Contrôleurs de sécurité : une extension

Hervé Grall

Équipe OBASCO (EMN/INRIA – LINA)
École des mines de Nantes

septembre 2005
Nantes

Plan

Plan

Objectifs

- Contrôler l'exécution d'un système pour assurer des propriétés de disponibilité
  $\longrightarrow$ modèle simple
  - des évènements (atomiques) : ils correspondent typiquement à des services, qui produisent et consomment des ressources
  - un système : son exécution est décrite par une suite (finie ou infinie) d'évènements
  - un contrôleur de sécurité : en entrée une suite d'évènements, en sortie une suite contrôlée d'évènements
    $\longrightarrow$ exécution potentielle $\mapsto$ exécution réelle (après contrôle)

Objectifs

- Question 1 : quelle propriété un contrôleur garantit-il en sortie ?
- Question 2 : comment définir un contrôleur de sécurité ?

Une solution

- Solution développée à partir d'un exemple simple, mais paradigmatique
- Caractérisation des propriétés de sécurité garanties : au delà des propriétés de sûreté $\longrightarrow$ disponibilité
- Définition des contrôleurs par automates transducteurs

Discussion des hypothèses

Quel rapport avec des réalités concrètes ?

- Un exemple concret : un programme appelant une bibliothèque
  $\longrightarrow$ Le contrôleur devient un proxy (mandataire), intermédiaire avec la bibliothèque.

Discussion des hypothèses

Quel rapport avec des réalités concrètes ?

- Proche de la programmation par composants (généralisation de l'exemple précédent).
  Le système : un composant envoyant des messages à un autre composant.
  $\longrightarrow$ Le contrôleur gère la réception des messages (avec d'éventuelles synchronisations).

Discussion des hypothèses

Quel rapport avec des réalités concrètes ?

- Proche de la programmation par aspects.
  "Event-based Aspect Oriented Programming".
  $\longrightarrow$ Le contrôleur devient le tisseur.

Plan

Le service

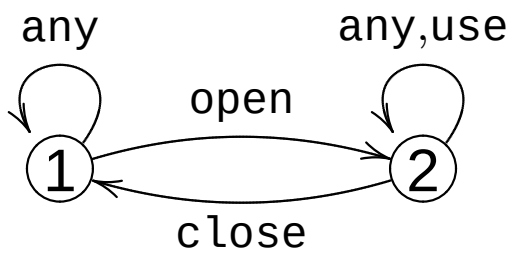
Le système : programme ouvrant un service, l'utilisant et le fermant.
Évènements possibles :

```
open
use
close
```

On ajoute une action générique `any`, représentant toute action possible autre que celles précédant.

Le protocole d'accès au service

Protocole ouverture–usage–fermeture



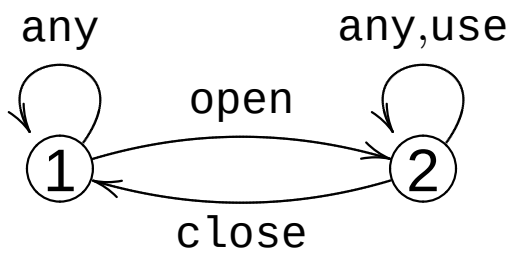
Traces finies reconnues :

$(\mathtt{any} + \mathtt{open}.(\mathtt{any} + \mathtt{use})^*.\mathtt{close})^*$

$(\mathtt{any} + \mathtt{open}.(\mathtt{any} + \mathtt{use})^*.\mathtt{close})^*.\mathtt{open}.(\mathtt{any} + \mathtt{use})^*$

Le protocole d'accès au service

Protocole ouverture–usage–fermeture



Traces infinies reconnues :

$(\mathtt{any} + \mathtt{open}.(\mathtt{any} + \mathtt{use})^*.\mathtt{close})^\omega$

$(\mathtt{any} + \mathtt{open}.(\mathtt{any} + \mathtt{use})^*.\mathtt{close})^*.\mathtt{open}.(\mathtt{any} + \mathtt{use})^\omega$

Premier problème

On voudrait vérifier la propriété suivante :

- si l'exécution se termine, alors le service est toujours fermé après ouverture :

$$(\mathtt{any} + \mathtt{open}.(\mathtt{any} + \mathtt{use})^*.\mathtt{close})^*$$

Interprétation en termes de disponibilité : la fermeture rend disponible le service à d'autres systèmes, régulièrement puis finalement.

Second problème

On voudrait vérifier la propriété suivante :

- si l'exécution ne se termine pas, alors le service est toujours fermé après ouverture :

$$(\mathtt{any} + \mathtt{open}.(\mathtt{any} + \mathtt{use})^*.\mathtt{close})^\omega$$

Interprétation en terme de disponibilité : la fermeture rend disponible le service à d'autres systèmes, finalement ou régulièrement.

Troisième problème

On voudrait vérifier la propriété suivante :

- si l'exécution ne se termine pas, alors le service est toujours fermé après ouverture et le service est ouvert une infinité de fois :

$$(\mathtt{any}^*.\mathtt{open}.(\mathtt{any} + \mathtt{use})^*.\mathtt{close})^\omega$$

Interprétation en terme de disponibilité : la fermeture rend disponible le service à d'autres systèmes, régulièrement mais pas finalement.

Plan

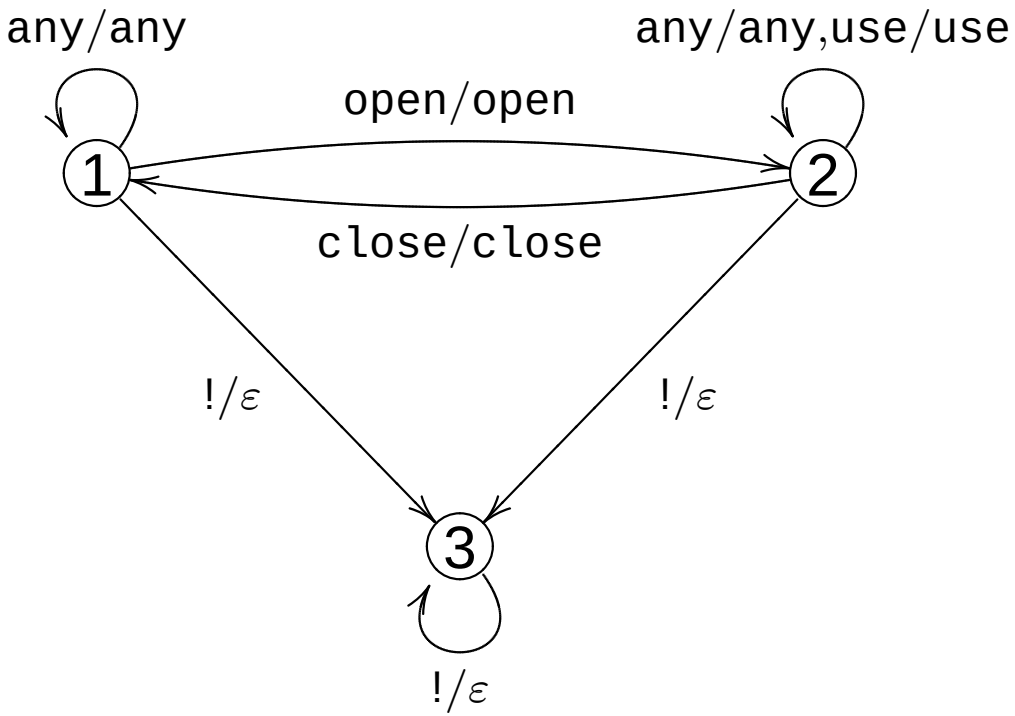
Le modèle usuel

Contrôleur de sécurité à la Schneider : il se déduit du protocole.
Deux comportements possibles :

- il accepte l'évènement en entrée et l'exécute,
- il refuse l'évènement en entrée et termine le programme.

Le contrôleur peut se représenter par un transducteur.

Le modèle usuel



Limitations du modèle usuel

Un contrôleur de sécurité à la Schneider garantit seulement des propriétés de sûreté.
$\rightarrow$ Aucun des trois problèmes précédents ne peut être résolu.
Vers une solution :

- propriétés de sûreté et contrôleurs
- classification des problèmes précédents
- amélioration du modèle de contrôleurs

Plan

Propriétés et traces

A : ensemble fini d'évènements $(0, 1, \ldots)$

P propriété $\overset{def}{\Leftrightarrow}$ ensemble de traces finies ou infinies

$\overset{def}{\Leftrightarrow}$ $P \subseteq A^* \cup A^\omega$

$\rightarrow$ deux projections :

1. P_* : ensemble des traces finies de P
2. P_ω : ensemble des traces infinies de P

Deux opérateurs sur les propriétés :

1. **Prefix**(P) : ensemble des traces finies préfixes d'une trace de P
2. **Lim**(P_*) : ensemble des traces infinies obtenues comme limite de traces finies dans P_* (limite d'Eilenberg)
   $a_1 \ldots a_{n_1} \ldots a_{n_i} \ldots$, avec pour tout i, $a_1 \ldots a_{n_i}$ dans P_*

Sûreté

- Sûreté finitaire : ça se passe bien pour une trace finie si et seulement si pour tout préfixe, ça se passe bien.

  P_* sûreté $\overset{def}{\Leftrightarrow}$ P_* est fermée par préfixe

  $\overset{def}{\Leftrightarrow}$ $\mathbf{Prefix}(P_*) = P_*$

- Sûreté infinitaire : ça se passe bien pour une trace infinie si et seulement pour tout préfixe, ça peut bien se passer.

  P_ω sûreté $\overset{def}{\Leftrightarrow}$ toute limite de traces préfixes de traces dans P_ω appartient à P_ω

  $\overset{def}{\Leftrightarrow}$ $\mathbf{Lim}(\mathbf{Prefix}(P_\omega)) = P_\omega$

Sûreté

- Sûreté (cas général)

  P sûreté $\overset{def}{\Leftrightarrow}$ P est fermée par préfixe et toute limite de préfixes de traces dans P appartient à P

  $\overset{def}{\Leftrightarrow}$ $\mathbf{Prefix}(P) = P_*$
  $\mathbf{Lim}(\mathbf{Prefix}(P)) = P_\omega$

- Les contrôleurs de sécurité à la Schneider garantissent en sortie exactement les propriétés de sûreté.

Sûreté

- P sûreté $\Rightarrow$ P_* et P_ω sûretés finitaire et infinitaire
- La réciproque est fausse : 1^* sûreté finitaire, 0^ω sûreté infinitaire, mais $1^* + 0^\omega$ n'est pas une sûreté.
  $\rightarrow$ Il est nécessaire de coordonner les traces finies et les traces infinies.
- P sûreté $\Leftrightarrow$ P_* et P_ω sûretés finitaire et infinitaire
  $\mathbf{Prefix}(P_\omega) \subseteq P_*$
  $\mathbf{Lim}(\mathbf{Prefix}(P_*)) \subseteq P_\omega$

Classification des problèmes

- Premier problème
  Si l'exécution se termine, alors le service est toujours fermé après ouverture :

$$(\mathtt{any} + \mathtt{open}.(\mathtt{any} + \mathtt{use})^*.\mathtt{close})^*$$

  $\longrightarrow$ Ce n'est pas une sûreté finitaire (non fermée par préfixe).

Classification des problèmes

- Second problème
  Si l'exécution ne se termine pas, alors le service est toujours fermé après ouverture :

$$(\mathsf{any} + \mathsf{open}.(\mathsf{any} + \mathsf{use})^*.\mathtt{close})^\omega$$

$\longrightarrow$ Ce n'est pas une sûreté infinitaire :

$$\mathsf{open}.\mathsf{use}^\omega \in \mathbf{Lim}(\mathbf{Prefix}(P))$$

Classification des problèmes

- Troisième problème
  Si l'exécution ne se termine pas, alors le service est toujours fermé après ouverture et le service est ouvert une infinité de fois :

$$(\mathtt{any}^*.\mathtt{open}.(\mathtt{any} + \mathtt{use})^*.\mathtt{close})^\omega$$

  $\rightarrow$ Ce n'est pas une sûreté infinitaire :

$$\mathtt{any}^\omega, \mathtt{open}.\mathtt{use}^\omega \in \mathbf{Lim}(\mathbf{Prefix}(P))$$

Vivacité

- Vivacité finitaire : pour toute trace finie, ça peut bien se passer.

  P_* vivacité $\overset{def}{\Leftrightarrow}$ toute trace finie est préfixe d'une trace de P_*

  $\overset{def}{\Leftrightarrow}$ $\mathbf{Prefix}(P_*) = A^*$

- Vivacité infinitaire : pour toute trace finie, ça peut bien se passer.

  P_ω vivacité $\overset{def}{\Leftrightarrow}$ toute trace finie est préfixe d'une trace de P_ω

  $\overset{def}{\Leftrightarrow}$ $\mathbf{Prefix}(P_\omega) = A^*$

Vivacité

- Vivacité (cas général)

  P vivacité $\overset{def}{\Leftrightarrow}$ toute trace finie est préfixe d'une trace de P

  $\overset{def}{\Leftrightarrow}$ $\mathbf{Prefix}(P) = A^*$

Vivacité

- P_* et P_ω vivacités $\Rightarrow$ P vivacité
- La réciproque est fausse : A^ω vivacité mais $\emptyset$ n'est pas une vivacité finitaire.
  $\rightarrow$ Il est nécessaire de coordonner les traces finies et les traces infinies.
- P vivacité
  Prefix$(P_\omega) =$ **Prefix**(P_*) $\Leftrightarrow$ P_* et P_ω vivacités finitaire et infinitaire

Décomposition des propriétés

Théorème d'Alpern-Schneider :
Toute propriété est égale à l'intersection d'une propriété de sûreté et d'une propriété de vivacité.

- Les trois propriétés de disponibilités précédentes ont chacune une composante de vivacité.
- C'est généralement le cas pour les propriétés de disponibilité.

Plan

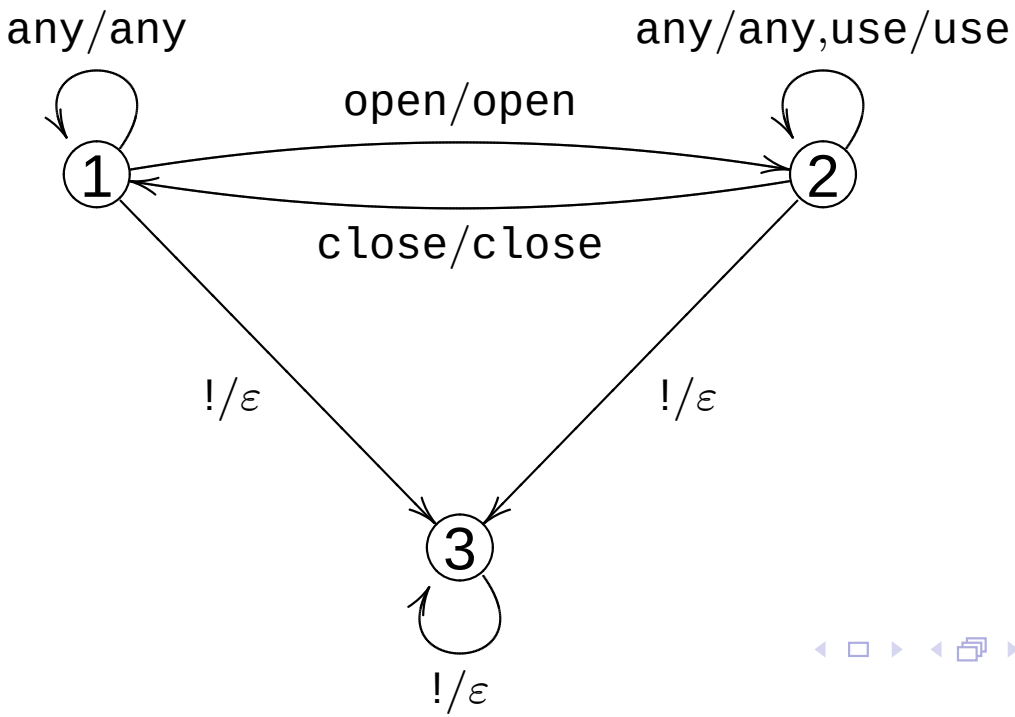
Nature du problème

Étant donné une propriété P,
définir un contrôleur de sécurité sous la forme d'un transducteur vérifiant :

- SORTIE : le contrôleur garantit en sortie exactement la propriété P,
- NEUTRALITÉ : le contrôleur accepte sans la modifier une trace appartenant à la propriété P.

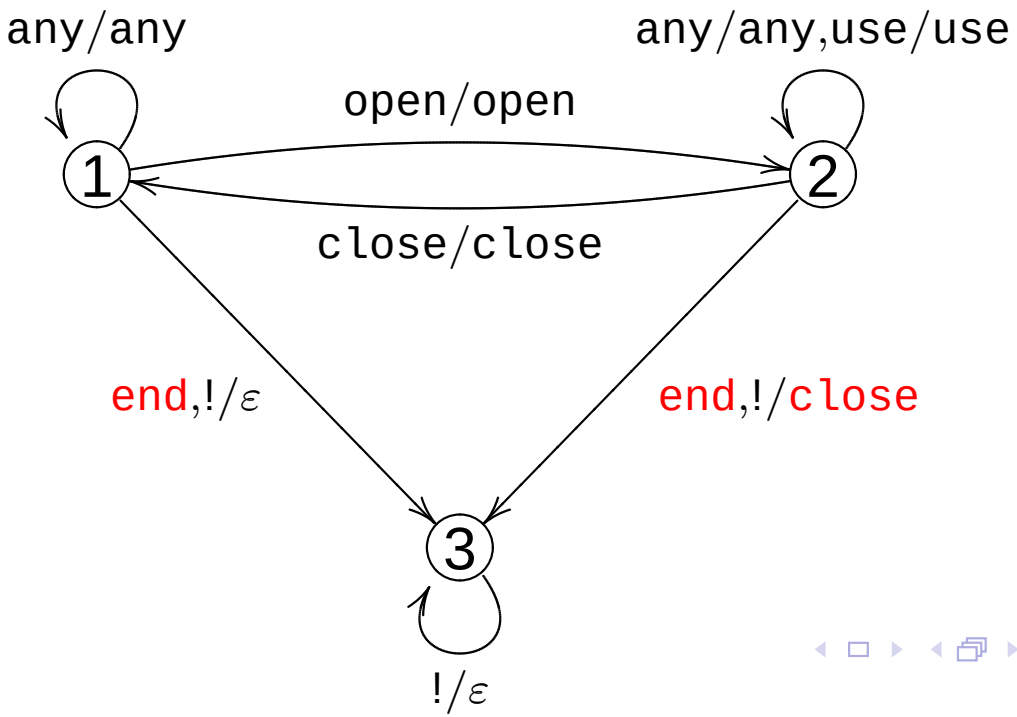
Résolution du premier problème

Si l'exécution se termine, alors le service est toujours fermé après ouverture : $(\mathtt{any} + \mathtt{open}.(\mathtt{any} + \mathtt{use})^*.\mathtt{close})^*$



Résolution du premier problème

Si l'exécution se termine, alors le service est toujours fermé après ouverture : $(\mathtt{any} + \mathtt{open}.(\mathtt{any} + \mathtt{use})^*.\mathtt{close})^*$



Résolution du premier problème

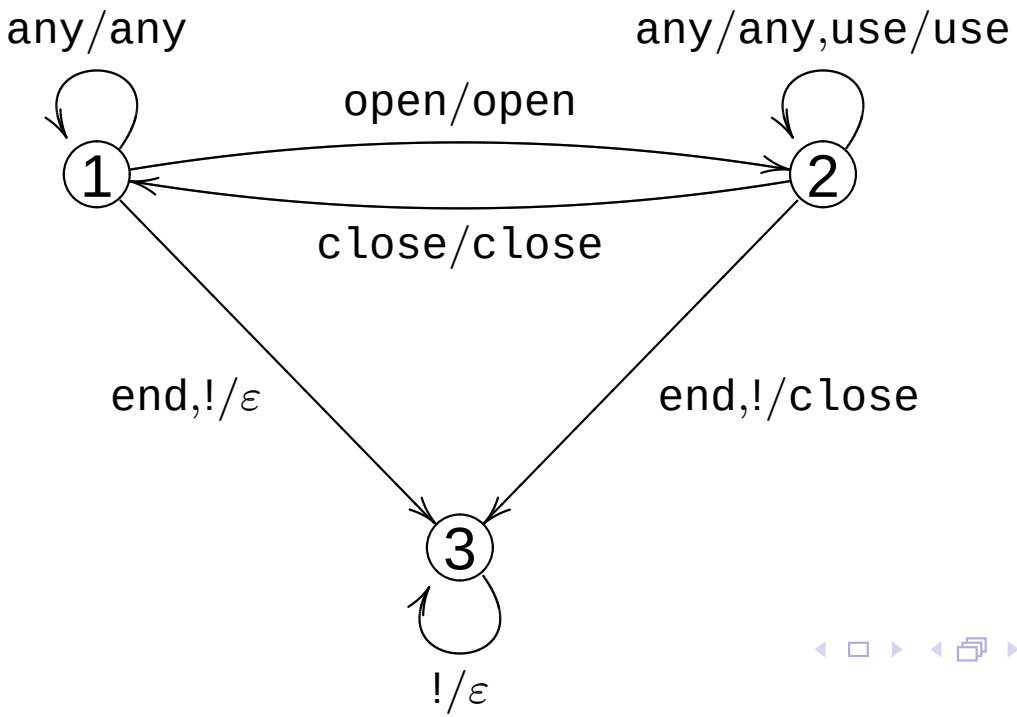
Il est nécessaire d'ajouter un évènement marquant la terminaison du programme (évènement end) :

- lorsque l'exécution se termine, le message end est transmis au contrôleur ;
- le contrôleur peut alors compléter la trace pour produire en sortie une trace acceptable.

$\rightarrow$ Le contrôleur garantit les propriétés :
$(\mathtt{any} + \mathtt{open}.(\mathtt{any} + \mathtt{use})^*.\mathtt{close})^*$ (du premier problème)
$(\mathtt{any} + \mathtt{open}.(\mathtt{any} + \mathtt{use})^*.\mathtt{close})^\omega +$
$(\mathtt{any} + \mathtt{open}.(\mathtt{any} + \mathtt{use})^*.\mathtt{close})^*.\mathtt{open}.(\mathtt{any} + \mathtt{use})^\omega$ (sûreté infinitaire)

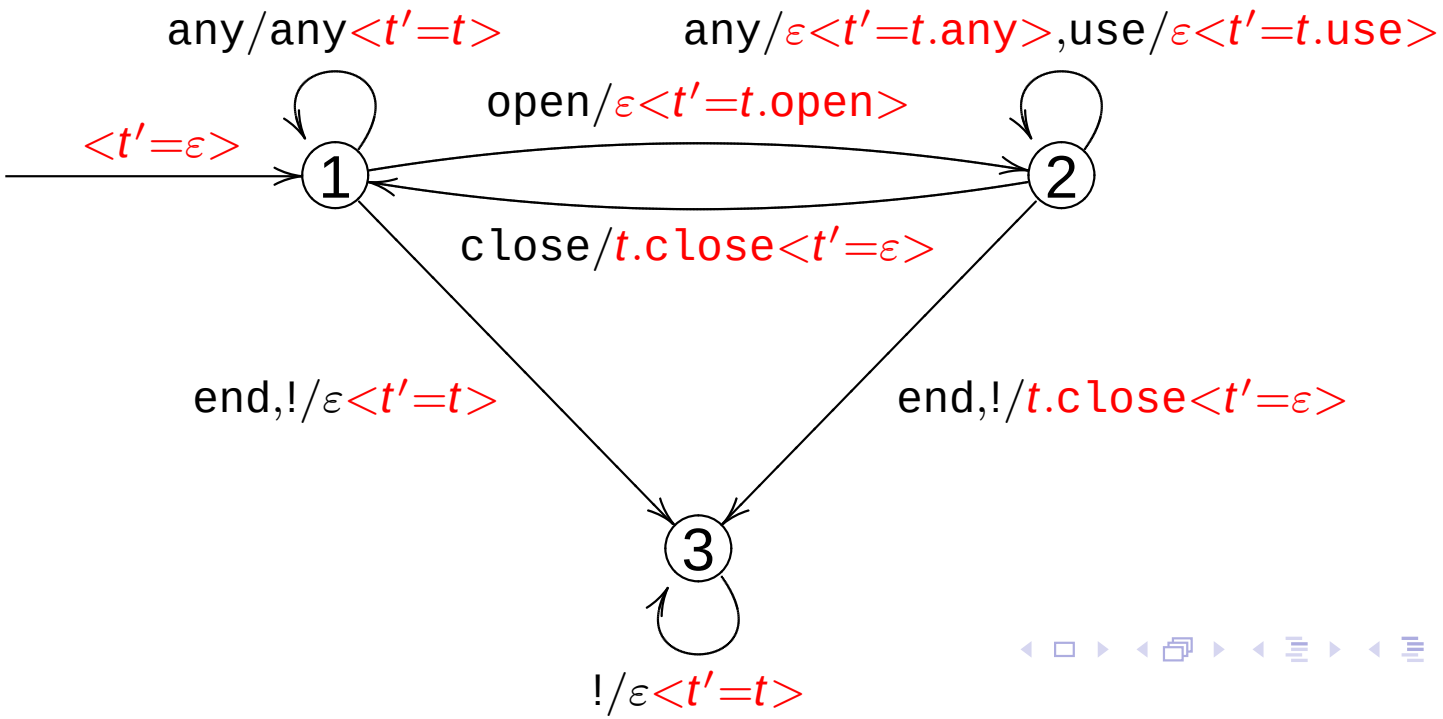
Résolution du second problème

Si l'exécution ne se termine pas, alors le service est toujours fermé après ouverture : $(\mathtt{any} + \mathtt{open}.(\mathtt{any} + \mathtt{use})^*.\mathtt{close})^\omega$



Résolution du second problème

Si l'exécution ne se termine pas, alors le service est toujours fermé après ouverture : $(\mathtt{any} + \mathtt{open}.(\mathtt{any} + \mathtt{use})^*.\mathtt{close})^\omega$



Résolution du second problème

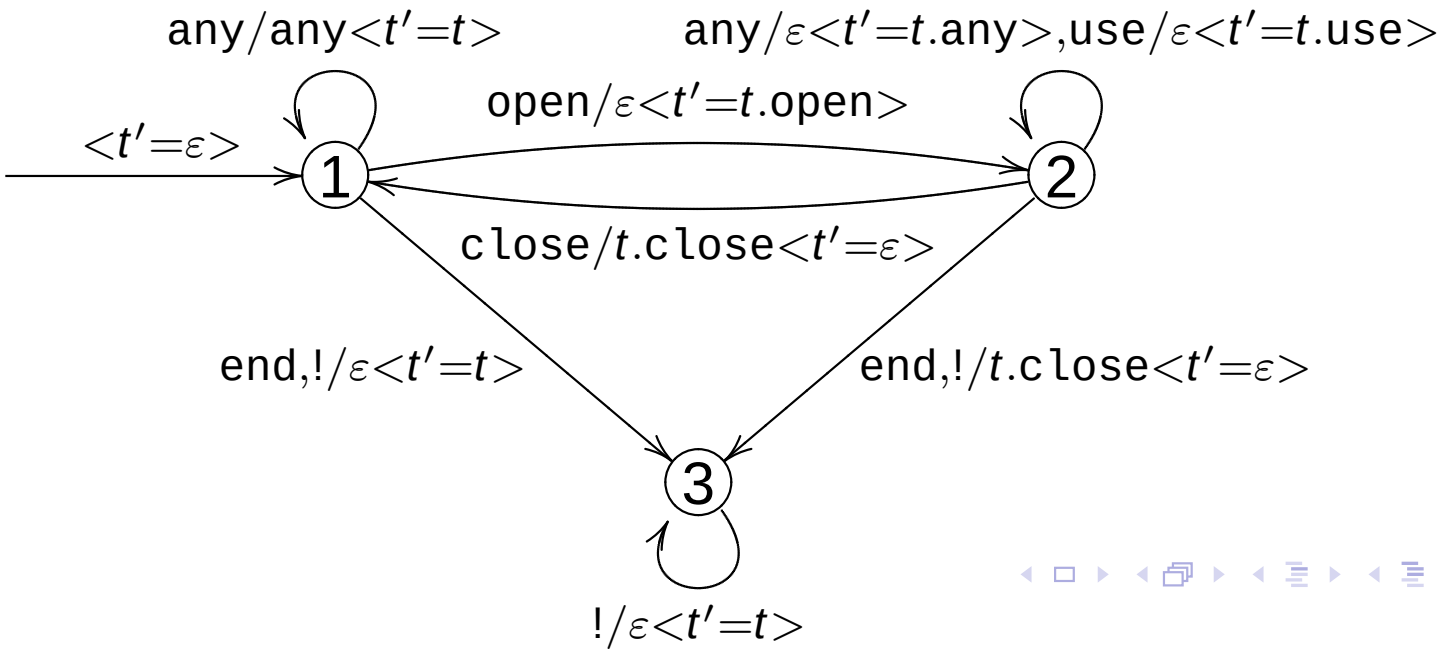
Il est nécessaire d'ajouter un mécanisme de transaction (représenté par les affectations de la variable t) :

- la transaction t est initialement vide ;
- à chaque demande d'ouverture du service, une nouvelle transaction commence ;
- elle se termine à la demande de fermeture du service ou en cas d'arrêt du programme : elle est alors exécutée puis réinitialisée.

$\rightarrow$ Le contrôleur garantit les propriétés :
$(\mathtt{any} + \mathtt{open}.(\mathtt{any} + \mathtt{use})^*.\mathtt{close})^*$ (du premier problème)
$(\mathtt{any} + \mathtt{open}.(\mathtt{any} + \mathtt{use})^*.\mathtt{close})^\omega$ (du second problème)

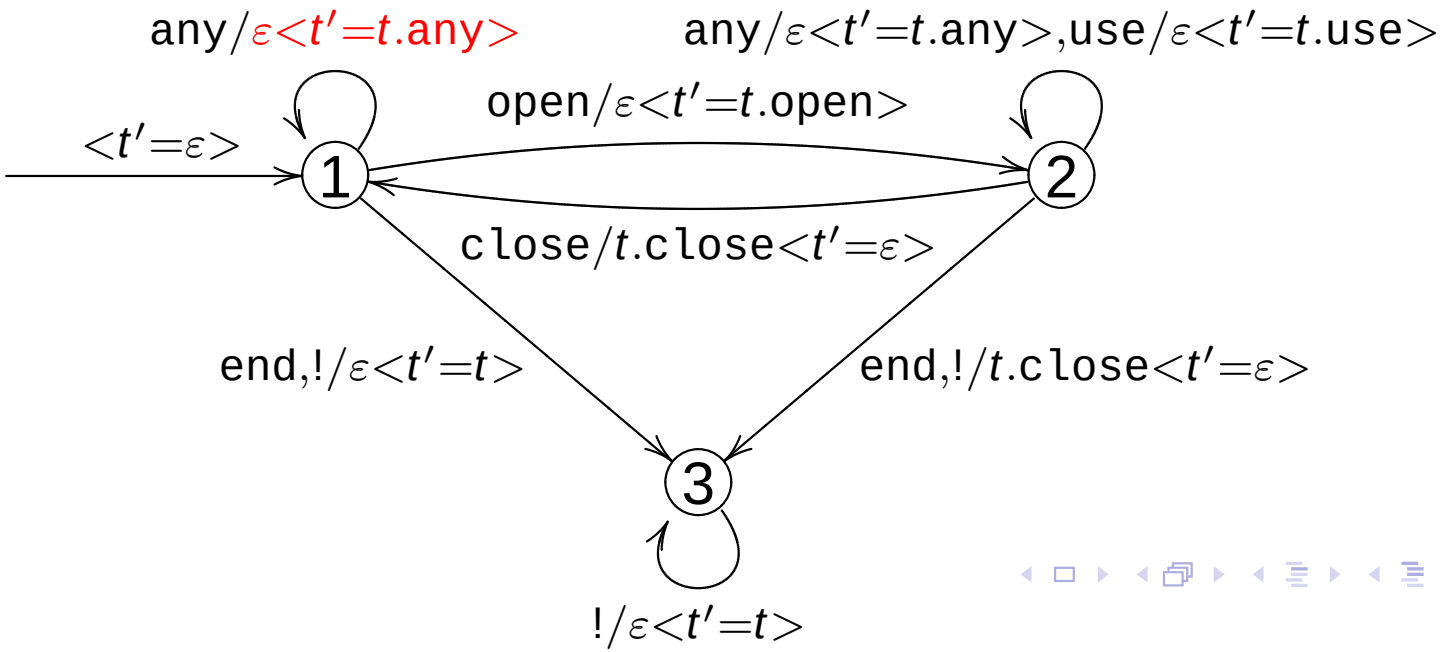
Résolution du troisième problème

Si l'exécution ne se termine pas, alors le service est toujours fermé après ouverture et le service est ouvert une infinité de fois : $(\mathrm{any}^*.\mathrm{open}.(\mathrm{any} + \mathrm{use})^*.\mathtt{close})^\omega$



Résolution du troisième problème

Si l'exécution ne se termine pas, alors le service est toujours fermé après ouverture et le service est ouvert une infinité de fois : $(\mathrm{any}^*.\mathrm{open}.(\mathrm{any} + \mathrm{use})^*.\mathtt{close})^\omega$



Résolution du troisième problème

Le contrôleur garantit les propriétés :
$(\mathtt{any}^*.\mathtt{open}.(\mathtt{any} + \mathtt{use})^*.\mathtt{close})^*$ (pas une sûreté finitaire)
$(\mathtt{any}^*.\mathtt{open}.(\mathtt{any} + \mathtt{use})^*.\mathtt{close})^\omega$ (du troisième problème)
Mais il ne garantit pas la propriété
$((\mathtt{open}.(\mathtt{any} + \mathtt{use})^*.\mathtt{close})^*.\mathtt{any})^*$ (du premier problème)
Par exemple :

$$\begin{aligned} \mathtt{any}^n &\longrightarrow \varepsilon \\ \mathtt{open.use.close.any}^n &\longrightarrow \mathtt{open.use.close} \end{aligned}$$

Plan

Formalisation et application

- Vers une caractérisation formelle des propriétés garanties par contrôleurs
- Synthèse de contrôleurs de sécurité à partir de propriétés
- Application : contrôle d'une bibliothèque logicielle (par exemple, d'entrées-sorties)
- Bibliographie :
  Schneider (2000), "Enforceable security policies"
  Bauer, Ligatti, Walker (2005), "Edit automata"

Plan

Tension entre spécification, composition et raffinement

Point de départ : un composant décrit par un système de transitions
$\rightarrow$ des états, des transitions (étiquetées) entre états
Spécification d'un comportement : une formule d'une logique temporelle

- Premier problème : la composition
  nouveau composant obtenu par composition
  $\rightarrow$ comment les spécifications logiques se composent-elles ? Mal.

Tension entre spécification, composition et raffinement

Point de départ : un composant décrit par un système de transitions
$\rightarrow$ des états, des transitions (étiquetées) entre états
Spécification d'un comportement : une formule d'une logique temporelle

- Second problème : le raffinement
  nouveau composant obtenu par raffinement (des états, des transitions, etc.)
  $\rightarrow$ comment les spécifications logiques se composent-elles ? Mal.

Tension entre spécification, composition et raffinement

Point de départ : un composant décrit par un système de transitions

$\rightarrow$ des états, des transitions (étiquetées) entre états

Spécification d'un comportement : une formule d'une logique temporelle

- Pourquoi ? Le nouveau composant ne révèle pas sa structure.

  $\rightarrow$ La décomposition doit être spécifiée logiquement.

Une solution : la logique modale

- On cherche à faciliter la composition des sous-spécifications logiques.
  $\longrightarrow$ une modalité par sous-composant
  $\longrightarrow$ le nouveau composant révèle sa structure (sous-composants et liaisons)
- On peut utiliser la technique développée pour la logique déontique.
- Bibliographie : Ulrich Endriss (2003), "Modal Logics of Ordered Trees"
  (suitable to model complex systems evolving over time in a modular fashion – extends temporal logic by a second dimension to "zoom" into states and to refine the specification of associated events)