ÉCOLE
SUPÉRIEURE
D'ÉLECTRICITÉ

Techniques de détection d'erreur avec phase d'apprentissage appliquées à la détection d'intrusion

Eric Totel

2014

Plan

- Apprentissage et vérification d'invariants
  1. **Application centralisée**
  2. Application distribuée

Application web centralisée: Contexte

- Projet ANR DALI (Design and Assessment of application Level Intrusion detection systems)

- Détection d'attaques contre les applications web
- Approche par invariants dynamiques

- Applications web (Insecure développée par Kereval)
- E-commerce, e-bank,…
- Langages interprétés (Ruby/Ruby on Rails)

Attaques

- Nombreuses attaques contre les données
  - Injections SQL
  - Modification de données en requêtes
  - Modification de variables de session/cookies

ÉCOLE SUPÉRIEURE D'ÉLECTRICITÉ

Comportement normal

```
1 class UsersController < ApplicationController
…
16 def login
17   if request.post?
18     # whole user is stored in the session
19     if session[:user] = User.authenticate(params[:user][:login], params[:user][:password])
20       flash[:notice]  = 'You have been successfully logged in.'
21       if session[:user].admin
22         redirect_to :controller => '/admin/home', :action => 'index'
23       else
24         redirect_to :controller => '/user/home',  :action => 'index'
25       end
26     else
27       flash.now[:error] = 'Login failed'
28     end
29   end
```

Ligne 21 :
21_session[:user].password == 17_params[:user][:password]

ÉCOLE SUPÉRIEURE D'ÉLECTRICITÉ

Comportement en présence d'attaques

```
1 class UsersController < ApplicationController
…
16 def login
17   if request.post?
18     # whole user is stored in the session
19     if session[:user] = User.authenticate(params[:user][:login], params[:user][:password])
20       flash[:notice]  = 'You have been successfully logged in.'
21       if session[:user].admin
22         redirect_to :controller => '/admin/home', :action => 'index'
23       else
24         redirect_to :controller => '/user/home',  :action => 'index'
25       end
26     else
27       flash.now[:error] = 'Login failed'
28     end
29   end
```

Ligne 21 :
21_session[:user].password ≠ 17_params[:user][:password]

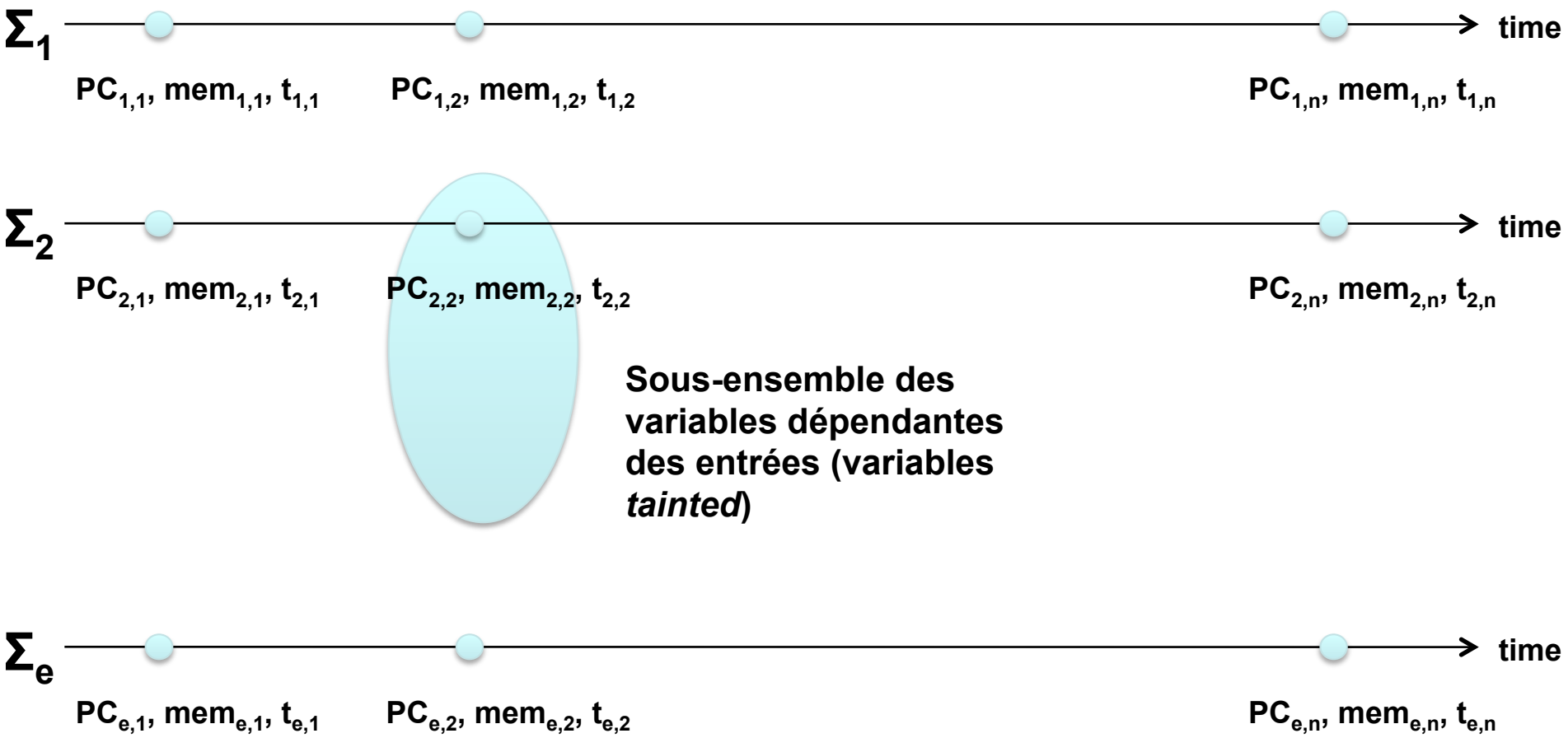
Construction du comportement normal

- Questions:
  - Sur quelles variables est-il intéressant de calculer des contraintes ?
  - Quelles contraintes chercher ?
  - Comment obtenir ces contraintes ?

ÉCOLE SUPÉRIEURE D'ÉLECTRICITÉ

Supélec

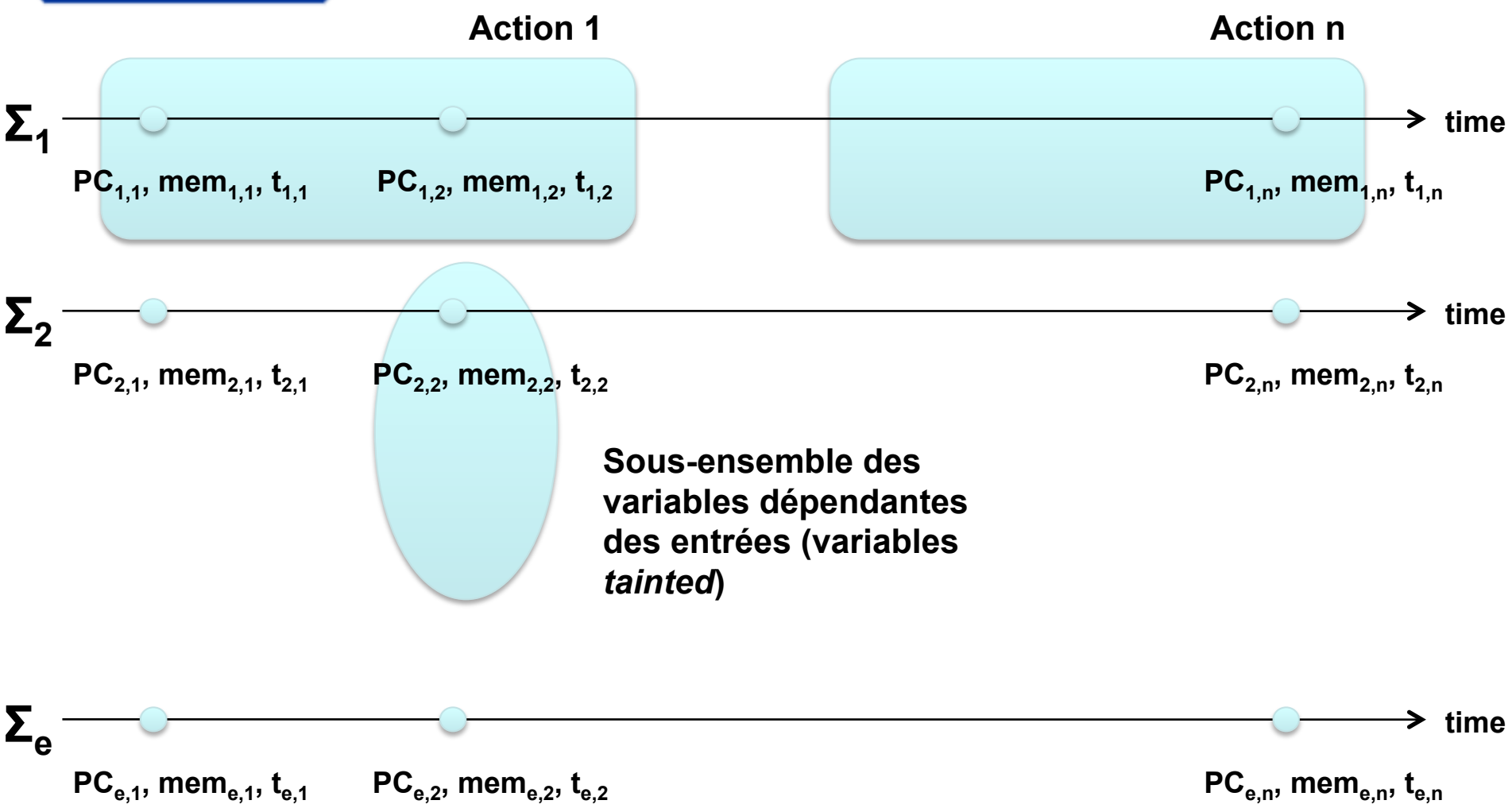
Ensemble de variables observées



ÉCOLE SUPÉRIEURE D'ÉLECTRICITÉ

Supélec

Ensemble de variables observées

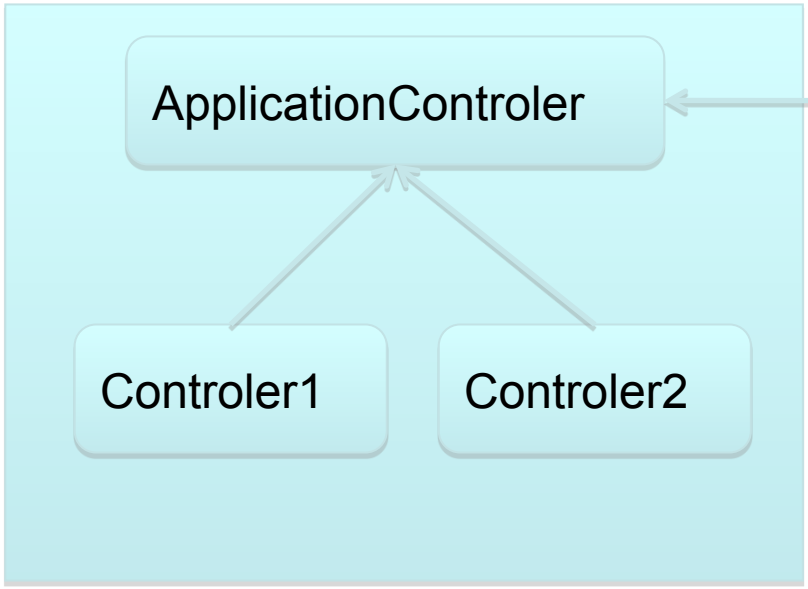


Framework Ruby on Rails

- Framework de conception d'applications web
  - Modèle-Vue-Contrôleur
- Un contrôleur est composé d'actions
- Chaque requête utilisateur passe par une action
- Observation des variables critiques dans chaque action de chaque contrôleur

ÉCOLE SUPÉRIEURE D'ÉLECTRICITÉ

Capacités d'introspection



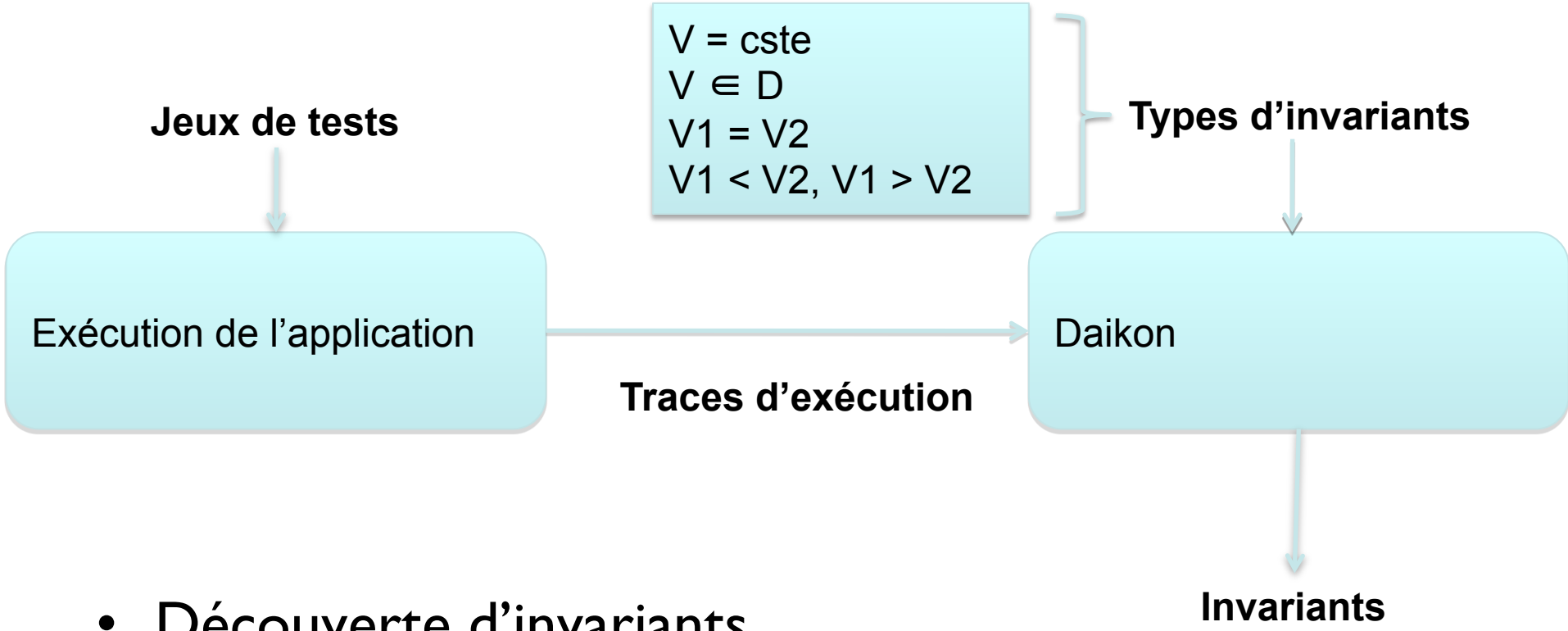
Plugin Ruby on Rails

1 Activation des mécanismes d'observation de l'exécution

2 Traçage des variables à chaque instruction exécutée dans le contrôleur

ÉCOLE SUPÉRIEURE D'ÉLECTRICITÉ

Apprentissage du comportement normal



- Découverte d'invariants
  - Utilisation de Daikon (algo generate and check)
  - Découverte d'invariants dynamiques

Génération des cas de tests

- Phase d'apprentissage
- Utilisation d'un outil de test
- Selenium
  - Automatise le test d'applications WEB
  - Initialement prévu pour faire du test fonctionnel
  - Ici, utilisé pour réaliser l'apprentissage des invariants

ÉCOLE SUPÉRIEURE D'ÉLECTRICITÉ

Supélec

Génération des invariants

Fichier de traces → **Daikon** → **Invariants**

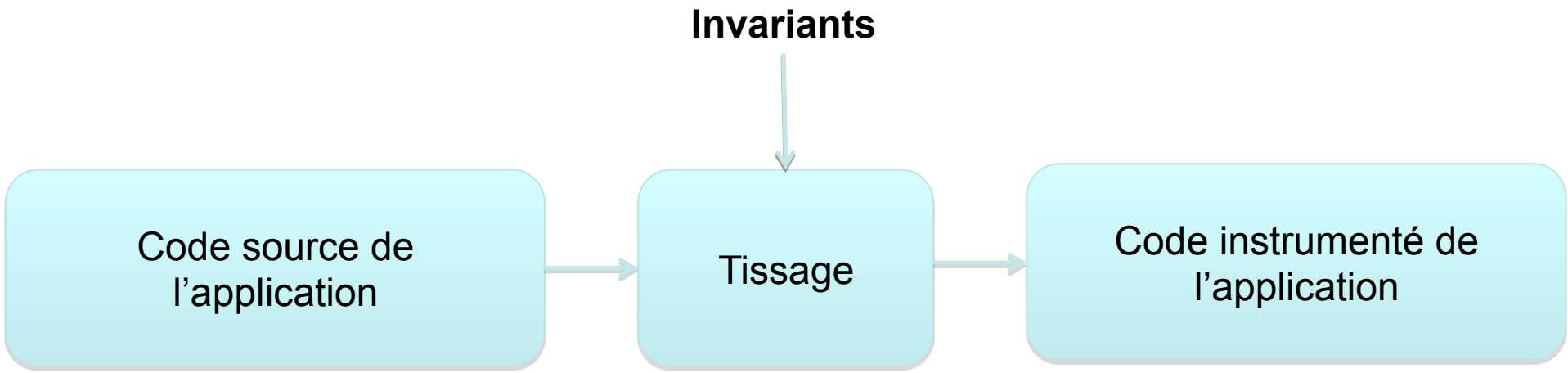
```
1.  ppt users.login:::POINT
2.  ppt-type point
3.      variable 17_p_user.password
4.          var-kind variable
5.          dec-type String
6.          rep-type java.lang.String
7.      variable 21_s_user.password
8.              var-kind variable
9.              dec-type String
10.             rep-type java.lang.String
11.
12. users.login:::POINT
13.     17_p_user.password
14.     "my_pass"
15.     21_s_user.password
16.     "my_pass"
```

```
1.  users.login:::POINT
2.  17_p_user.password == 21_s_user.password
```

Compilateur d' invariants

Invariants à tisser

Tissage d'invariants



Tissage et performances

Insecure	Lignes	Code instrumenté	Nombre d'invariants
Home_controller.rb	11	537	239
Products_controller.rb	26	1503	675
Reviews_controller.rb	37	1547	691
Users_controller.rb	51	1681	771

- Temps moyen d'exécution d'une page
  - Avant l'instrumentation: 16.5 ms
  - Après l'instrumentation: 125 ms (x8)

Faux positifs, faux négatifs

Trafic normal

Nombre de requêtes	Nombre d'alarmes	Nombre d'invariants violés
1623	20	4

Attaques automatisées

Nombre de requêtes générées	Nombre d'attaques réussies	Nombre d'alarmes générées	Faux positifs	Faux négatifs
13320	11	11	0	0

Conclusion

- Difficulté de la phase d'apprentissage
- Capacité de générer des invariants très complexes (configurable)
- Génère de très nombreux invariants (d'où l'excellente capacité de détection)
- Inconvénient: beaucoup d'invariants => perte de performance …

Plan

- Apprentissage et vérification d'invariants
  1. Application centralisée
  2. **Application distribuée**

Apprentissage du comportement d'une application distribuée

1. Contexte général d'une application distribuée
2. Quelles attaques ?
3. Modélisation d'une application distribuée par apprentissage
    1. Traces locales et traces globales
    2. Inférence d'un automate à états finis
    3. Invariants temporels

Application distribuée

- Un nombre défini de nœuds qui interagissent
- Horloges pas forcément bien synchronisées
- Ordre global des actions sur les nœuds difficile à obtenir

- Etat de l'art en détection d'intrusion
  - Hypothèse: horloges synchronisées
  - Corrélation centralisée d'alertes produites localement

Attaques

- Confidentialité
  - Interception des messages (prévention par Cryptographie)
- Intégrité
  - Modification d'un message (prévention par crypto)
  - Rejouer un message
  - Destruction d'un message
  - Forger un message à destination d'un des processus de l'application distribuée, indépendamment de son contexte global

Détection locale vs détection globale

- Application distribuée
  - Logique globale
  - Chaque nœud responsable d'une partie de l'application
    - Problème de cohérence entre les nœuds

- Détection locale insuffisante pour détecter la compromission d'un nœud par rapport à l'état global

Traces d'exécution

- Obtenues localement et constituées:
  - D'actions locales
    - Appels de fonctions/méthodes, appels systèmes
  - D'échanges de messages
    - Envoi de messages
    - Réception de messages
    - sur des canaux de communication entre les nœuds/ processus

ÉCOLE SUPÉRIEURE D'ÉLECTRICITÉ

Exemple de traces

Processus 1 (nœud 1)	Processus 2 (nœud 2)
a	d
c0	c1?m
c1!m	y

- Un canal de communication unidirectionnel c1 de P1vers P2
- Processus 1: Envoi de message m sur c1
- Processus 2 : Réception de message m sur c1

Exemple de traces

Processus 1 (nœud 1)	Processus 2 (nœud 2)
a	d
c0	c1?m
c1!m	y

- Propriétés des canaux
  - Pas de perte de messages
  - FIFO: les messages sont consommés dans l'ordre où ils sont émis

Exemple de traces

Processus 1 (nœud 1)	Processus 2 (nœud 2)
a	d
c0	c1?m
c1!m	y

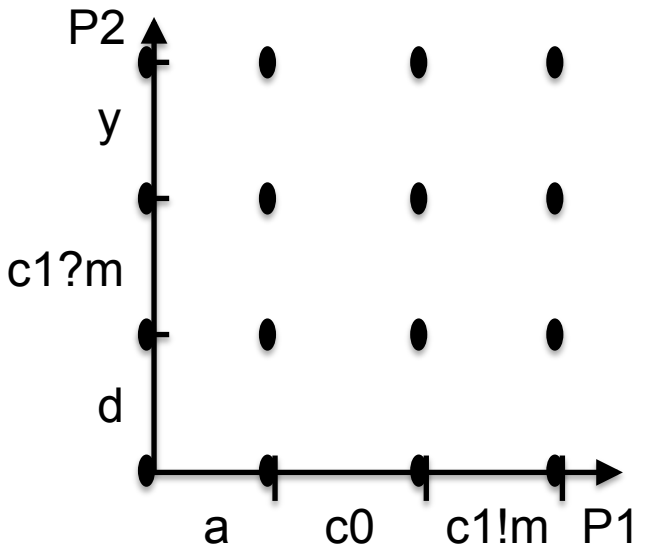
- Ordre total des événements pour chaque processus
- Pas d'horloge globale donc pas d'ordre total connu entre les processus
- Ordre sur les messages: une réception d'un message a lieu après son émission

Modélisation du comportement d'une application

- Deux types de modèles
  - Machine à états:
    - décrire les enchainements possibles d'actions (de manière globale)
  - Invariants:
    - capturer des enchainements d'actions qui restent toujours vrais
- Choix:
  - Les deux !!
  - Pourquoi … Explication dans la suite…

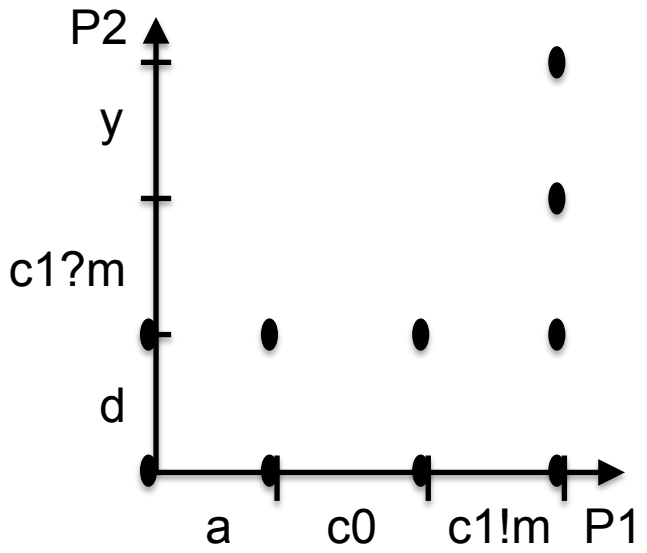
Des traces locales au comportement global

- Exemple de représentation de l'ordonnancement des événements dans les traces:



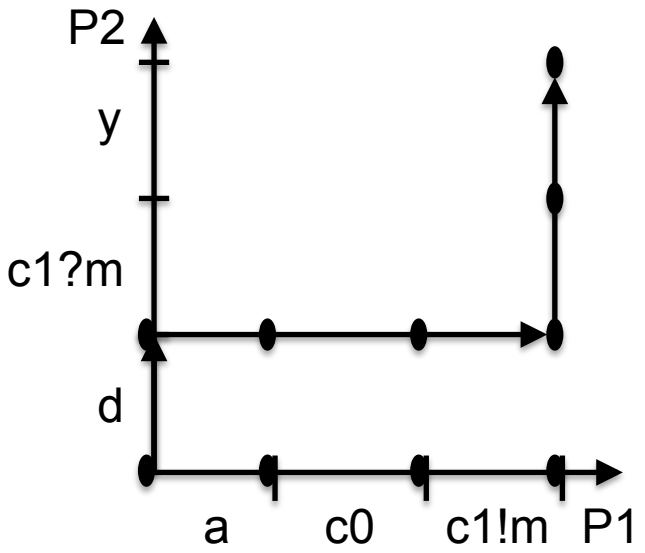
Des traces locales au comportement global

- La réception du message ne peut avoir lieu qu'après son envoi: définition d'un treillis
- Points atteignables dans la représentation:



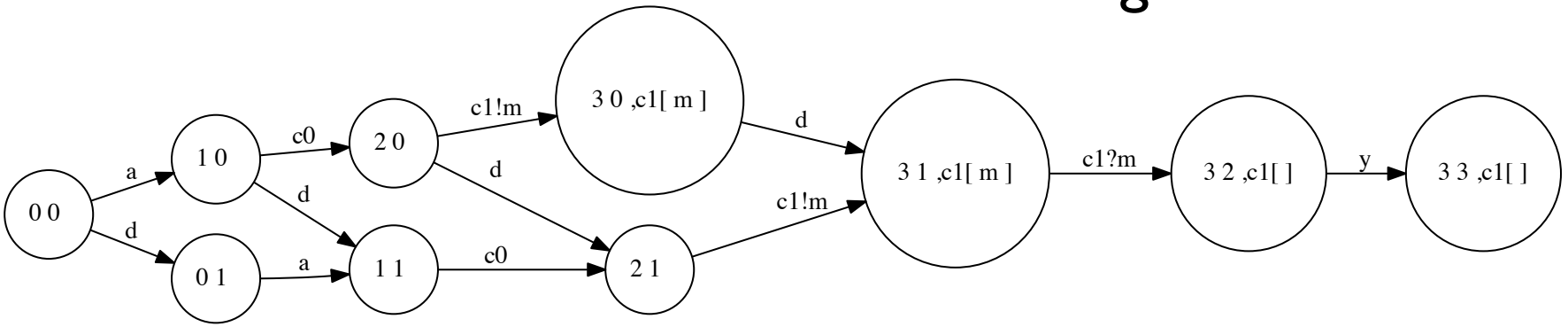
Des traces locales au comportement global

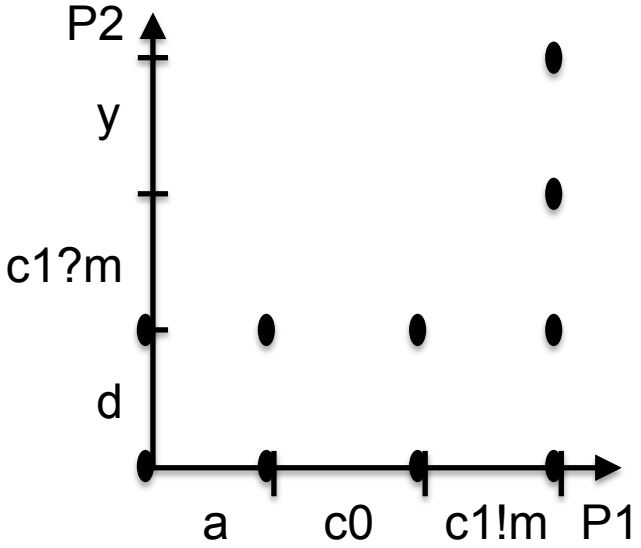
- Construction de tous les chemins possibles du départ à l'arrivée
- Exemple de chemin



Des traces locales au comportement global

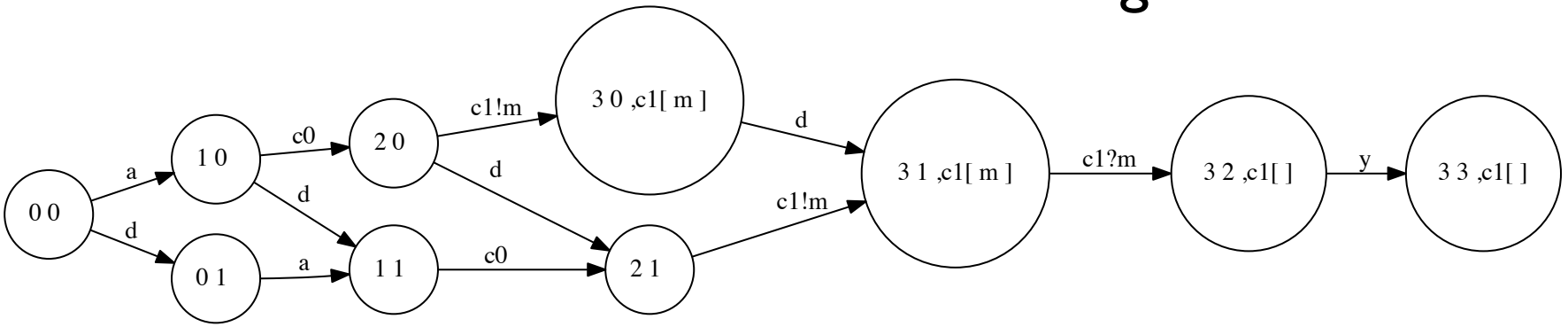
- Dérivation d'un automate à états finis global





Des traces locales aux traces globales

- Dérivation d'un automate à états finis global

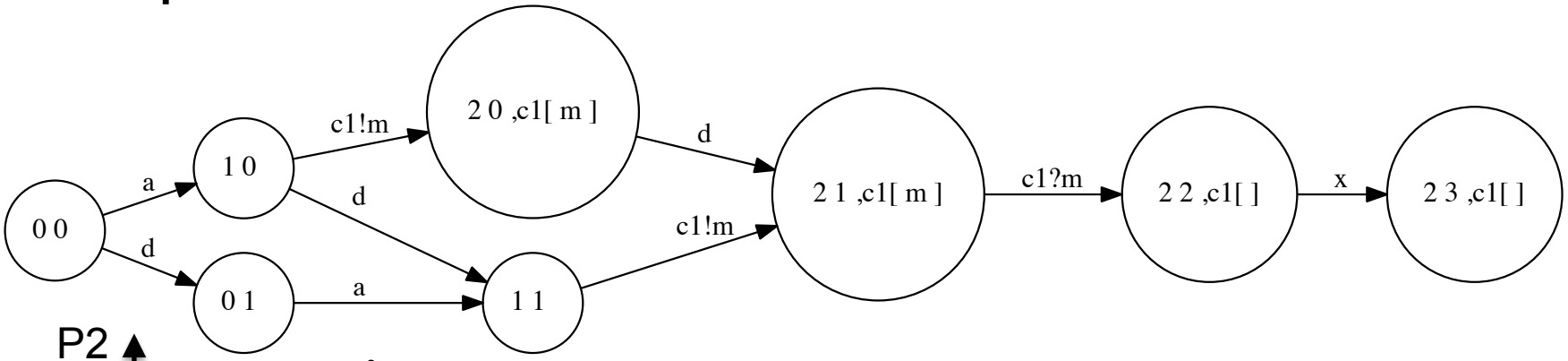


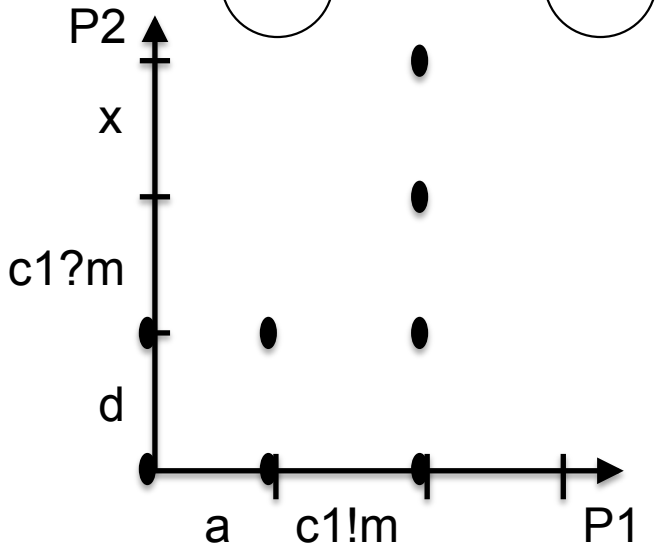
- Traces globales possibles

a	a	a	d
c0	c0	d	a
c1!m	d	c0	c0
d	c1!m	c1!m	c1!m
c1?m	c1?m	c1?m	c1?m
y	y	y	y

Apprentissage sur plusieurs traces

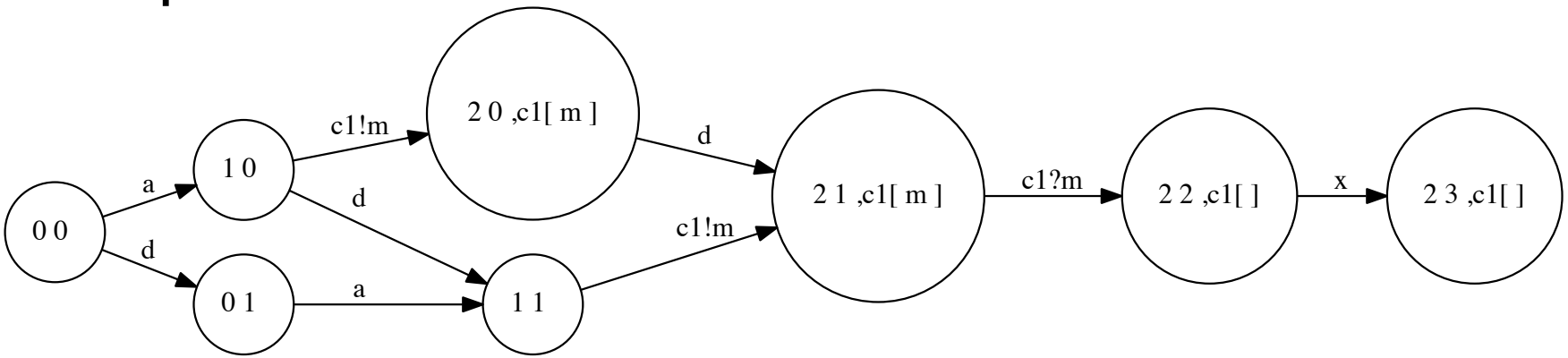
- Exploitation d'une deuxième trace





Apprentissage sur plusieurs traces

- Exploitation d'une deuxième trace

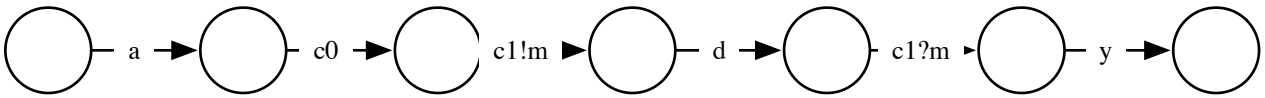


- Traces globales possibles

a	a	d
c1!m	d	a
d	c1!m	c1!m
c1?m	c1?m	c1?m
x	x	x

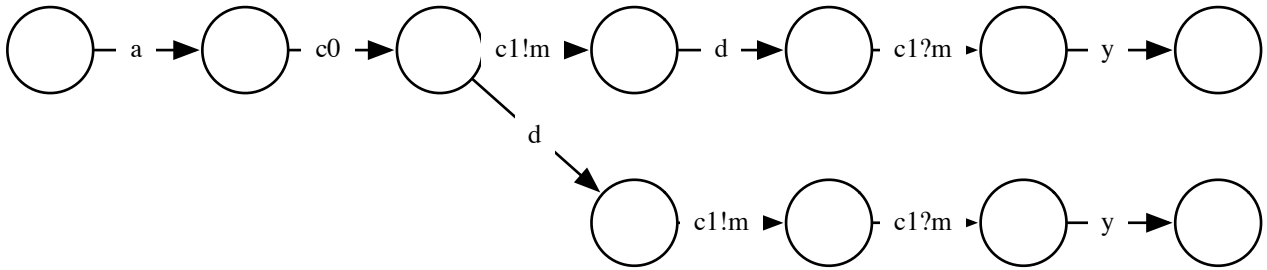
Construction de l'automate décrivant toutes les traces (PTA)

a	a	a	d	a	a	d
c0	c0	d	a	c1!m	d	a
c1!m	d	c0	c0	d	c1!m	c1!m
d	c1!m	c1!m	c1!m	c1?m	c1?m	c1?m
c1?m	c1?m	c1?m	c1?m	x	x	x
y	y	y	y			



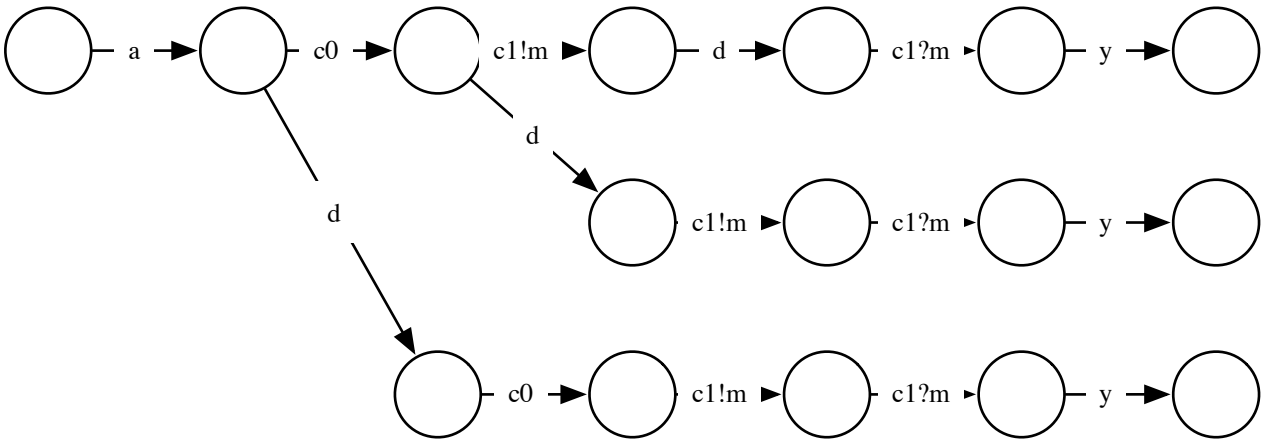
Construction de l'automate décrivant toutes les traces (PTA)

a	a	a	d	a	a	d
c0	c0	d	a	c1!m	d	a
c1!m	d	c0	c0	d	c1!m	c1!m
d	c1!m	c1!m	c1!m	c1?m	c1?m	c1?m
c1?m	c1?m	c1?m	c1?m	x	x	x
y	y	y	y			



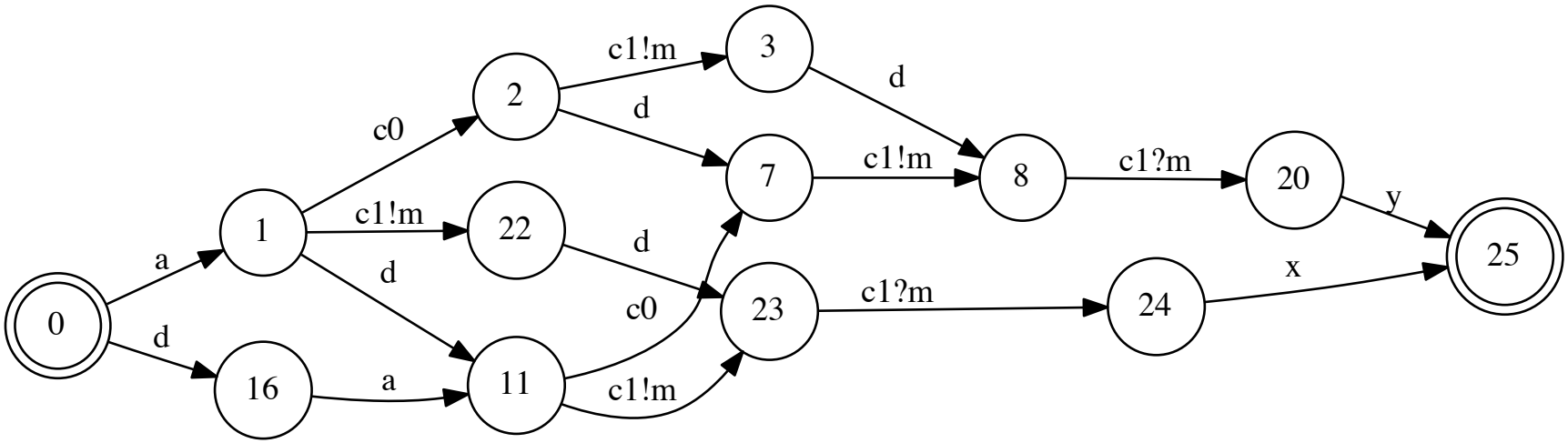
Construction de l'automate décrivant toutes les traces (PTA)

a	a	a	d	a	a	d
c0	c0	d	a	c1!m	d	a
c1!m	d	c0	c0	d	c1!m	c1!m
d	c1!m	c1!m	c1!m	c1?m	c1?m	c1?m
c1?m	c1?m	c1?m	c1?m	x	x	x
y	y	y	y			



Construction de l'automate décrivant toutes les traces (optimisé)

a c0 c1!m d c1?m y	a c0 d c1!m c1?m y	a d c0 c1!m c1?m y	d a c0 c1!m c1?m y	a c1!m d c1?m x	a d c1!m c1?m x	d a c1!m c1?m x



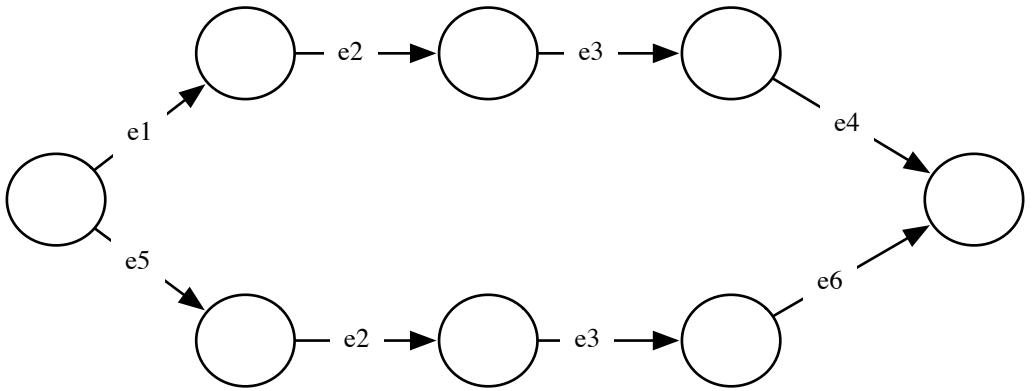
Automate satisfaisant ?

- L'automate contient potentiellement beaucoup d'états
- Donc besoin de le "compacter"

- Nombreux algorithmes possibles
  - Exemple d'algorithme: k-tail

Fonctionnement intuitif du k-tail

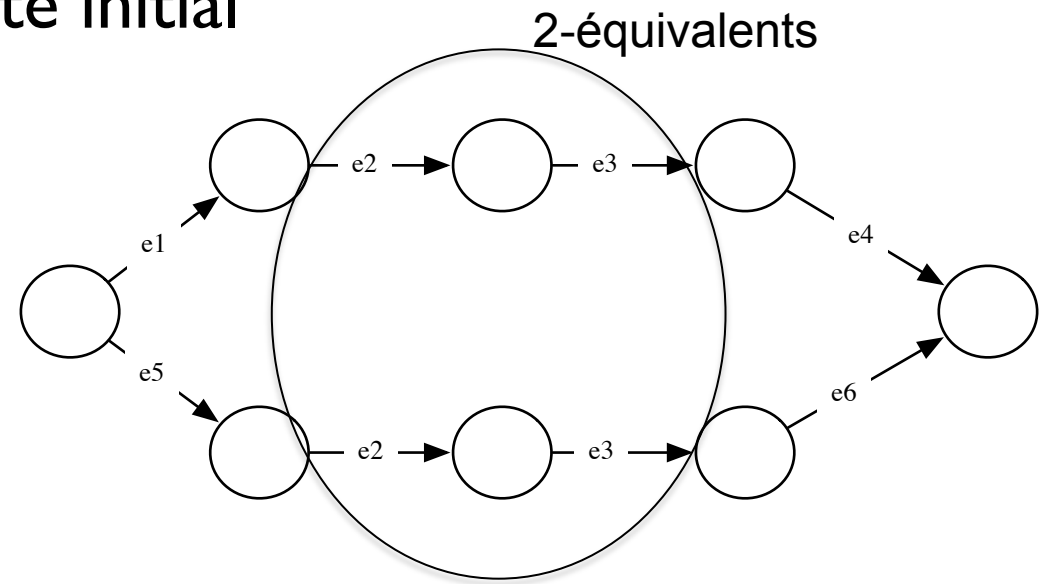
- Automate initial



- Application de 2-tail: deux états sont 2-équivalents si leur 2-future est le même

Fonctionnement intuitif du k-tail

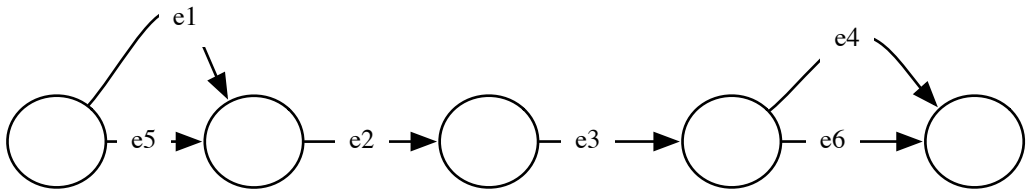
- Automate initial



- Application de 2-tail: deux états sont 2-équivalents si leur 2-future est le même

Fonctionnement intuitif du k-tail

- Automate compacté

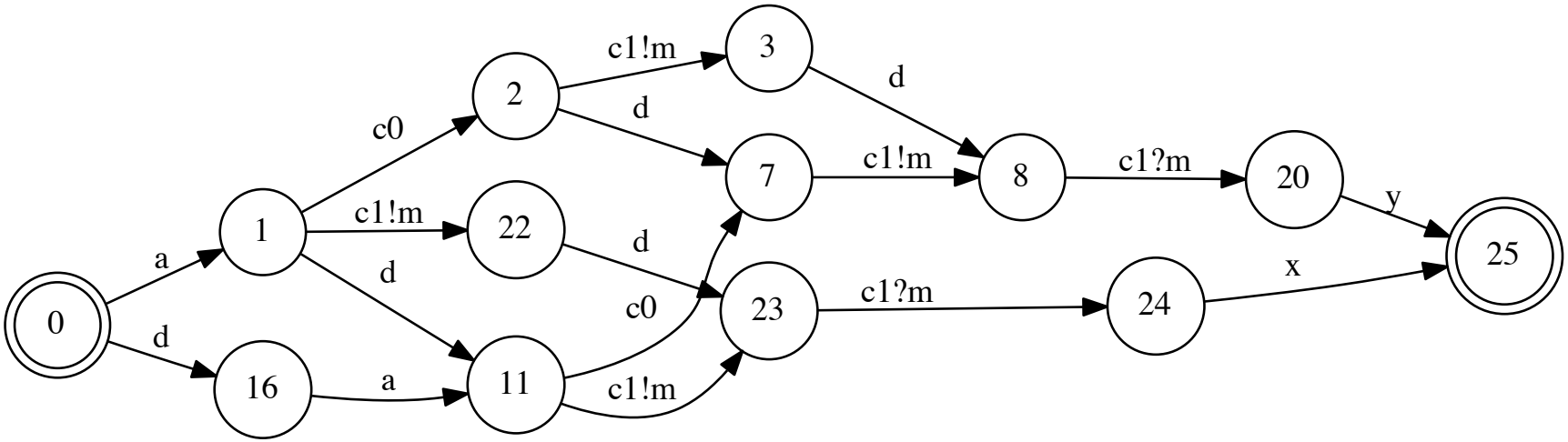


- Souci:
  - La séquence e1, e2, e3, e6 est acceptée par l'automate compacté, pas par l'automate initial

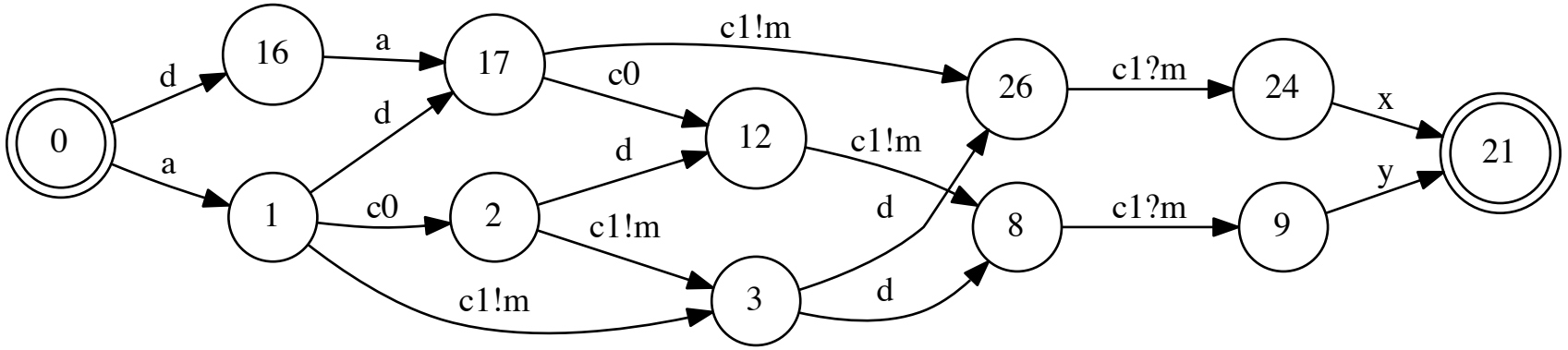
Analyse du résultat

- Ces algorithmes sur-généralisent le comportement:
  - Tous les comportements appris sont inclus dans l'automate final
  - L'automate final peut introduire des comportements qui n'appartiennent pas à l'ensemble des comportements observés
- Risque d'introduction de faux négatifs
  - Besoin de conserver les propriétés invariantes vérifiées par l'automate initial

2-tail sur l'exemple (utilisation de k-behaviour)



- Après compactage (ici un seul état de moins)

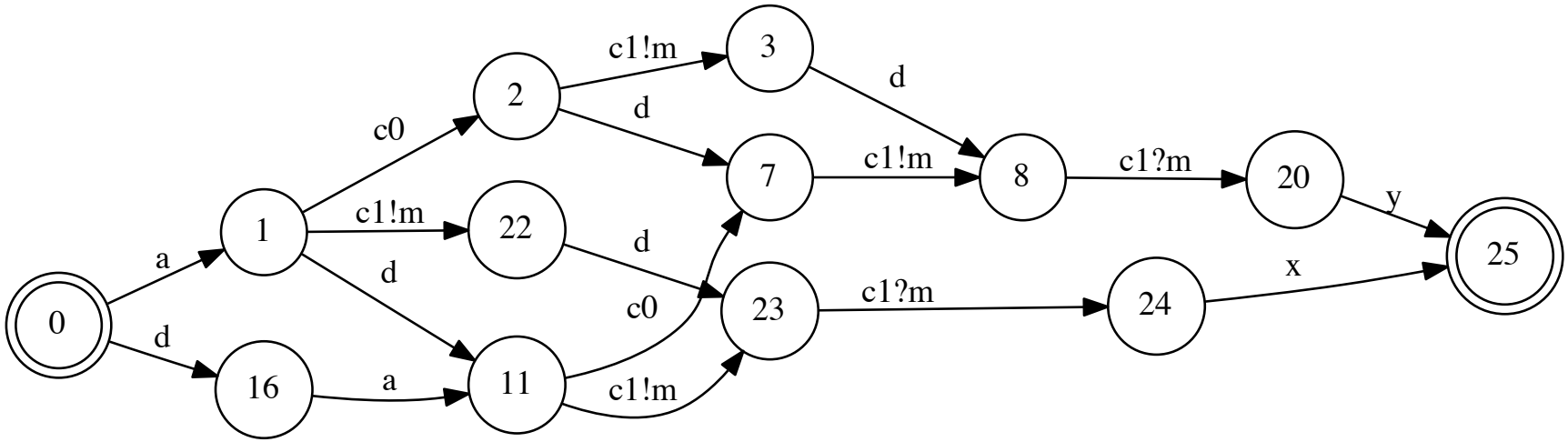


Introduction des invariants temporels

- Généralement, on considère trois types d'invariants
  - A est toujours suivi de B (A AlwaysFollowedBy B)
  - A n'est jamais suivi de B (A NeverFollowedBy B)
  - A précède toujours B (A AlwaysPrecedes B)

- On va calculer tous les invariants possibles sur le comportement global appris

Utilisation de synoptic pour le calcul des invariants



a AlwaysFollowedBy(t) c1!m
a AlwaysFollowedBy(t) c1?m
a AlwaysPrecedes(t) c0
a AlwaysPrecedes(t) c1!m
a AlwaysPrecedes(t) c1?m
a AlwaysPrecedes(t) x
a AlwaysPrecedes(t) y
a NeverFollowedBy(t) a
…

c0 AlwaysFollowedBy(t) c1!m
c0 AlwaysFollowedBy(t) c1?m
c0 AlwaysFollowedBy(t) y
c0 AlwaysPrecedes(t) y
c0 NeverFollowedBy(t) a
c0 NeverFollowedBy(t) c0
c0 NeverFollowedBy(t) x

…

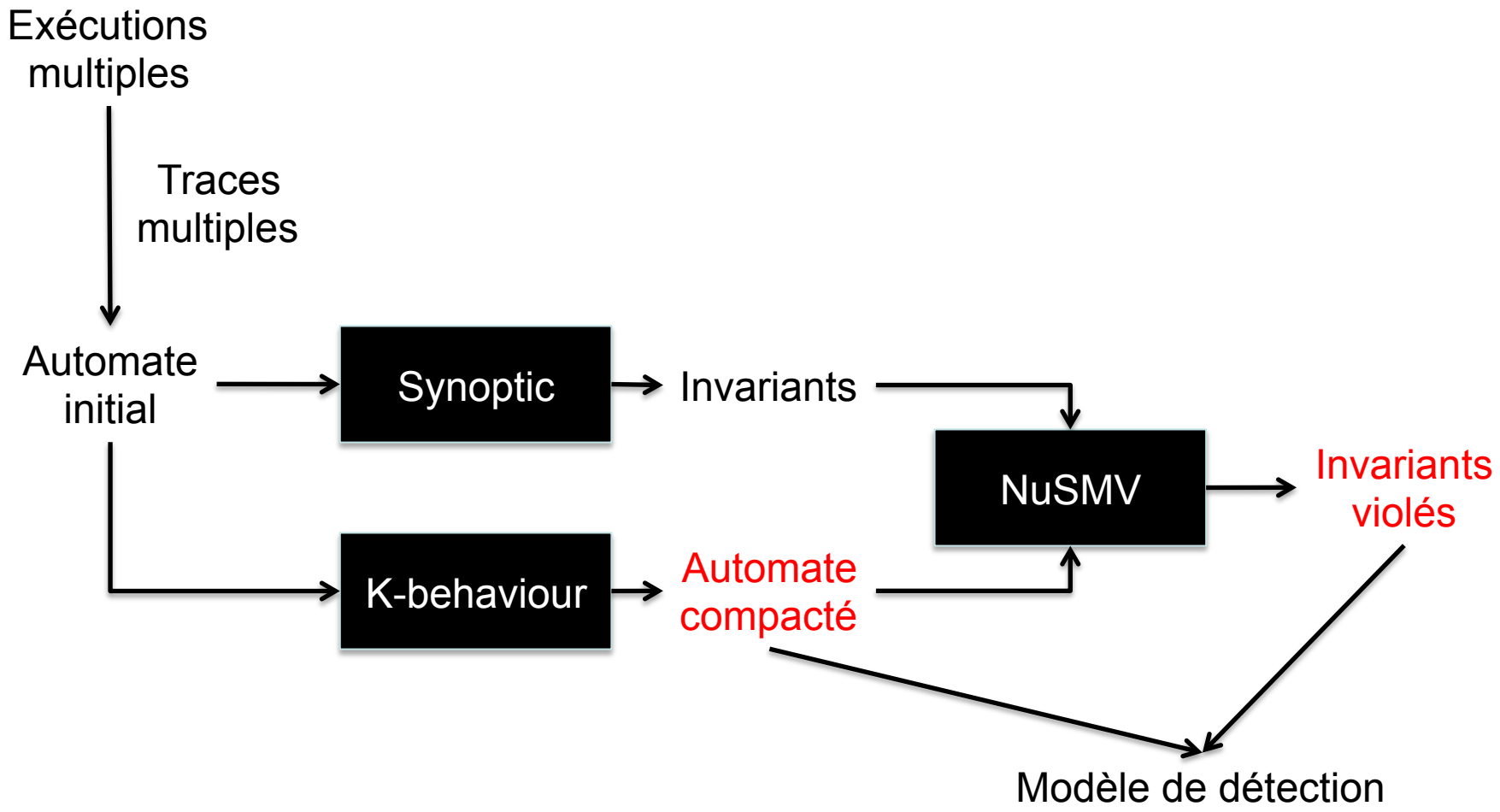
Invariants et compactage

- Question: après compactage de l'automate, les invariants sont-ils toujours vérifiés ?

- Réponse apportée par le monde du "model-checking"
  - Étant donné un modèle (automate)
  - Des propriétés (invariants) sont-elles vérifiées par l'automate ?

ÉCOLE SUPÉRIEURE D'ÉLECTRICITÉ

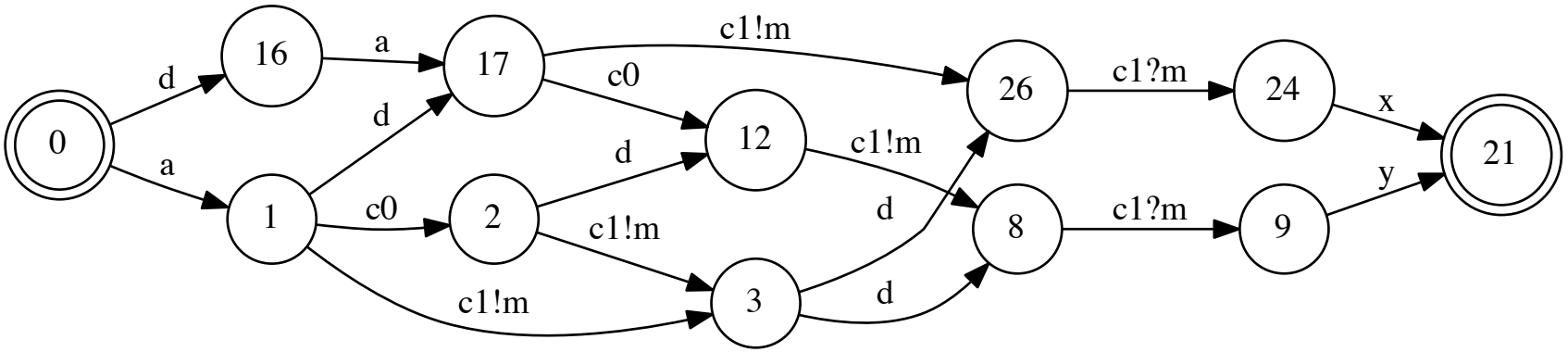
Schéma global



Model checker: NuSMV

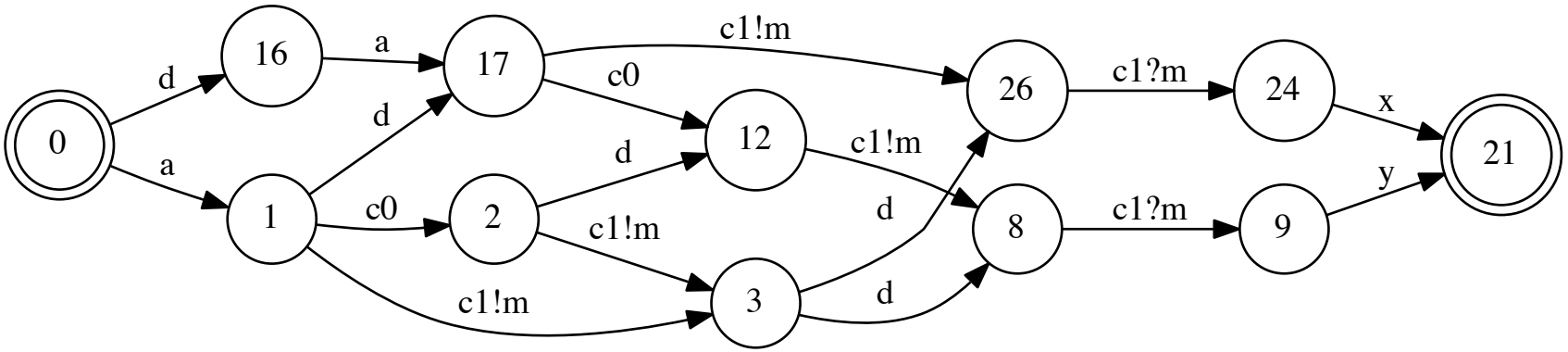
- Modélisation de l'automate compact:
  - Évolution de variables représentant les états et les transitions
- Modélisation des invariants (LTL: Logique Temporelle Linéaire) de l'automate avant compactage

Modélisation de l'automate: enchainement des transitions




```
INIT
        trans=INITIAL;
ASSIGN
        next(trans):=case
                trans = INITIAL : {d,a};
                state = 16 & trans = a : {c1_s_m,c0};
                state = 12 & trans = c1_s_m : {c1_r_m};
                state = 0 & trans = d : {a};
                state = 0 & trans = a : {c1_s_m,d,c0};
                …
        esac;
```

Modélisation de l'automate: enchainement des états




```
INIT
        state=0;
ASSIGN
        next(state):=case
                state = 16 & trans = a : {17};
                state = 12 & trans = c1_s_m : {8};
                state = 0 & trans = d : {16};
                state = 0 & trans = a : {1};
                state = 8 & trans = c1_r_m : {9};
                ...
        esac;
```

Modélisation des propriétés: invariants temporels

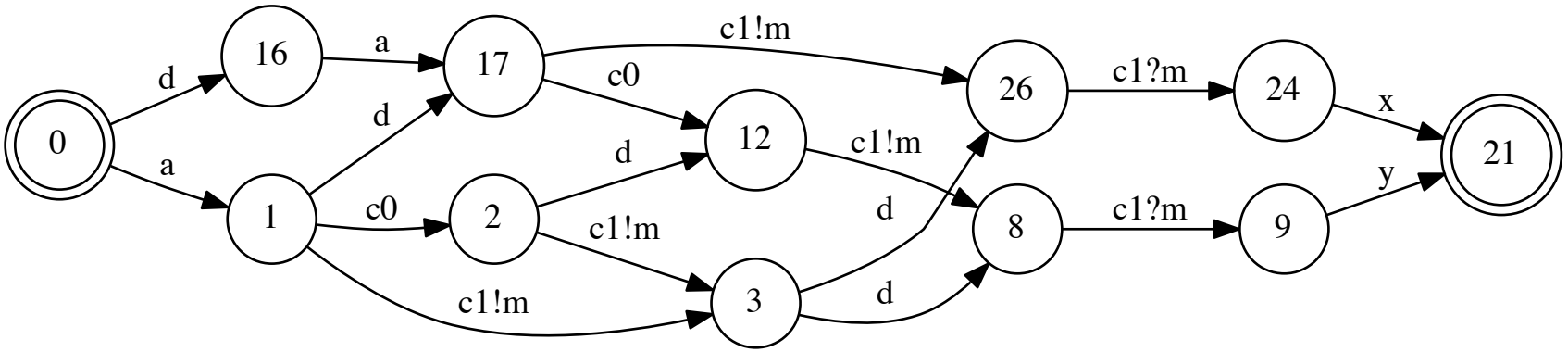
- Trois types d'invariants à modéliser en LTL
  - A AlwaysFollowedBy B
  - A NeverFollowedBy B
  - A AlwaysPrecedes B

- Sur l'exemple:

```
a AlwaysFollowedBy(t) c1!m  -- specification ( F trans = END ->  G (trans = a ->  F trans = c1_s_m))
a NeverFollowedBy(t) a      -- specification  G (trans = a ->  X ( G !( F trans = a)))
a AlwaysPrecedes(t) c0      -- specification ( F trans = END -> ( F trans = c0 -> (!(trans = c0) U trans = a)))
```

ÉCOLE SUPÉRIEURE D'ÉLECTRICITÉ

Analyse de l'automate compacté



- Résultat du model-checker

c0 AlwaysFollowedBy(t) y is false
c0 AlwaysPrecedes(t) y is false
c0 NeverFollowedBy(t) x is false

Modèle du comportement de l'application distribuée

- Un automate à état fini représentant le fonctionnement global de l'application
- Un ensemble d'invariants permettant d'augmenter la précision de la détection, en éliminant de potentiels faux négatifs
- Compromis à trouver entre le nombre d'états et le nombre d'invariants à vérifier (variation du k de k-tail)
- L'IDS se sert de ces deux modèles pour effectuer la détection