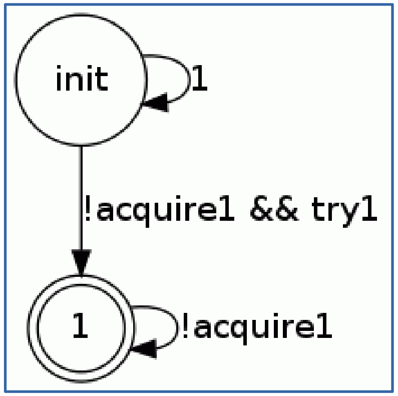


Utilisation de la composition

- En Model Checking



X

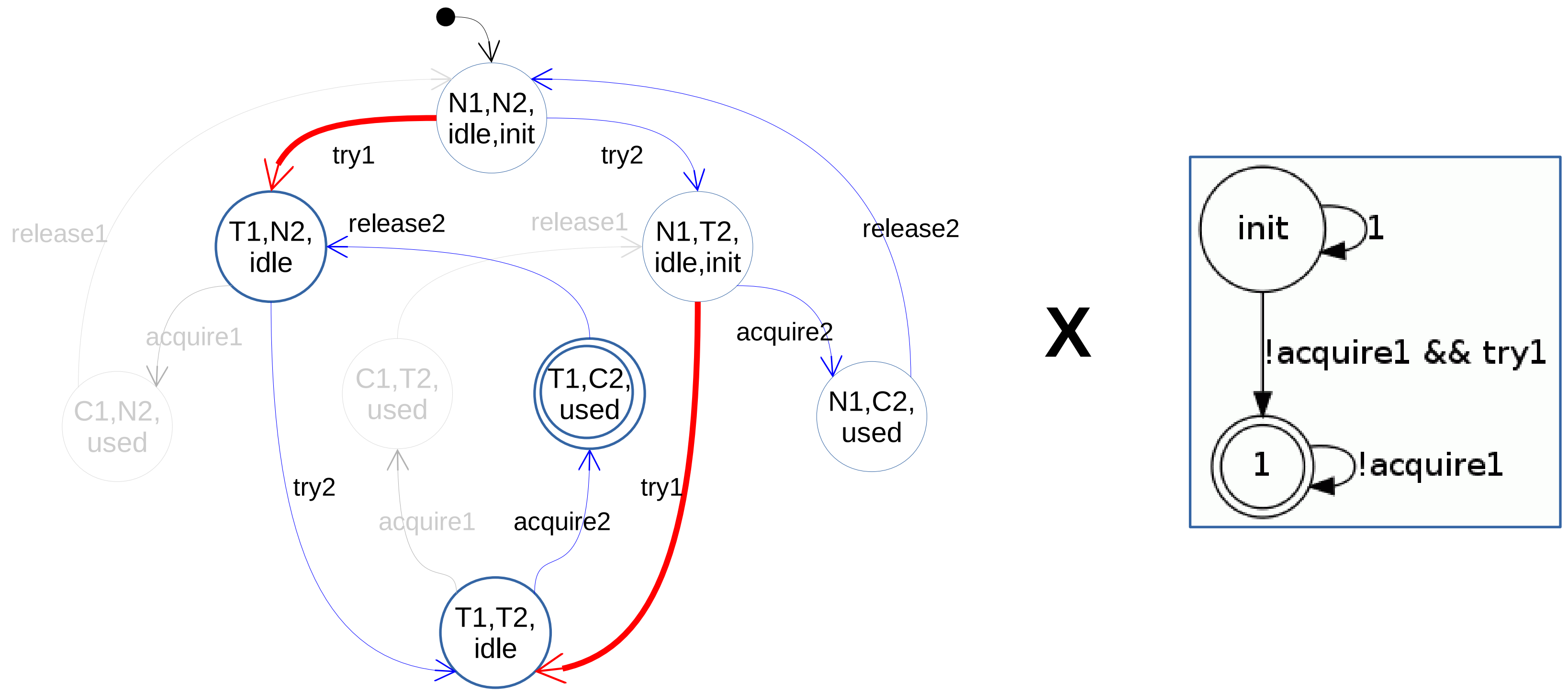


Are you fair or not ?

Fair => that if a choice is possibly infinitely often, then each branch will be chosen infinitely often

Utilisation de la composition

- En Model Checking

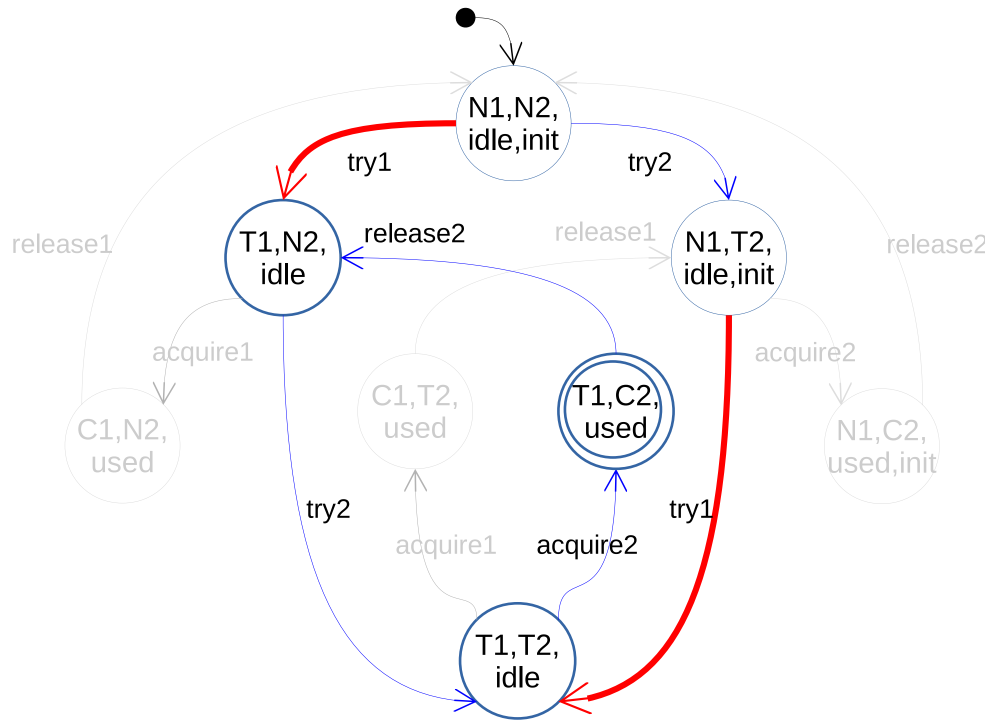


Are you fair or not ?

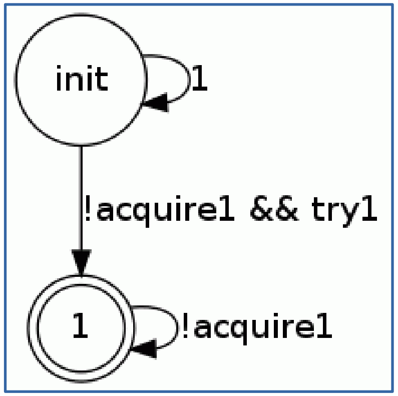
Fair => that if a choice is possibly infinitely often, then each branch will be chosen infinitely often

Utilisation de la composition

- En Model Checking



X



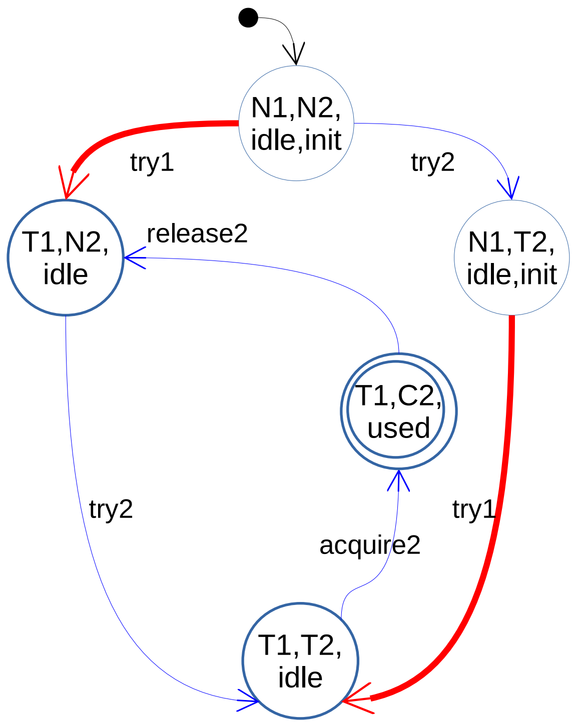
Are you fair or not ?

Fair => that if a choice is possibly infinitely often, then each branch will be chosen infinitely often

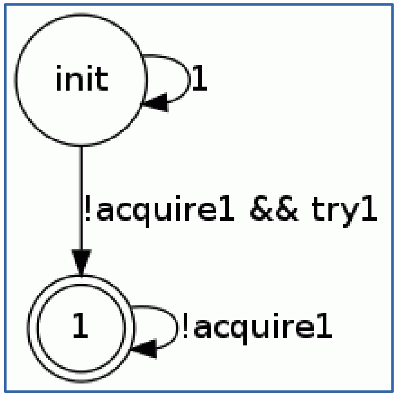
Utilisation de la composition

- En Model Checking

→ L'automate résultant possède un cycle passant par un état acceptant (le visitant infiniment souvent) donc **la propriété n'est pas vérifiée**



X

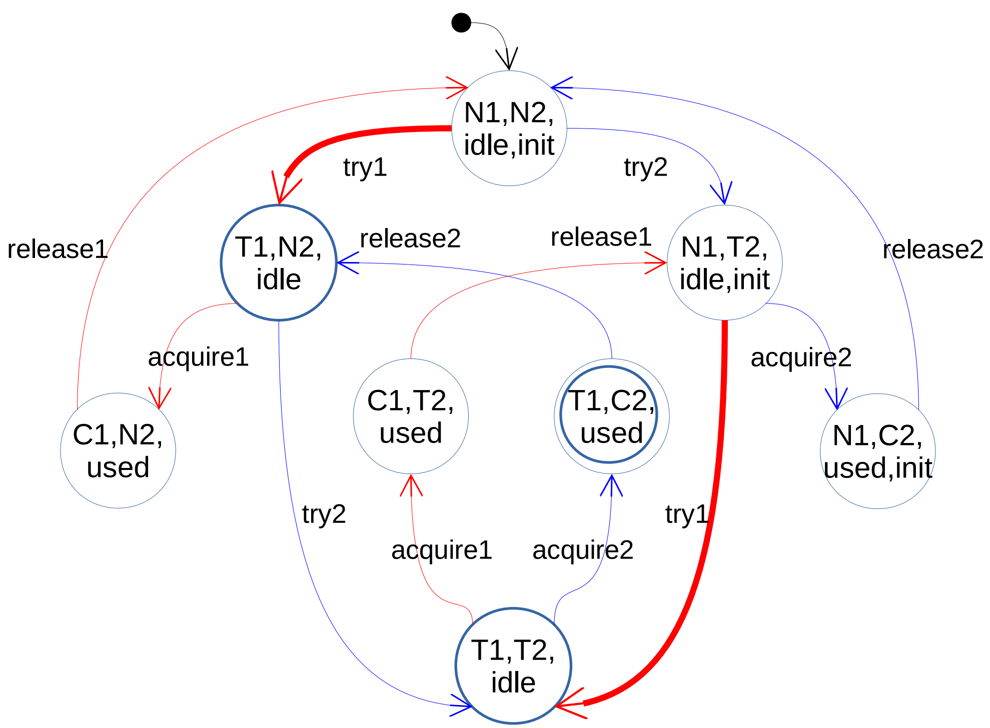


Are you fair or **not** **?**

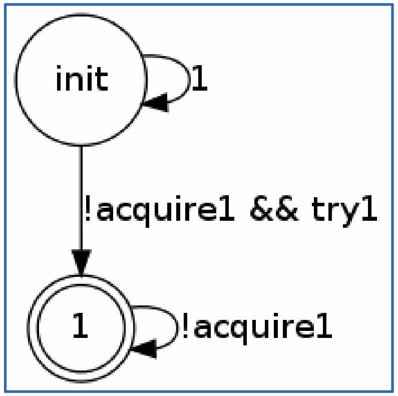
Fair => that if a choice is possibly infinitely often, then each branch will be chosen infinitely often

Utilisation de la composition

- En Model Checking



X



Are you fair or not ?

Fair => that if a choice is possibly infinitely often, then each branch will be chosen infinitely often

LTS analyser en 2 slides

- Outil académique permettant la simulation et verification de réseaux d'automates
- Disponible ici: http://www.doc.ic.ac.uk/~jnm/book/ltsa/download.html mais une version avec une meilleure layout est disponible ici: http://lvl.info.ucl.ac.be/Tools/LTSADelforge

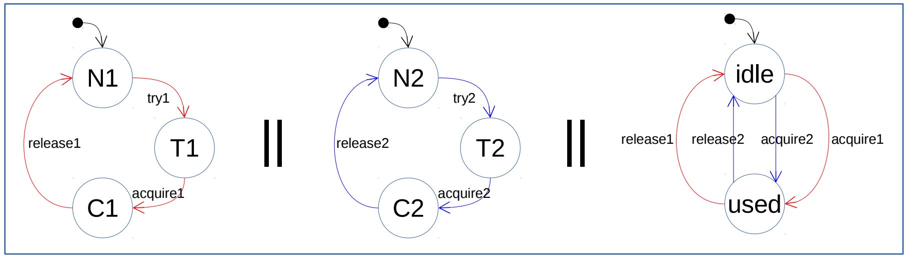
```
User1 = (try1 -> acquire1-> release1 -> User1)
User2 = (try2 -> acquire2-> release2 -> User2).

Mutex   = (acquire1 -> release1 -> Mutex
          |acquire2 -> release2 -> Mutex).

assert Prop = [] ( try1 -> (<> acquire1))
```

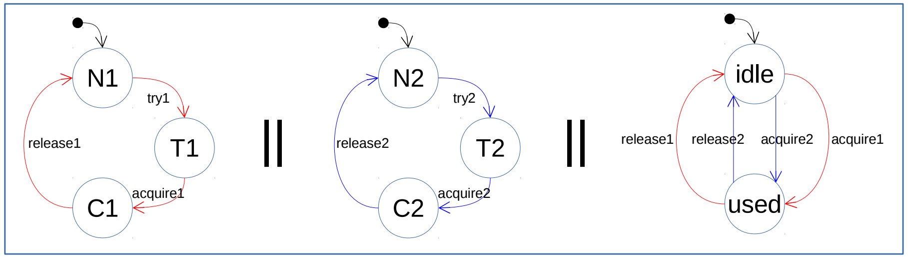
Utilisation de la composition

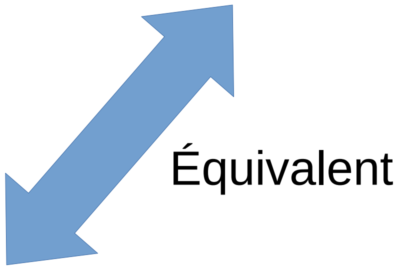
- Génération de code

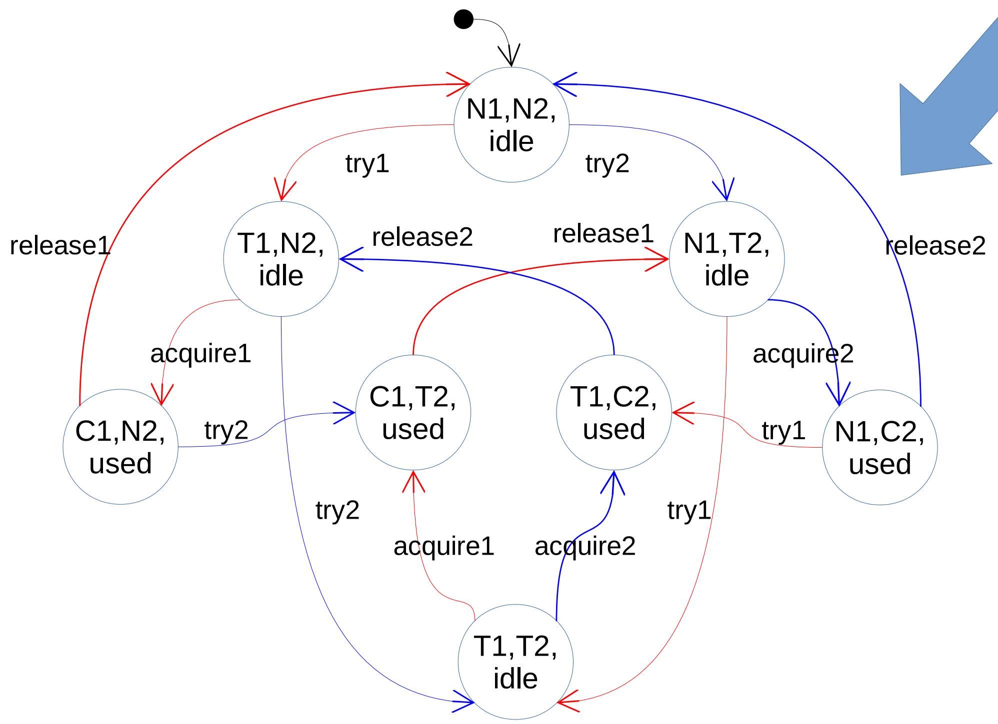


Utilisation de la composition

- Génération de code

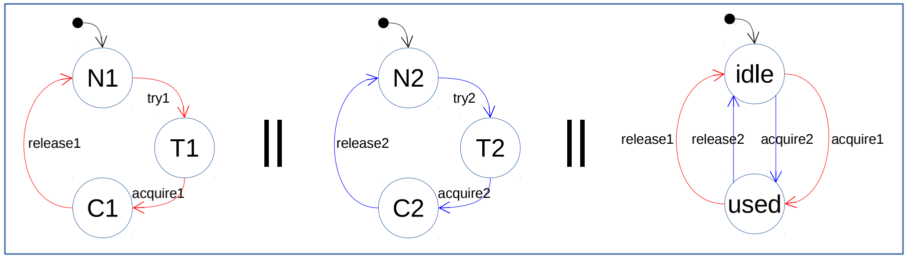


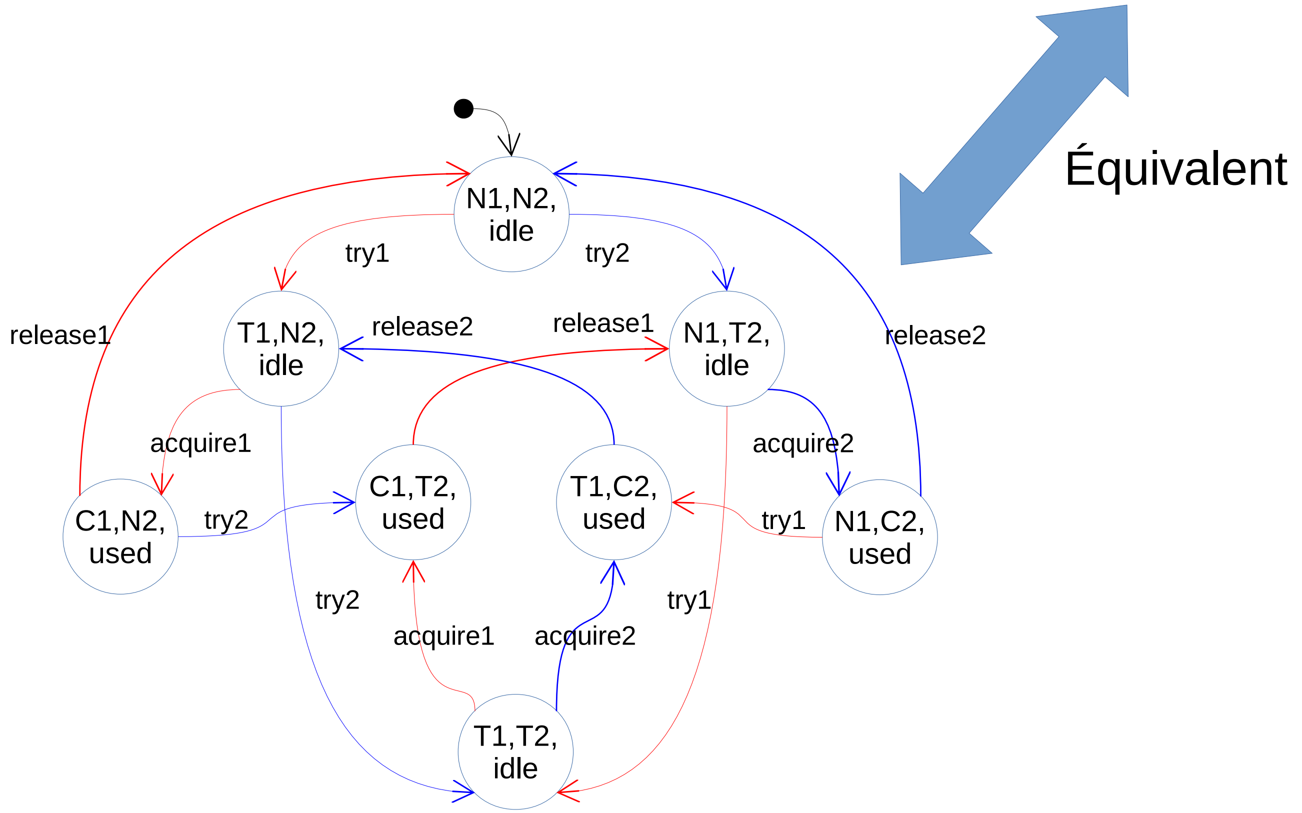




Utilisation de la composition

- Génération de code

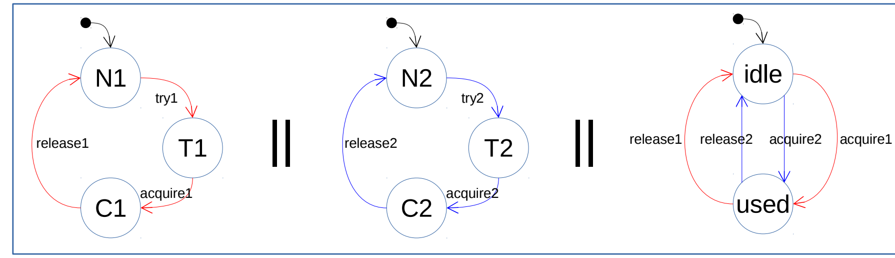


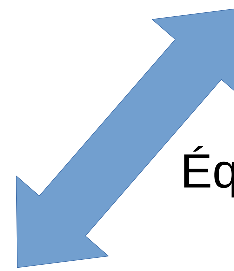


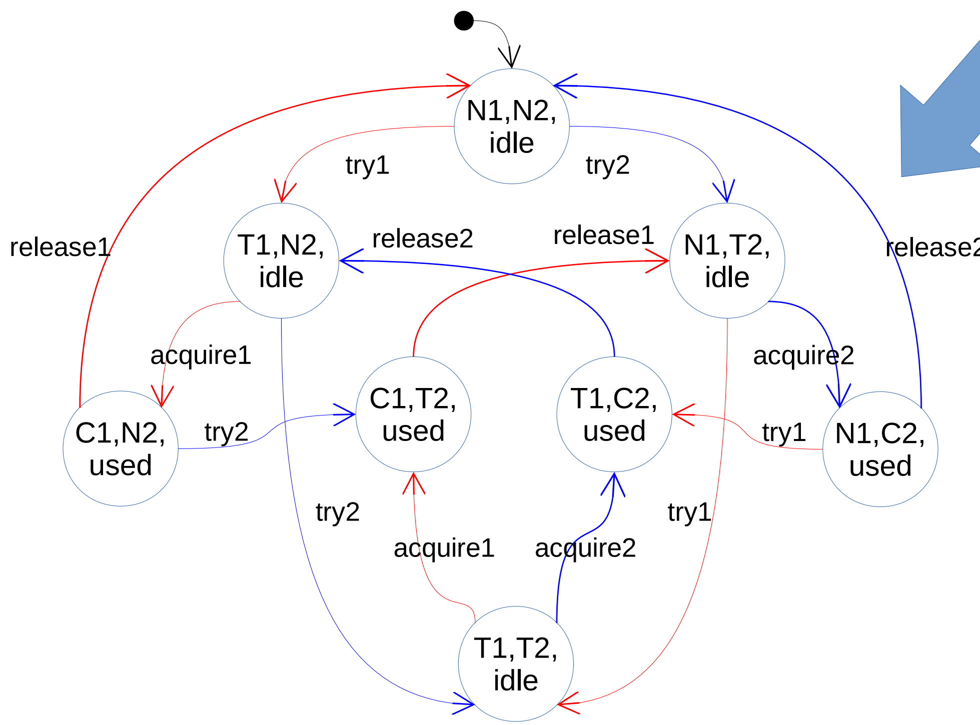
Vous savez générer le code d’un automate non parallèle et vous savez faire le produits asynchrone d’automates ==> vous savez générer le code d’automates parallèles.

Utilisation de la composition

- Génération de code





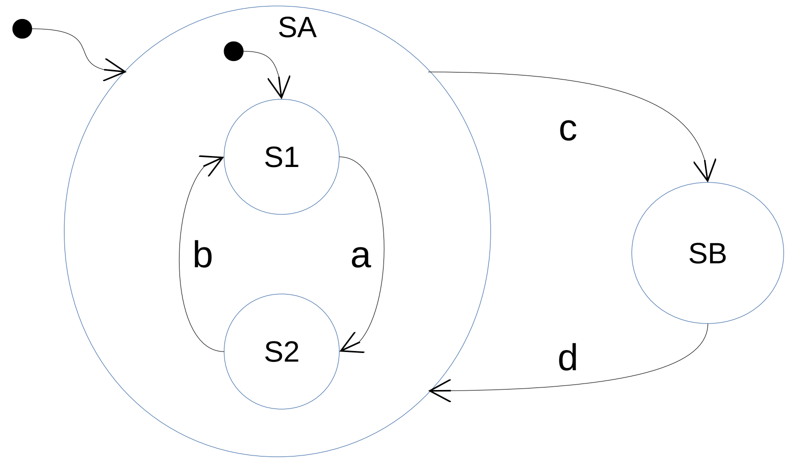


Vous savez générer le code d'un automate non parallèle et vous savez faire le produits asynchrone d'automates ==> vous savez générer le code d'automates parallèles.

attention à la sémantique de votre dialecte de FSM !
attention à l'explosion en mémoire du produit des automates !

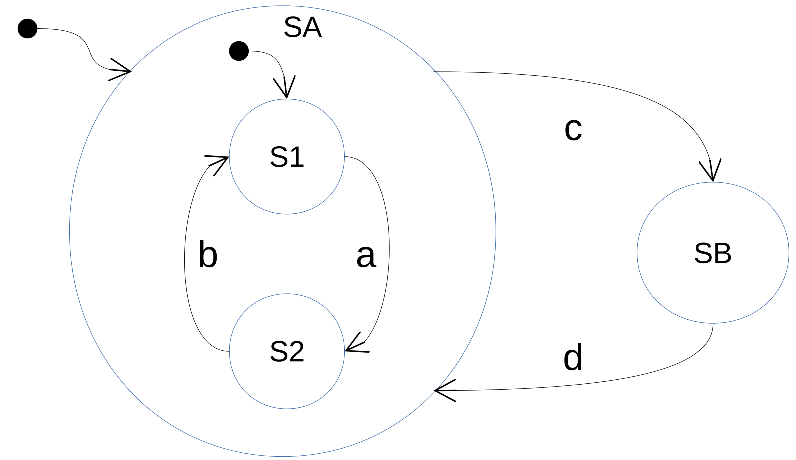
Mise à plat

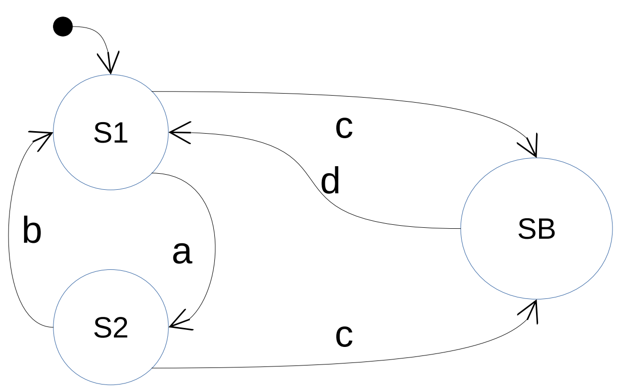
- Intuitivement: on remplace tous les états hiérachiques par leur contenu, Les alphabets restent les même, seules des actions sont mergées



Mise à plat

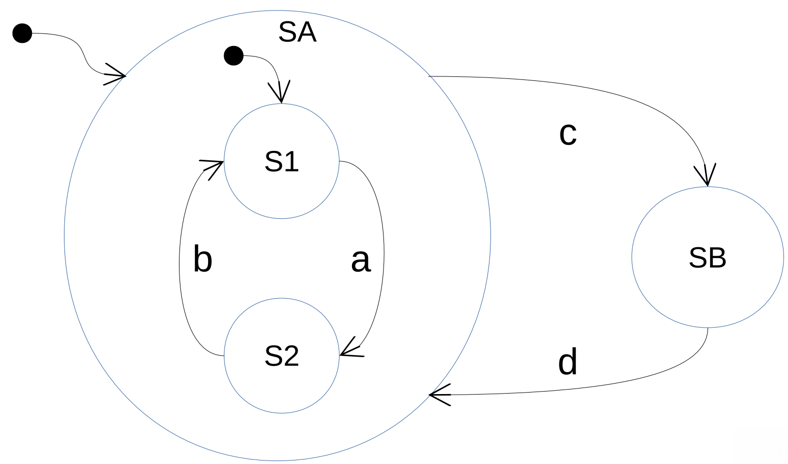
- Intuitivement: on remplace tous les états hiérachiques par leur contenu, Les alphabets restent les même, seules des actions sont mergées



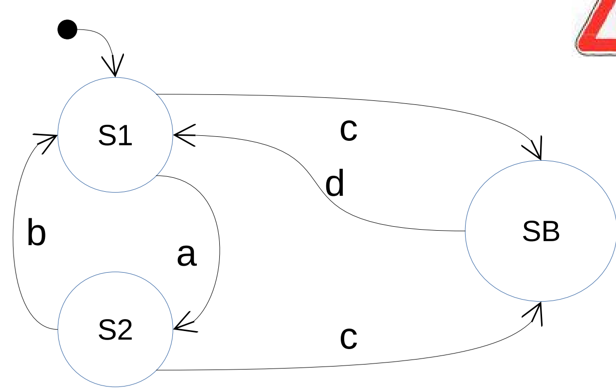


Mise à plat

- Intuitivement: on remplace tous les états hiérachiques par leur contenu, Les alphabets restent les même, seules des actions sont mergées

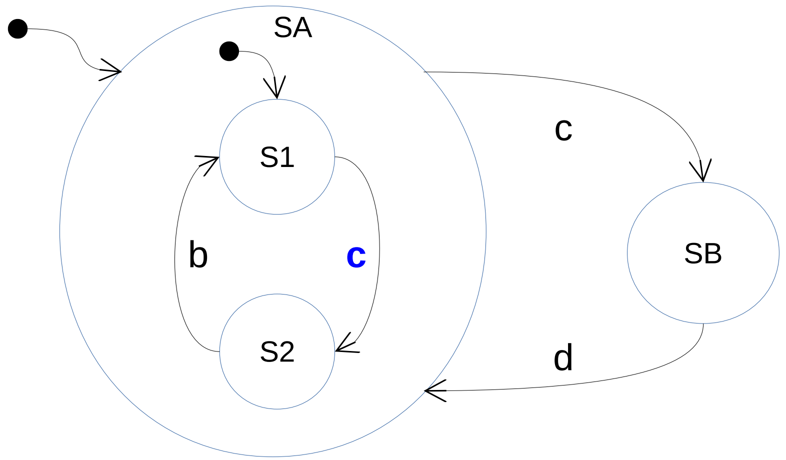


Ne pas oublier la gestion des *onEnter* / *onExit*



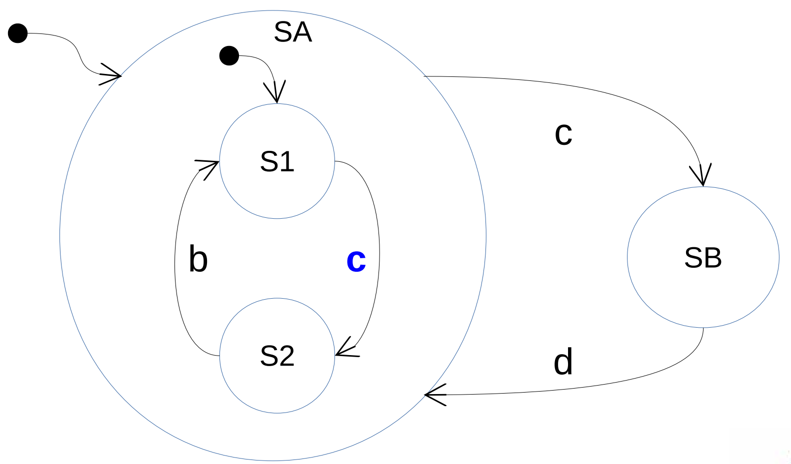
Mise à plat

- Intuitivement: on remplace tous les états hiérachiques par leur contenu, Les alphabets restent les même, seules des actions sont mergées



Mise à plat

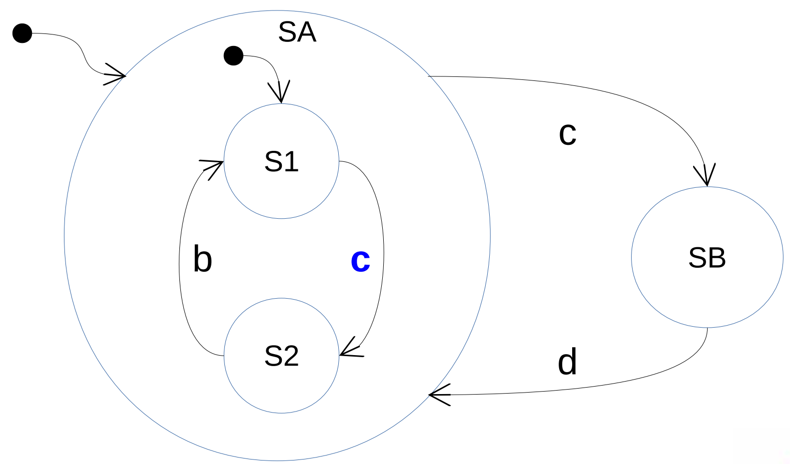
- Intuitivement: on remplace tous les états hiérachiques par leur contenu, Les alphabets restent les même, seules des actions sont mergées



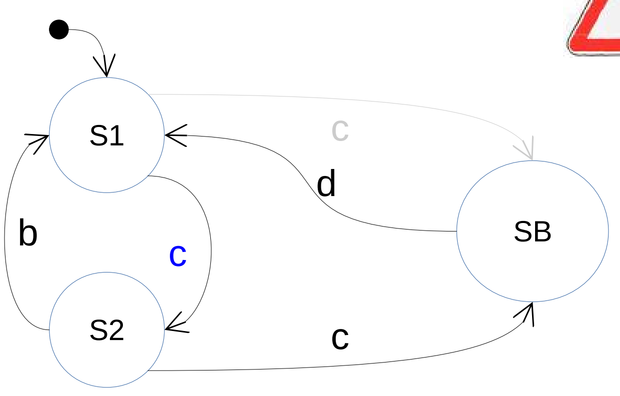
s’assurer de la sémantique de la FSM

Mise à plat

- Intuitivement: on remplace tous les états hiérachiques par leur contenu, Les alphabets restent les même, seules des actions sont mergées

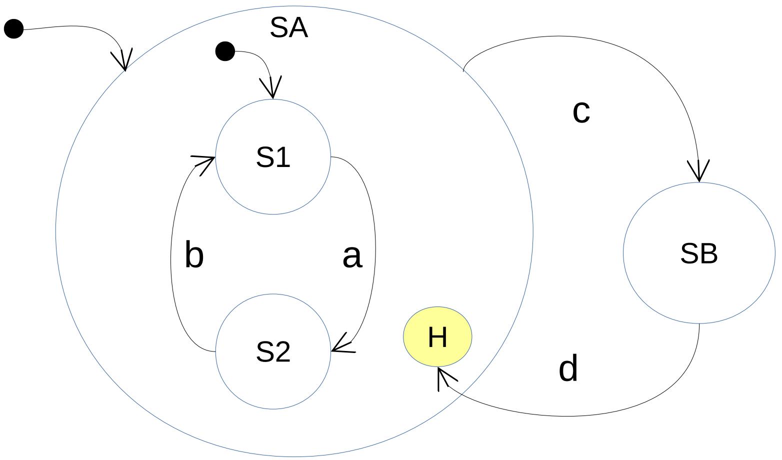


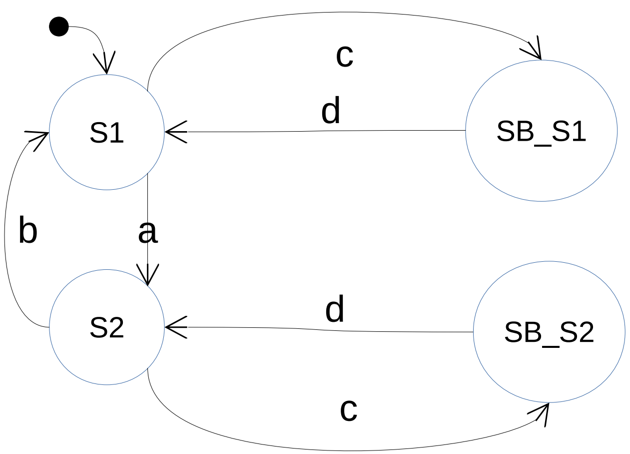
s’assurer de la sémantique de la FSM



Mise à plat

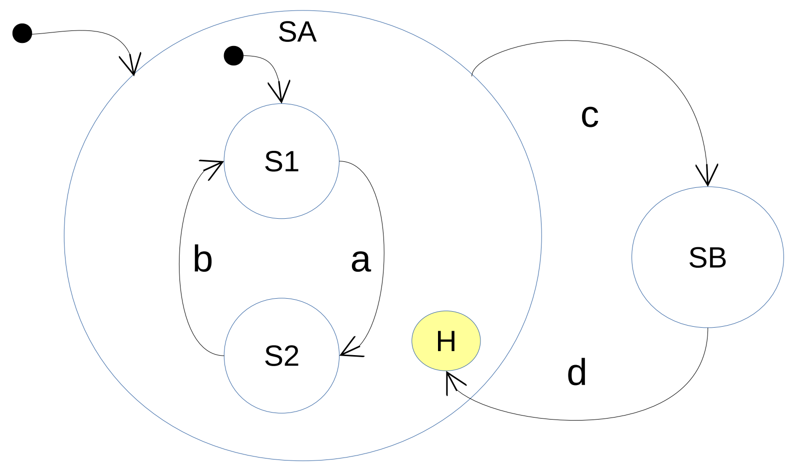
- Intuitivement: on remplace tous les états hiérachiques par leur contenu, Les alphabets restent les même, seules des actions sont mergées



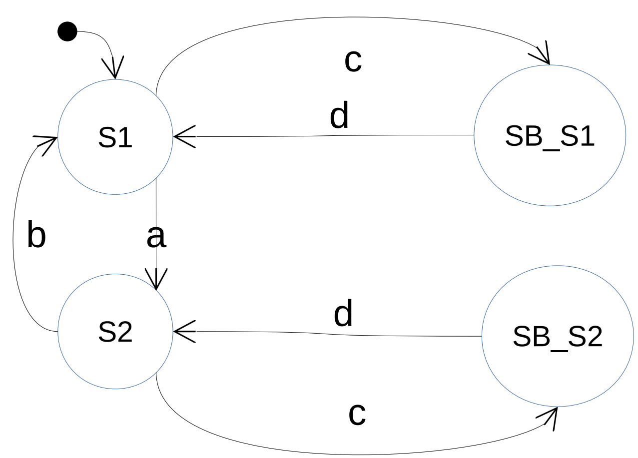


Mise à plat

- Intuitivement: on remplace tous les états hiérachiques par leur contenu, Les alphabets restent les même, seules des actions sont mergées



Etc...



Conclusion

- Il existe une littérature importante sur la manipulation de state machine à états finis. Derrière cette littérature ce cache une multitude de dialectes plus ou moins complexes.
- La difficulté de manipulation dépend beaucoup de l’expressivité du dialecte et de sa concision.
- Il est possible de passer de “state chart” à des FSM simples mais le coup est exponentiel
- Les FSM devraient au minimum vous aider à structurer votre pensée, au mieux à structurer/générer/vérifier/valider le code de contrôle de vos programmes.