Logique et Déduction - Cours 4
Fiabilité logicielle et modélisation

Pierre-Cyrille Héam

pheam [at] femto-st.fr

Licence 2 Informatique

Vol 501 Ariane 5

toto

Vol 501 Ariane 5

Le 04 Juin 1996, le premier vol du lanceur Ariane 5 explose en vol après environ 40 secondes de vol.

- Le système intertiel envoie de mauvaises données au système de pilotage automatique,

Vol 501 Ariane 5

Le 04 Juin 1996, le premier vol du lanceur Ariane 5 explose en vol après environ 40 secondes de vol.

- Le système intertiel envoie de mauvaises données au système de pilotage automatique,
- Le système intertiel de secours se met en route et réagit exactement de la même façon,

Vol 501 Ariane 5

Le 04 Juin 1996, le premier vol du lanceur Ariane 5 explose en vol après environ 40 secondes de vol.

- Le système intertiel envoie de mauvaises données au système de pilotage automatique,
- Le système intertiel de secours se met en route et réagit exactement de la même façon,
- Le pilote automatique, ayant de mauvaises données, ordonne de façon inapproprié un changement de trajectoire violent,

Vol 501 Ariane 5

Le 04 Juin 1996, le premier vol du lanceur Ariane 5 explose en vol après environ 40 secondes de vol.

- Le système intertiel envoie de mauvaises données au système de pilotage automatique,
- Le système intertiel de secours se met en route et réagit exactement de la même façon,
- Le pilote automatique, ayant de mauvaises données, ordonne de façon inapproprié un changement de trajectoire violent,
- Ce virage serré provoque un arrachage de pièce de la fusée,

Vol 501 Ariane 5

Le 04 Juin 1996, le premier vol du lanceur Ariane 5 explose en vol après environ 40 secondes de vol.

- Le système intertiel envoie de mauvaises données au système de pilotage automatique,
- Le système intertiel de secours se met en route et réagit exactement de la même façon,
- Le pilote automatique, ayant de mauvaises données, ordonne de façon inapproprié un changement de trajectoire violent,
- Ce virage serré provoque un arrachage de pièce de la fusée,
- La fusée étant alors hors de contrôle, le système d'auto-destruction s'enclenche et la fusée explose (le navigateur au sol ayant lui aussi ordonné la destruction de la fusée).

Vol 501 Ariane 5

- L'origine du problème provient du dépassement mémoire (codage sur 8 bits) d'un entier, il en aurait fallut 9.
- Ce codage vient du code Ariane 4.
- Ariane 5 est plus puissance, d'où le dépassement.

Cependant [1]

1. pour plus d'informations
http://deschamp.free.fr/exinria/divers/ariane_501.html

Vol 501 Ariane 5

- L'origine du problème provient du dépassement mémoire (codage sur 8 bits) d'un entier, il en aurait fallut 9.
- Ce codage vient du code Ariane 4.
- Ariane 5 est plus puissance, d'où le dépassement.

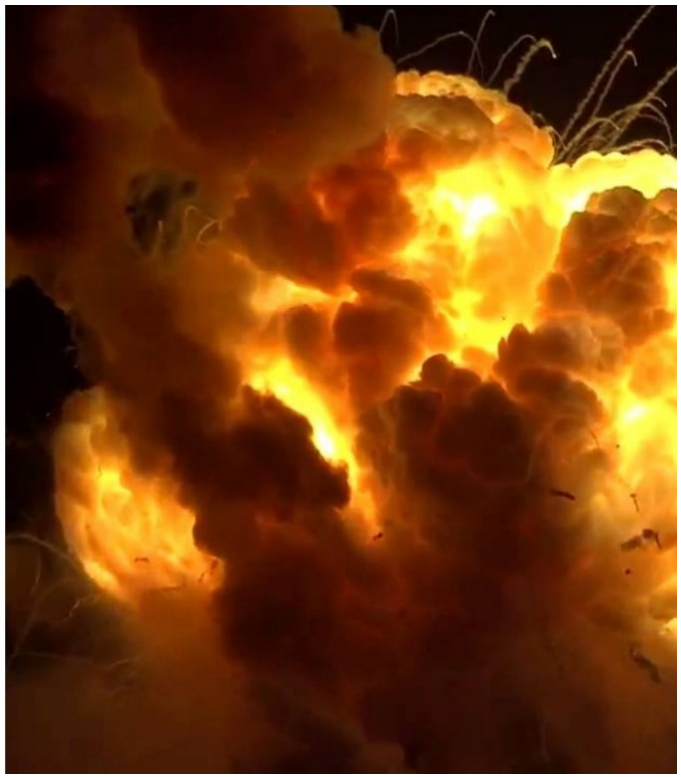
Cependant [1]

- Copier-coller depuis Ariane 4 : la réutilisation de code est fréquente.
- Afin d'économiser environ 120 000 euros, des simulations n'ont pas été effectuées. Réalisées après coups, il a été montré qu'elles auraient permis de détecter le problème.
- Cahier des charges de pannes s'appuie sur des failles probabilistes. Il n'y a qu'un système de secours. Les problèmes de conceptions ne sont pas pris en compte.

1. pour plus d'informations
http://deschamp.free.fr/exinria/divers/ariane_501.html

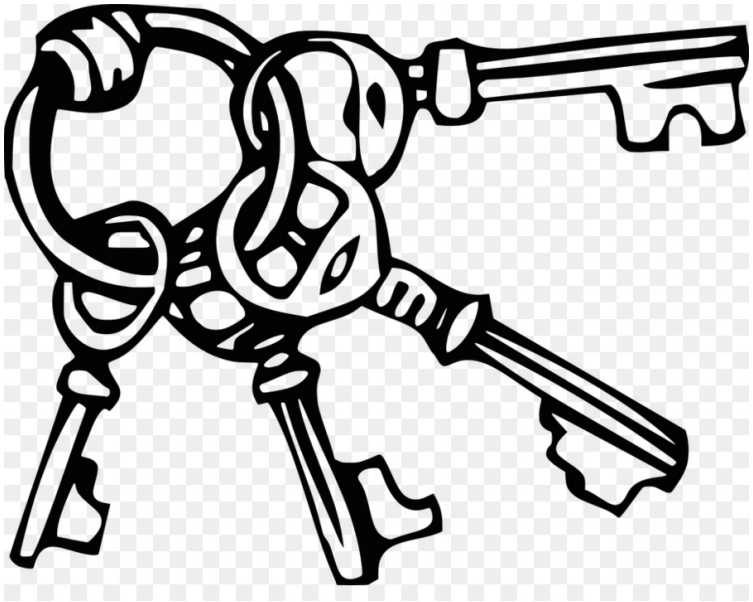
Mariner 1

Le 28 Juillet 1962 la fusée *Marnier 1* est détruite par le contrôle au sol au dessus de l'atlantique pour une mauvaise trajectoire : en cause une mauvaise transcription d'un caractère (il manquait un _).



Kerberos

Le système de gestion de clé Kerberos (système de sécurité) a une une faille de 1988 à 1996 générant des clés prédictible. En cause, l'utilisation d'une mauvaise graine pour la génération pseudo-aléatoire.



Prix négatifs

- En avril 2021, en plein confinement COVID19, la demande en pétrole s'effondre.
- Les supers-tankers, plein, ne savent pas où décharger. Les compagnie sont prête à payer pour se débarrasser de leur pétrole.
- Les prix du pétrole deviennent négatifs.

Prix négatifs

- En avril 2021, en plein confinement COVID19, la demande en pétrole s'effondre.
- Les supers-tankers, plein, ne savent pas où décharger. Les compagnie sont prête à payer pour se débarrasser de leur pétrole.
- Les prix du pétrole deviennent négatifs.
- Le logiciel d'un société de Traders Sud-coréenne a planté, ne sachant gérer des prix négatifs : impossibilité de vendre pour les traders qui a entraîné de lourdes pertes.

Problèmes d'horloge

En 2004, le système de communication aérien de Los Angeles tombe en panne totale : plus aucune communication. Pas d'accident mais une dizaine d'avions sont passé proche.

Le problème : l'horloge Windows limité à 4 294 967 295 secondes (un peu moins de 50 jours). Les ordinateurs n'avaient pas été rebooté depuis cette durée.

Problèmes d'horloge

En 2004, le système de communication aérien de Los Angeles tombe en panne totale : plus aucune communication. Pas d'accident mais une dizaine d'avions sont passé proche.

Le problème : l'horloge Windows limité à 4 294 967 295 secondes (un peu moins de 50 jours). Les ordinateurs n'avaient pas été rebooté depuis cette durée.
En 2015, même erreur sur le Boing 787, qui doit être éteint au moins tous les 248 jours (heureusement pas de d'accident).

Autres problèmes logiciels

- Février 1991 : bug logiciel d'horodatage dans les batteries anti-missiles partiot : 28 morts.
- 2004 : logiciel de pension britanique plante : 1 milliard d'euros.
- 1988 : création involontaire d'un vers informatique (pour détecter des routes réseau). Coût 100 millions de dollars.
- Récemment : un F22 *reboot* (system out) lorsqu'il passe la ligne imaginaire de changement de jour dans le pacifique (division par zéro).
- ...

En général le problème n'est pas que logiciel, il vient d'une mauvaise organisation, d'un manque de spécification, d'un manque de tests, etc.

Plan

Schéma général

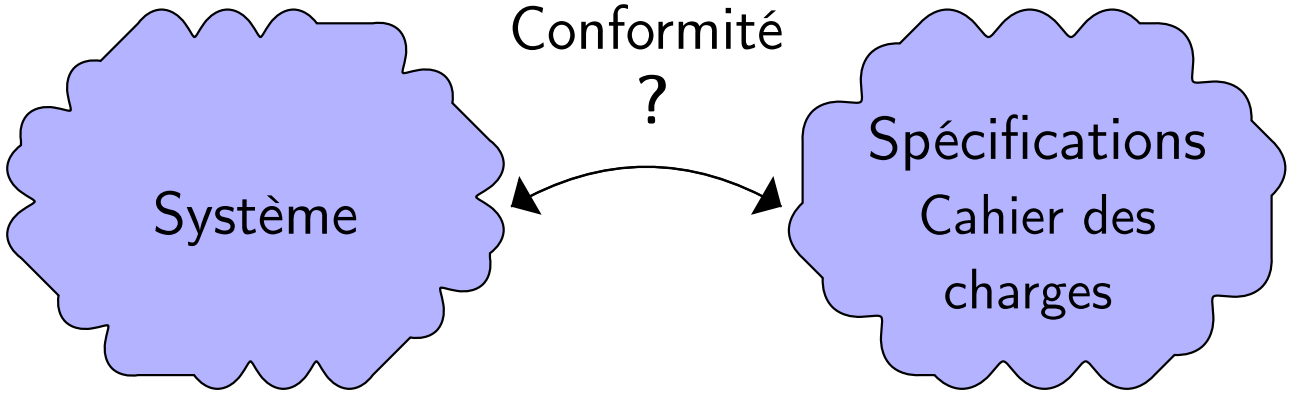


Schéma général

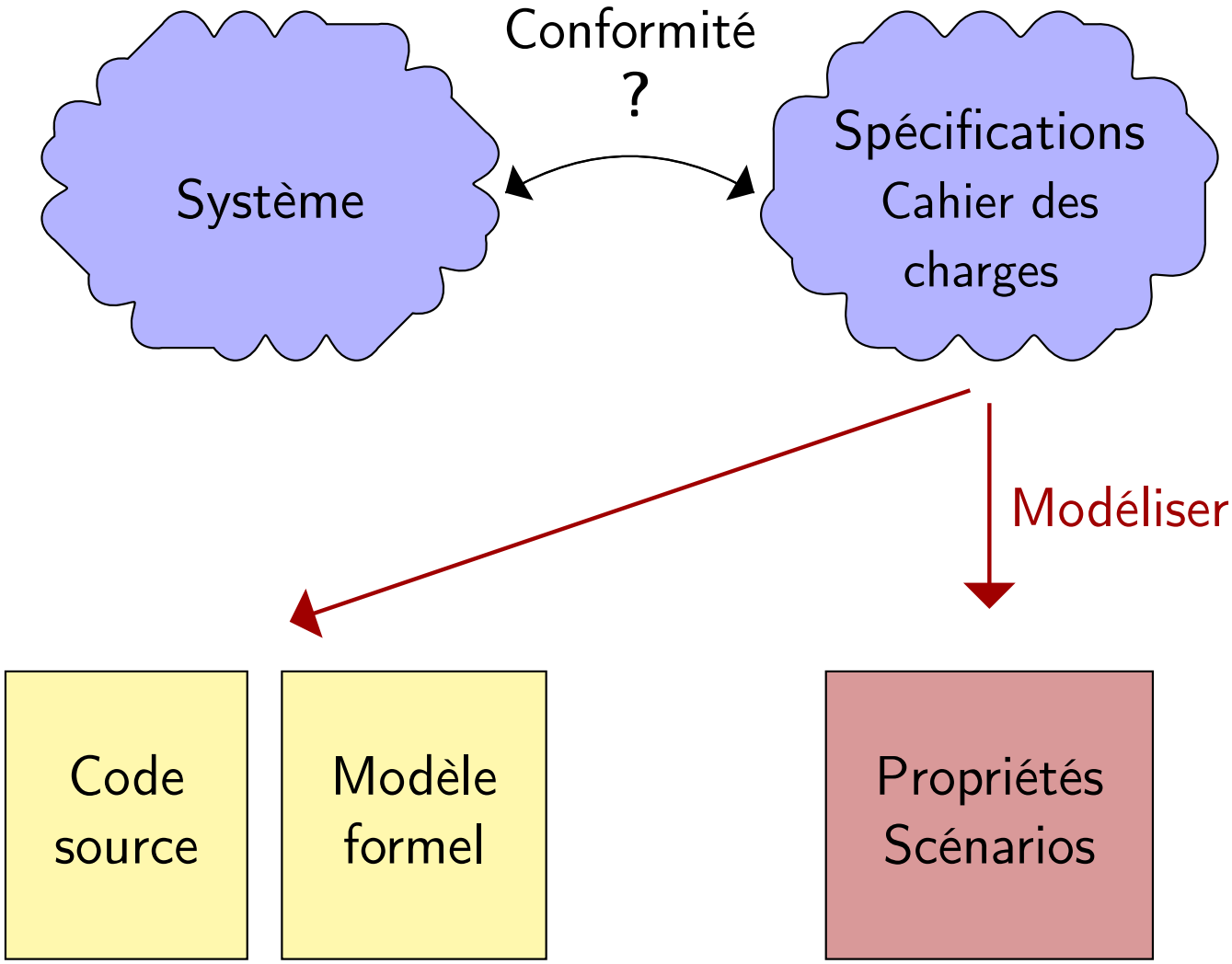
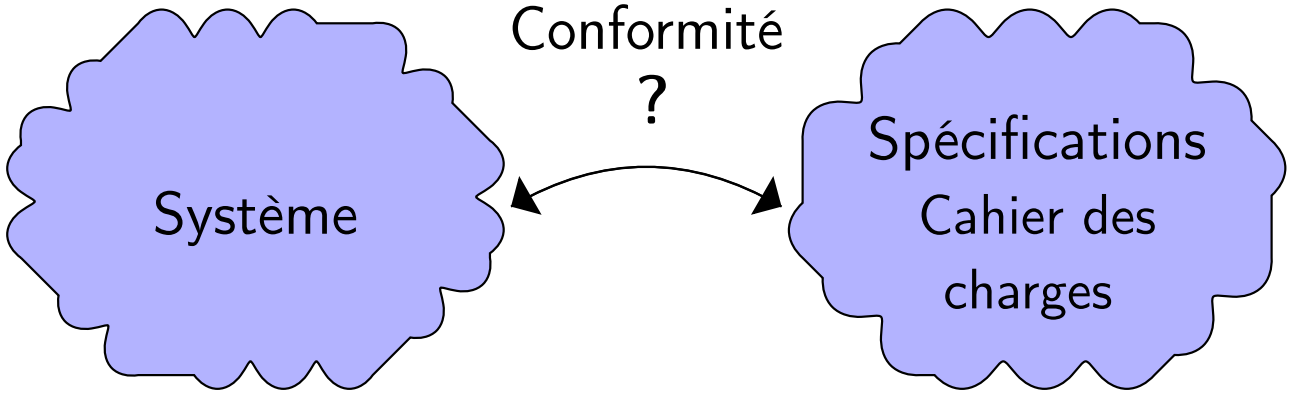


Schéma général



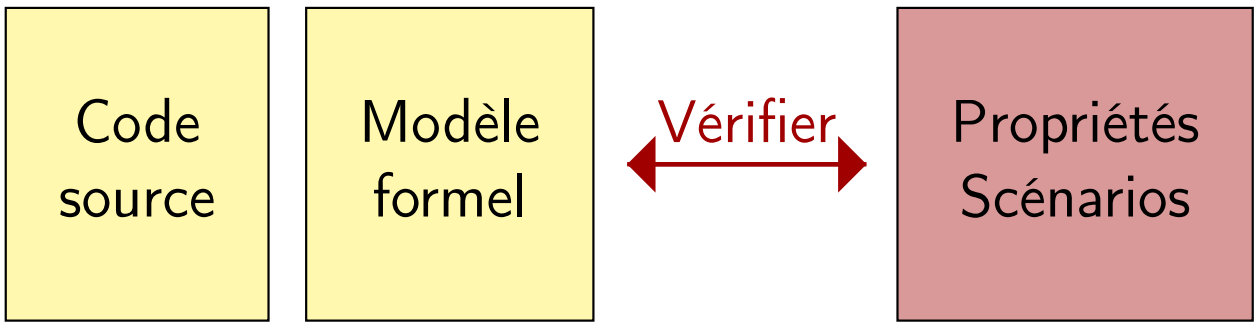
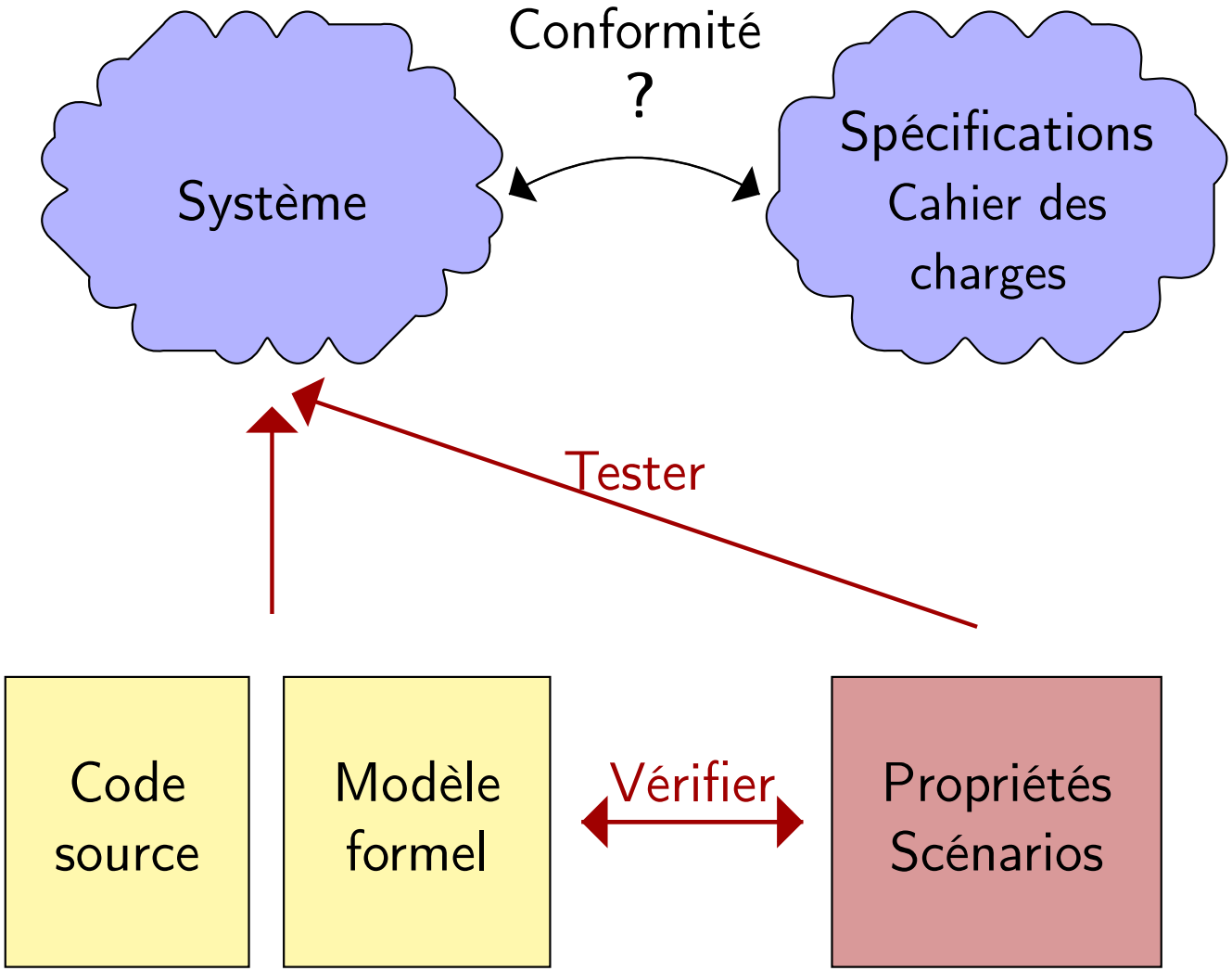


Schéma général



Approches automatiques

La vérification (preuve)

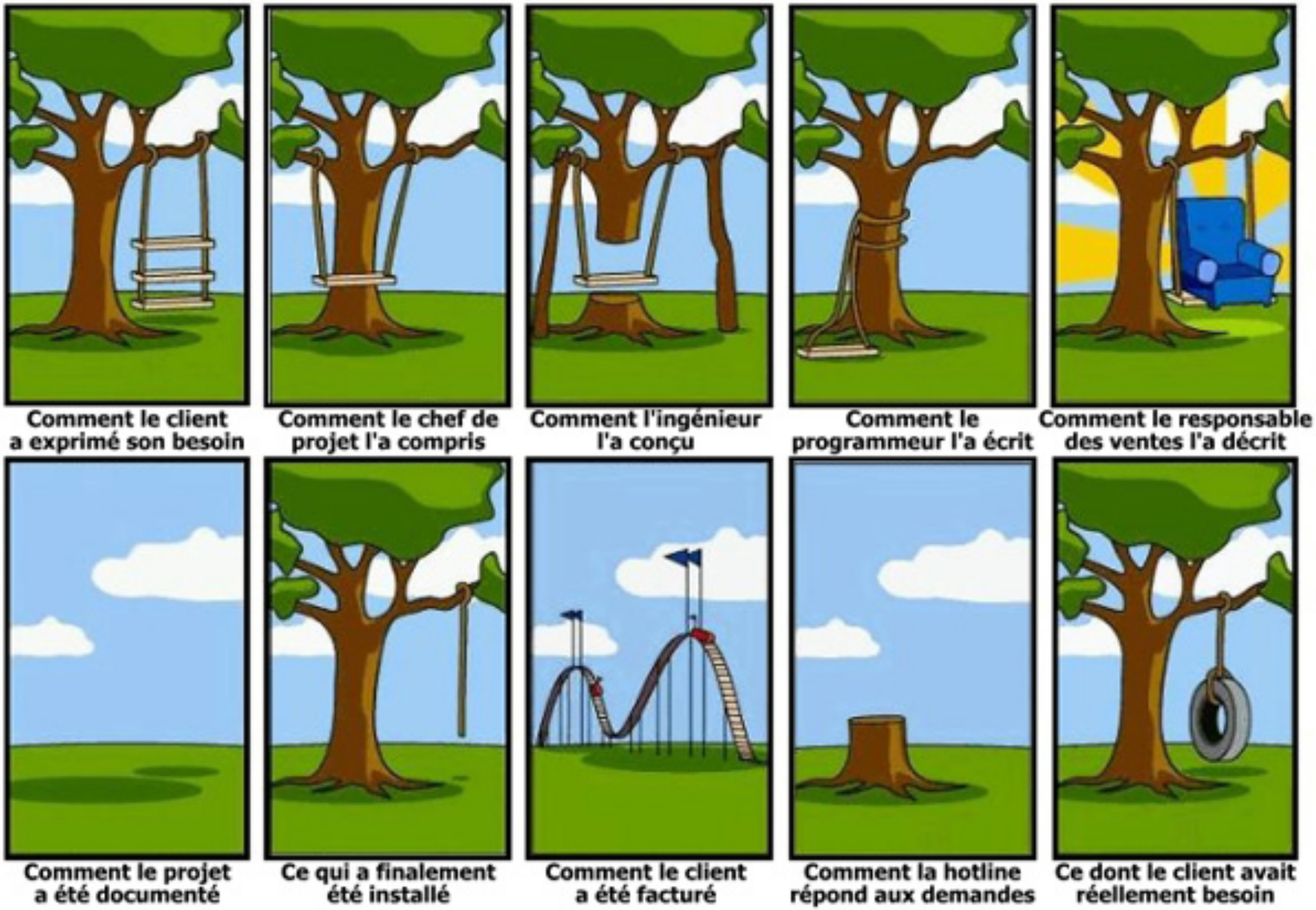
- Indécidabilité,
- Lourd à mettre en place (temps),
- Hautes qualifications,

La validation (test)

- Non exhaustif, sans garantie,
- Coûteux.

La logique est intensivement utilisé, notamment pour spécifier/modéliser les propriétés, mais aussi sur le plan algorithmique (pour la vérification et la validation).

La conception logicielle



Plan

Tests

Les programmes sont intensivement testés ; en simplifiant un peu :

- Les tests unitaires valident le comportement des fonctions individuellement,
- Les tests d'intégration valident les comportements du logiciel dans son intégralité,
- Les tests fonctionnels valident le déploiement de l'application.

Il existe aussi des tests de performance, de robustesse,...

Exemple de tests unitaires

Des propriétés vérifiées par un programme qui trie une liste ; L est la liste d'origine et Ls la liste retournée par la fonction. On doit avoir :

- $\text{len}(Ls) = \text{len}(L)$
- $\forall x \in Ls,\ x \in L$
- $\forall x \in L,\ x \in Ls$
- $\forall x \in Ls,\ x \geq Ls[0]$
- $\forall x \in Ls,\ x \leq Ls[-1]$
- $\forall i \in \text{range}(\text{len}(L) - 1),\ Ls[i] < Ls[i+1]$
- ...

Exemple

```
def montri(L):
    for j in range(len(L)):
        minimumindex=j
        for i in range(j,len(L)):
            if L[i] < L[minimumindex] :
                minimumindex=i
        z=L[minimumindex]
        L[minimumindex]=L[j]
        L[j]=z
```

```
    class Mestest(unittest.TestCase):

    def testlongueur(self):
        L=list(range(10,0,-1))
        trier.montri(L)
        self.assertEqual(len(L),10)

    def testmemeselements1(self):
        L=[]
        for i in range(10):
            L.append(random.random())
        K=copy.copy(L)
        trier.montri(L)
        for x in K:
            self.assertIn(x,L)
```

- $\text{len}(Ls) = \text{len}(L)$
- $\forall x \in Ls,\ x \in L$

Exécution

```
python3 -m unittest
```

```
...
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Exemple

```
def montri(L):
    for j in range(len(L)):
        minimumindex=j
        for i in range(j,len(L)):
            if L[i] < L[minimumindex] :
                minimumindex=i
        z=L[minimumindex]
        #L[minimumindex]=L[j]
        L[j]=z
```

```
    class Mestest(unittest.TestCase):

    def testlongueur(self):
        L=list(range(10,0,-1))
        trier.montri(L)
        self.assertEqual(len(L),10)

    def testmemeselements1(self):
        L=[]
        for i in range(10):
            L.append(random.random())
        K=copy.copy(L)
        trier.montri(L)
        for x in K:
            self.assertIn(x,L)
```

- $\text{len}(Ls) = \text{len}(L)$
- $\forall x \in Ls,\ x \in L$

Exécution

```
python3 -m unittest
```

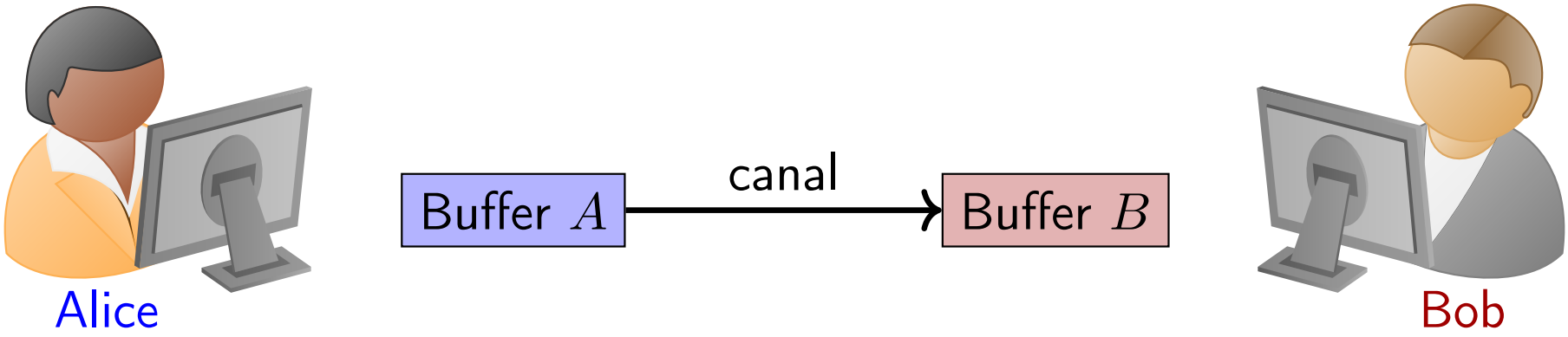
```
======================================================================
FAIL: testmemeselements2 (testunit.Mestest)
----------------------------------------------------------------------
Traceback (most recent call last):
File "/home/pcheam/pcheam/Enseignement/LD/Cours/testunit.py", line 31,
in testmemeselements2
    self.assertIn(x,K)
AssertionError: 4 not found in [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]

----------------------------------------------------------------------
Ran 3 tests in 0.000s
```

Tester

- Les tests s'exécutent sur des instances particulières, ce ne sont pas des preuves.
- Les spécifications sont fondamentales pour un développement par une équipe.
- L'utilisation de formalismes logiques évite les erreurs d'interprétation.
- Cours de Test en L3.

Plan

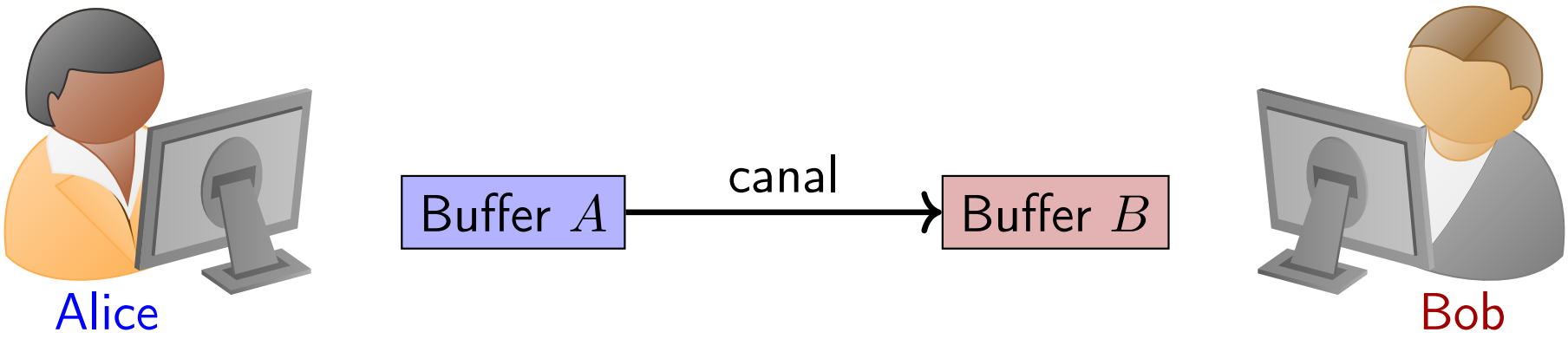
LTL : un exemple de canal de communication [2]



- Alice envoie des messages à Bob par un canal unidirectionnel.
- Leurs messages sont placés dans des buffers respectifs de capacités non bornées.
- A chaque étape, on sait quels messages sont dans chaque buffer, parmi un ensemble $\Sigma = \{m_1, \ldots, m_k\}$ de messages possibles.
- Une configuration du système est un couple de $2^{\Sigma} \times 2^{\Sigma}$.

2. Ch. Baier and J.-P. Katoen, Principles of Model Checking, MIT Press 2008

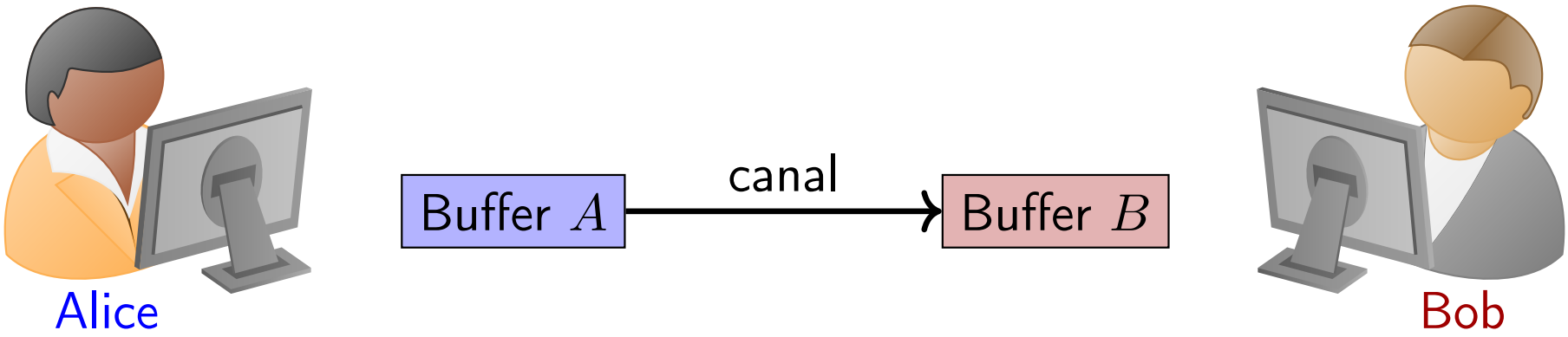
LTL : un exemple de canal de communication [2]



$(\emptyset, \emptyset)$

2. Ch. Baier and J.-P. Katoen, Principles of Model Checking, MIT Press 2008

LTL : un exemple de canal de communication [2]

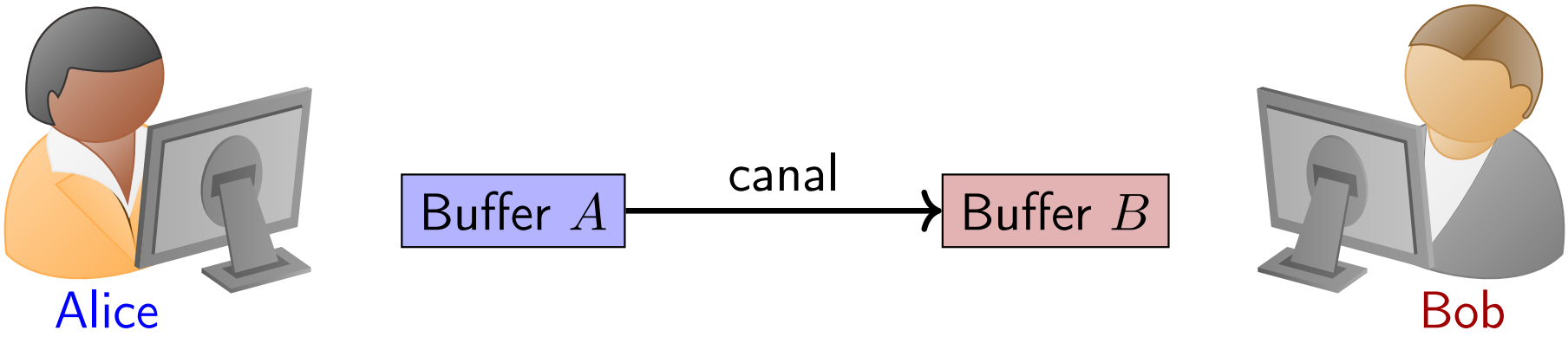


m_1

$(\emptyset, \emptyset) \rightarrow (\{m_1\}, \emptyset)$

2. Ch. Baier and J.-P. Katoen, Principles of Model Checking, MIT Press 2008

LTL : un exemple de canal de communication [2]

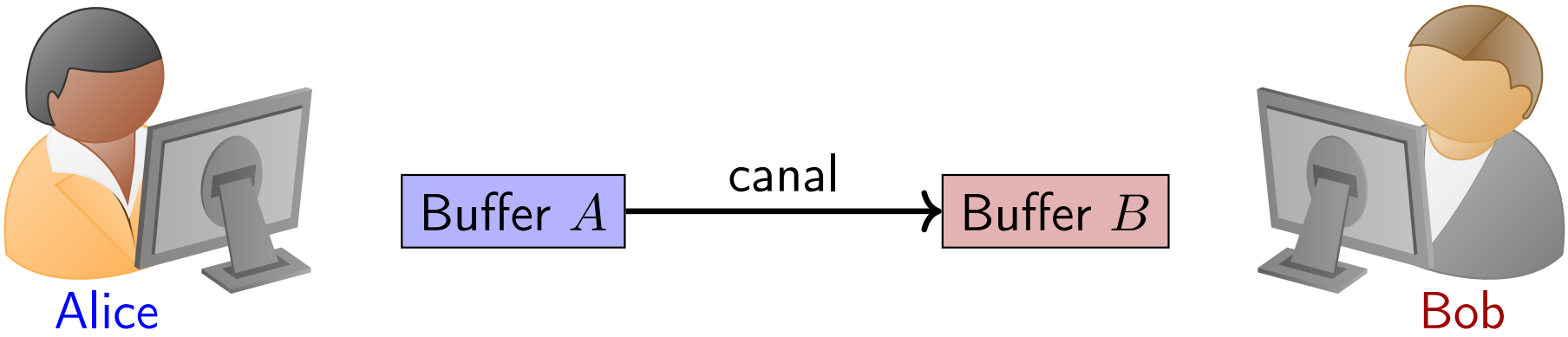


m_1, m_2

$(\emptyset, \emptyset) \rightarrow (\{m_1\}, \emptyset) \rightarrow (\{m_1, m_2\}, \emptyset)$

2. Ch. Baier and J.-P. Katoen, Principles of Model Checking, MIT Press 2008

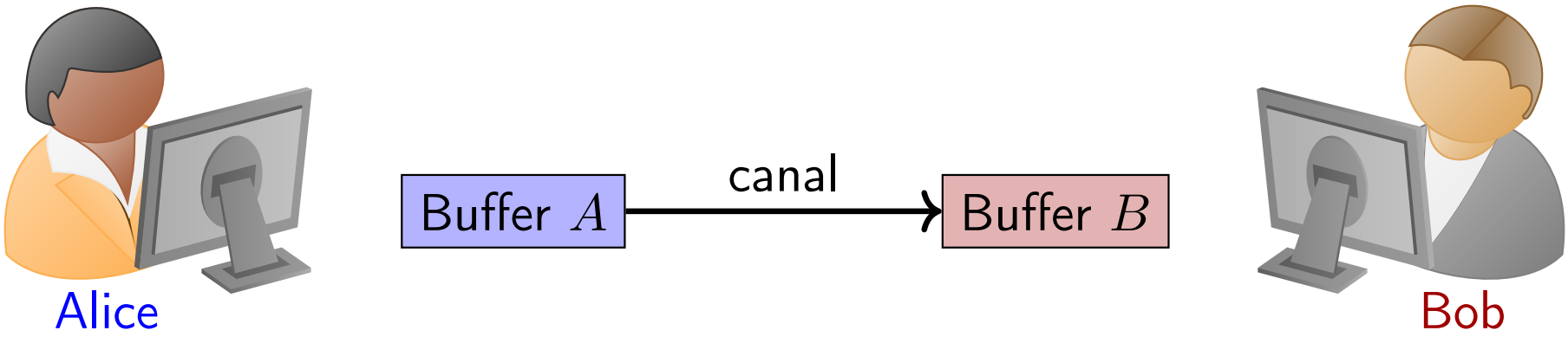
LTL : un exemple de canal de communication [2]



$$(\emptyset, \emptyset) \rightarrow (\{m_1\}, \emptyset) \rightarrow (\{m_1, m_2\}, \emptyset) \rightarrow (\{m_2\}, \{m_1\})$$

2. Ch. Baier and J.-P. Katoen, Principles of Model Checking, MIT Press 2008

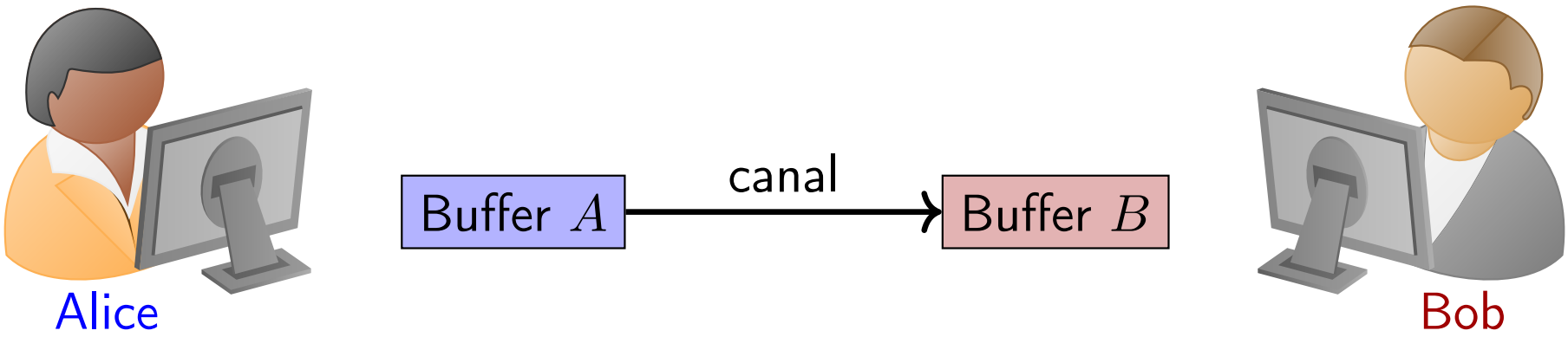
LTL : un exemple de canal de communication [2]



$$(\emptyset, \emptyset) \rightarrow (\{m_1\}, \emptyset) \rightarrow (\{m_1, m_2\}, \emptyset) \rightarrow (\{m_2\}, \{m_1\}) \rightarrow (\{m_2\}, \emptyset) \rightarrow \ldots$$

2. Ch. Baier and J.-P. Katoen, Principles of Model Checking, MIT Press 2008

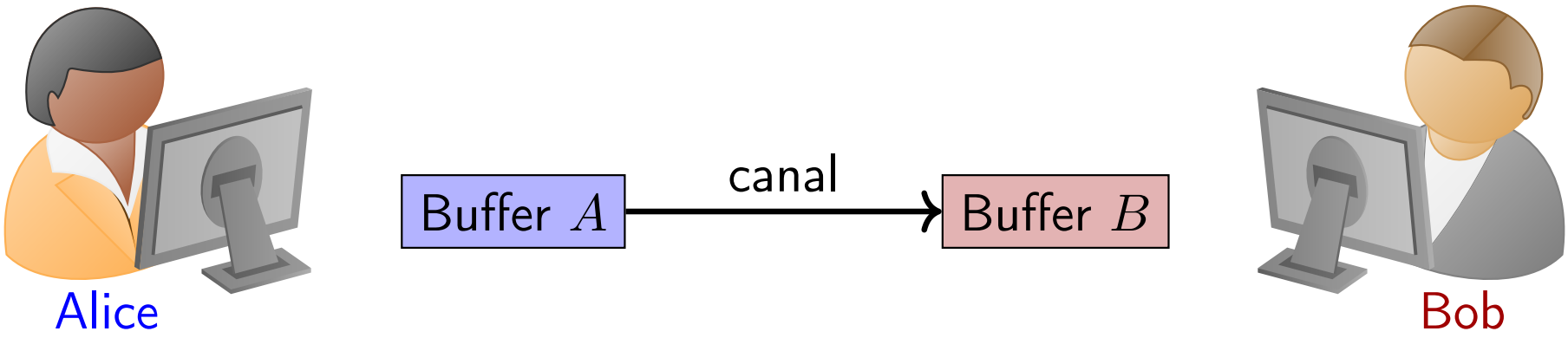
LTL : un exemple de canal de communication [2]



On note par $m \in A$ le fait que le message m soit dans la première composante et dualement $m \in B$ pour la seconde :
Si la configuration courante est $(\{m_2, m_3\}, \{m_1\})$, alors à cet instant $m_2 \in A$ est vraie, de même que $m_1 \in B$, mais pas $m_2 \in B$.

2. Ch. Baier and J.-P. Katoen, Principles of Model Checking, MIT Press 2008

LTL : un exemple de canal de communication [2]

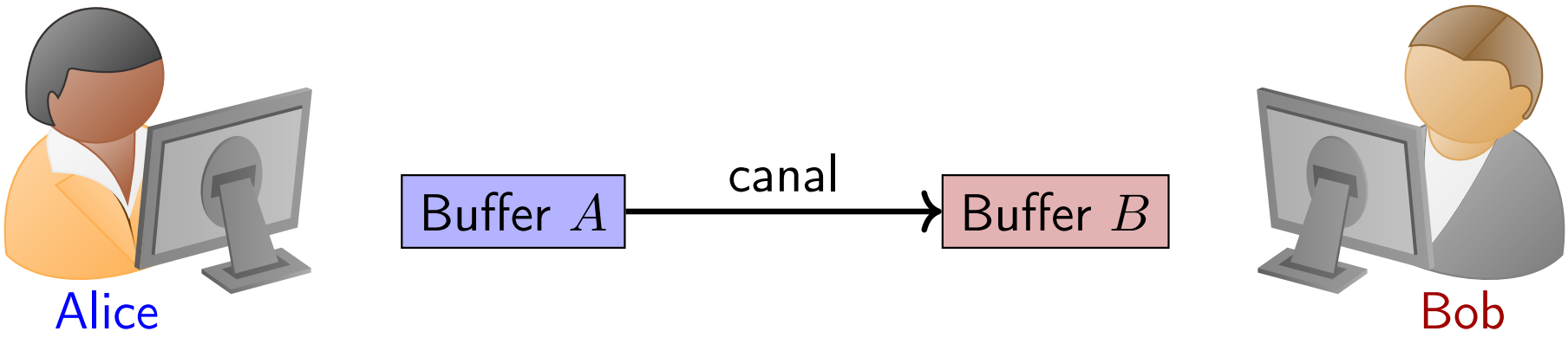


Le canal est sûr et tous les messages sont envoyés :

$$\Box(m \in A \Rightarrow \diamond\, m \in B)$$

2. Ch. Baier and J.-P. Katoen, Principles of Model Checking, MIT Press 2008

LTL : un exemple de canal de communication [2]

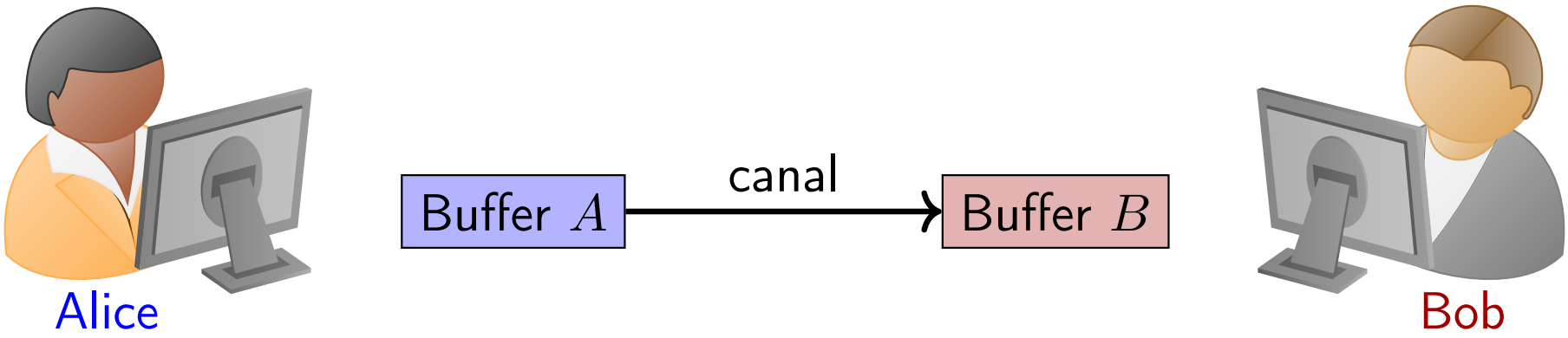


Un même message ne peut pas être dans les deux buffers à la fois :

$$\Box(\neg(m \in A \wedge m \in B))$$

2. Ch. Baier and J.-P. Katoen, Principles of Model Checking, MIT Press 2008

LTL : un exemple de canal de communication [2]

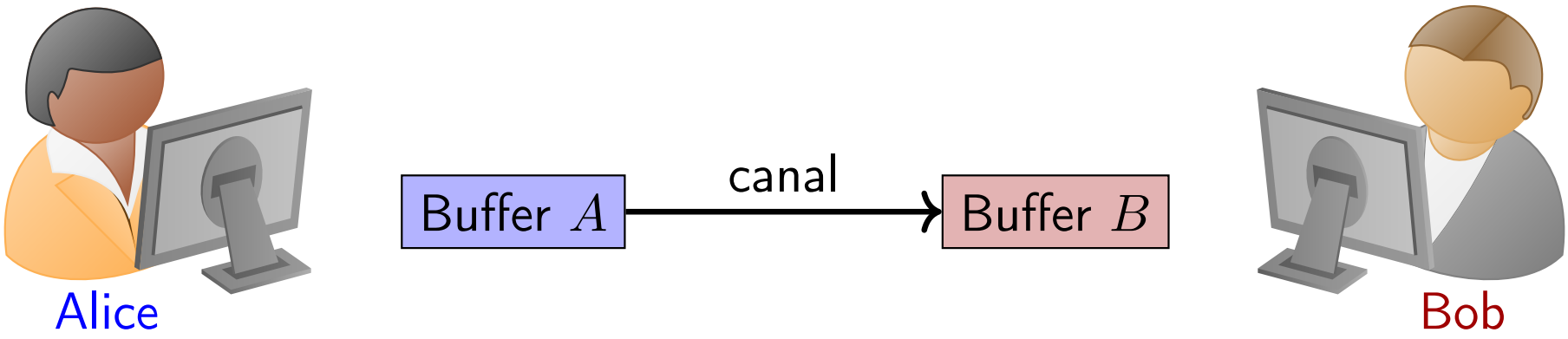


Aucun message ne disparaît :

$$\Box(m \in A \Rightarrow (m \in A \,\mathcal{U}\, m \in B))$$

2. Ch. Baier and J.-P. Katoen, Principles of Model Checking, MIT Press 2008

LTL : un exemple de canal de communication [2]

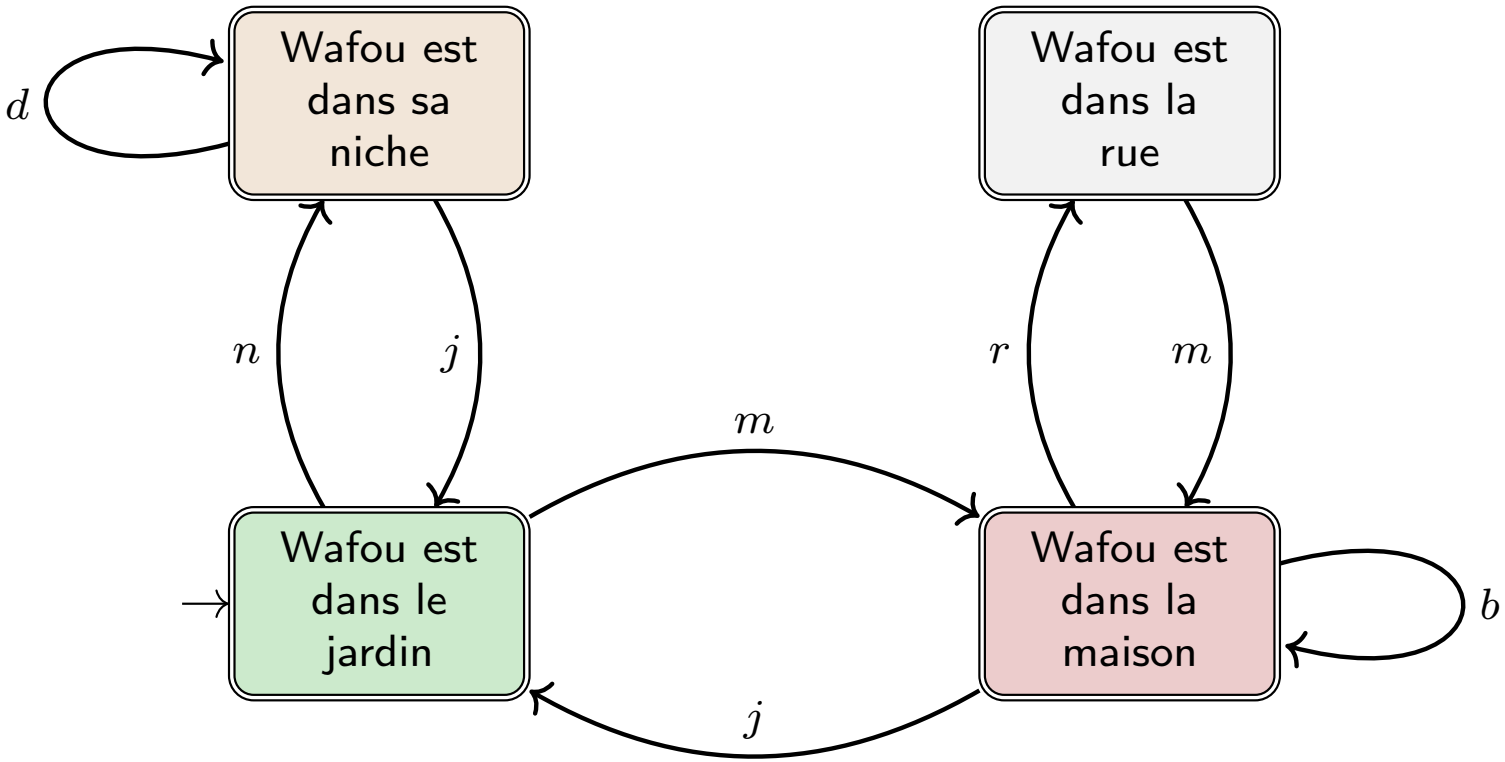


L'ordre des messages est respecté :

$$\Box\Big(m \in A \wedge \neg m' \in A \wedge \Diamond(m' \in A) \Rightarrow \Diamond\Big(m \in B \wedge \neg m' \in B \wedge \Diamond(m' \in B)\Big)\Big)$$

2. Ch. Baier and J.-P. Katoen, Principles of Model Checking, MIT Press 2008

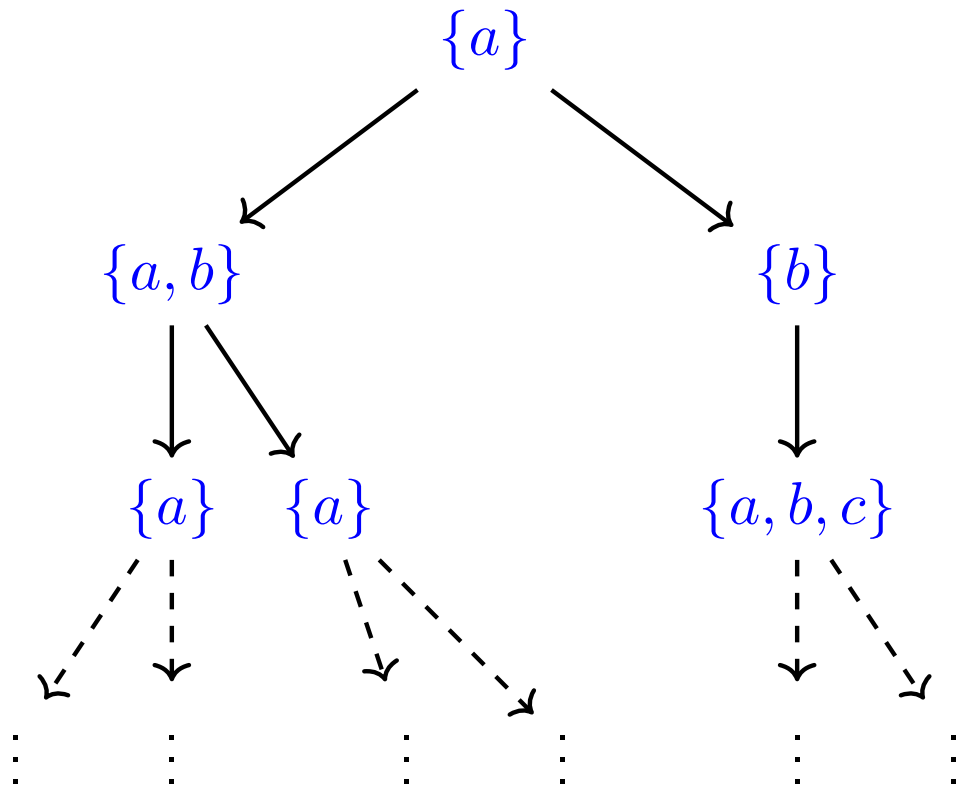
Computational Tree Logic



On ne peut pas, avec LTL, spécifier une propriété du type
Le chien Wafou peut toujours revenir dans sa niche.

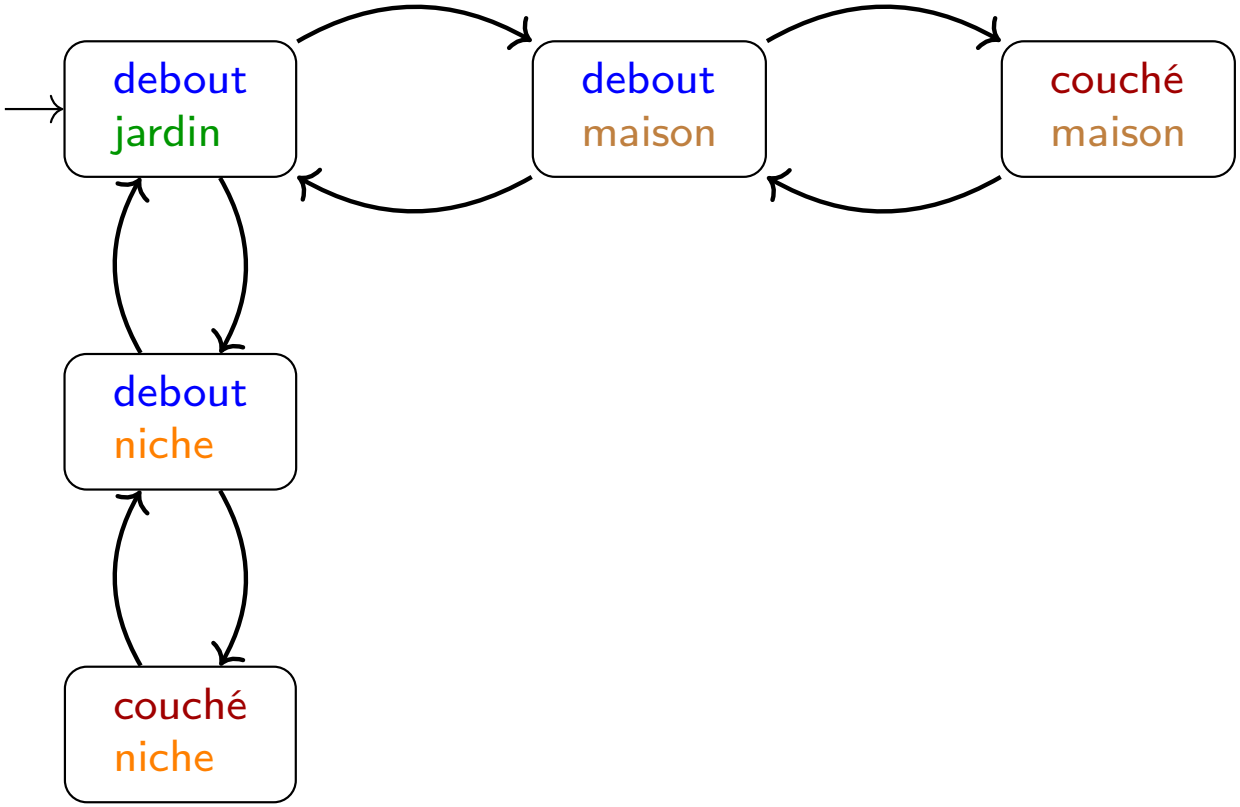
CTL une logique sur les arbres infinis

On se donne un alphabet Σ On considère des arbres infinis en profondeur, mais d'arité finie. Chaque sommet de l'arbre est étiqueté par un sous-ensemble de Σ.

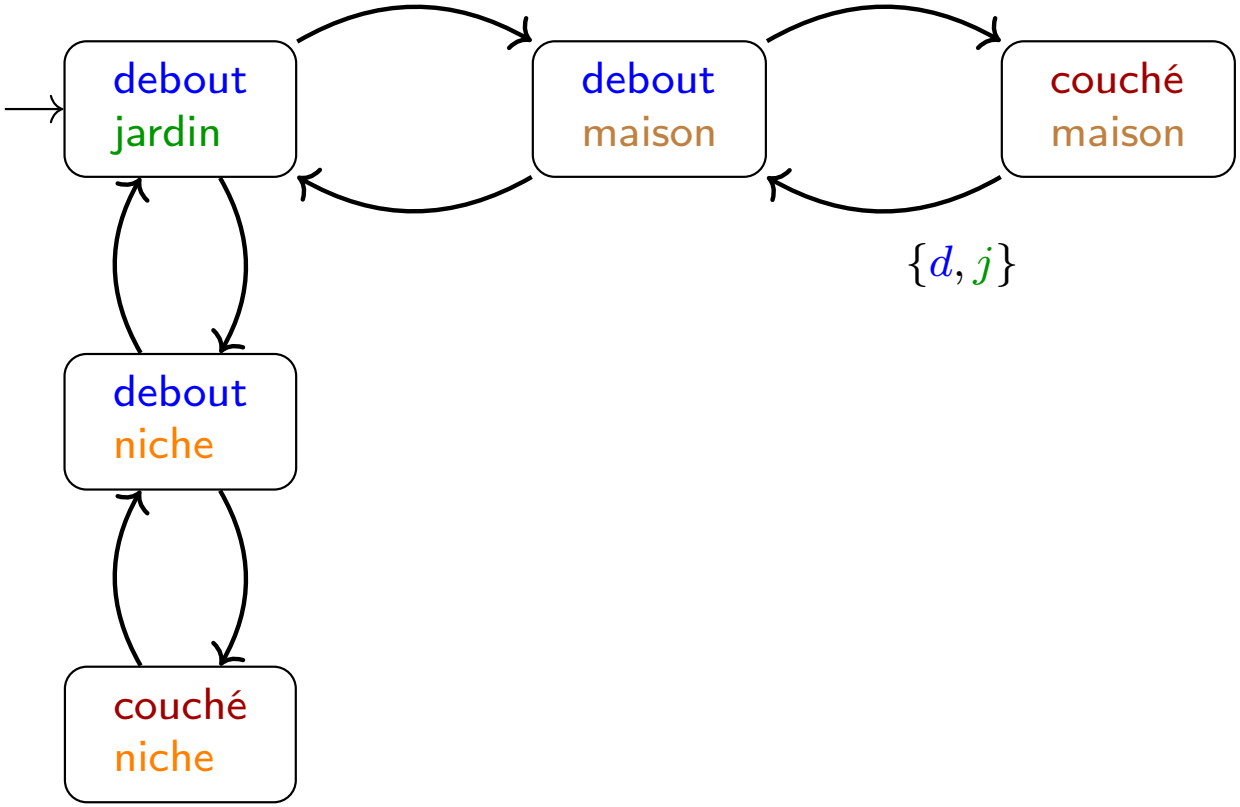


Si s est une position de l'arbre $\mathcal{A}$, alors on note $\mathcal{A}_s$ l'étiquette correspondante. $\mathcal{A}_\varepsilon = \{a\}$

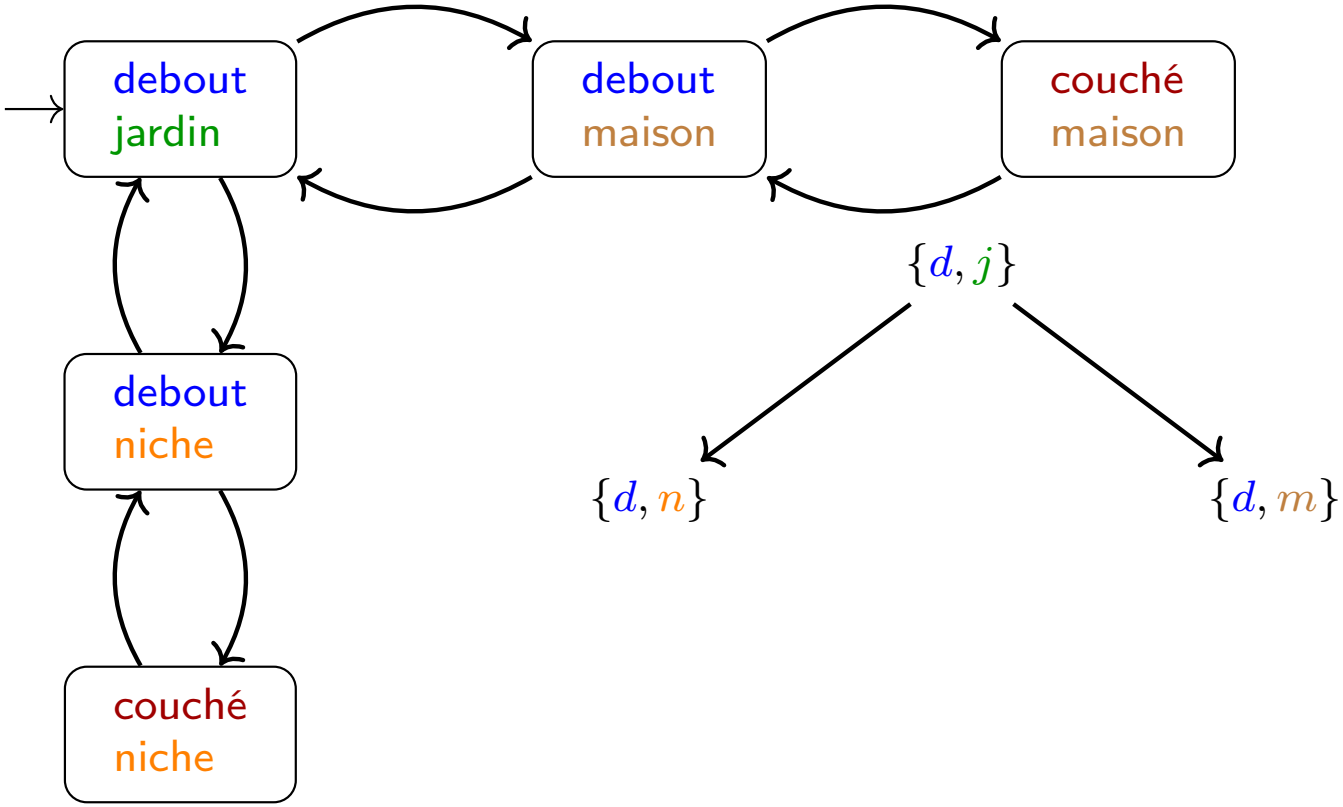
Dépliement d'un graphe



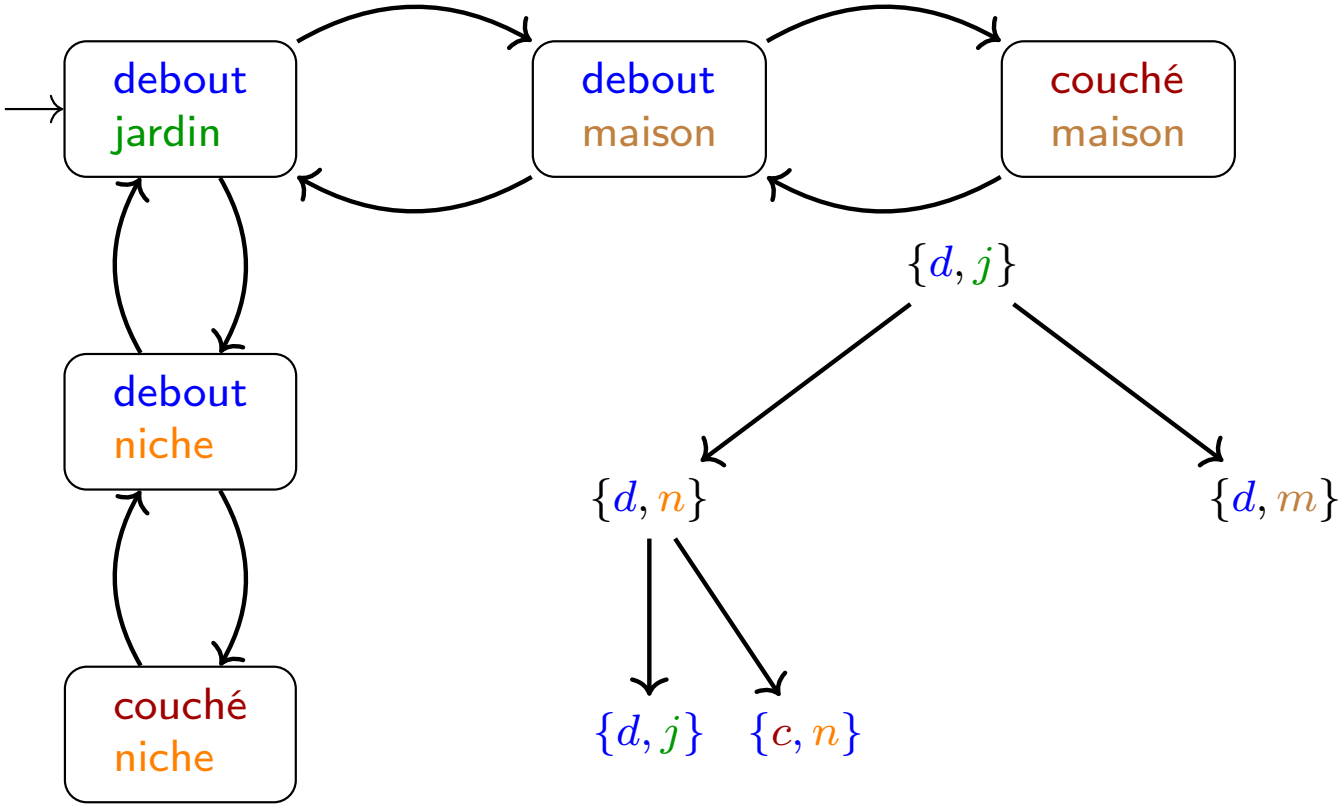
Dépliement d'un graphe



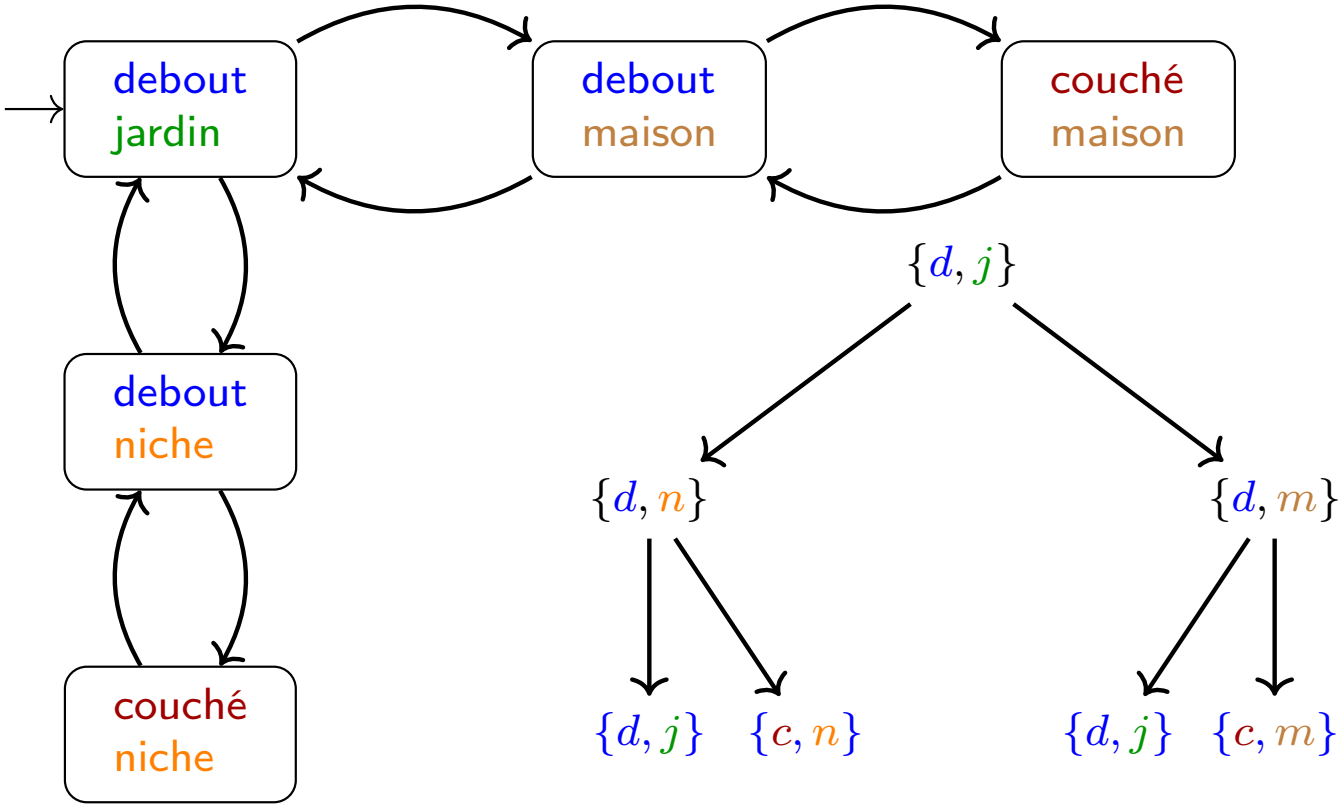
Dépliement d'un graphe



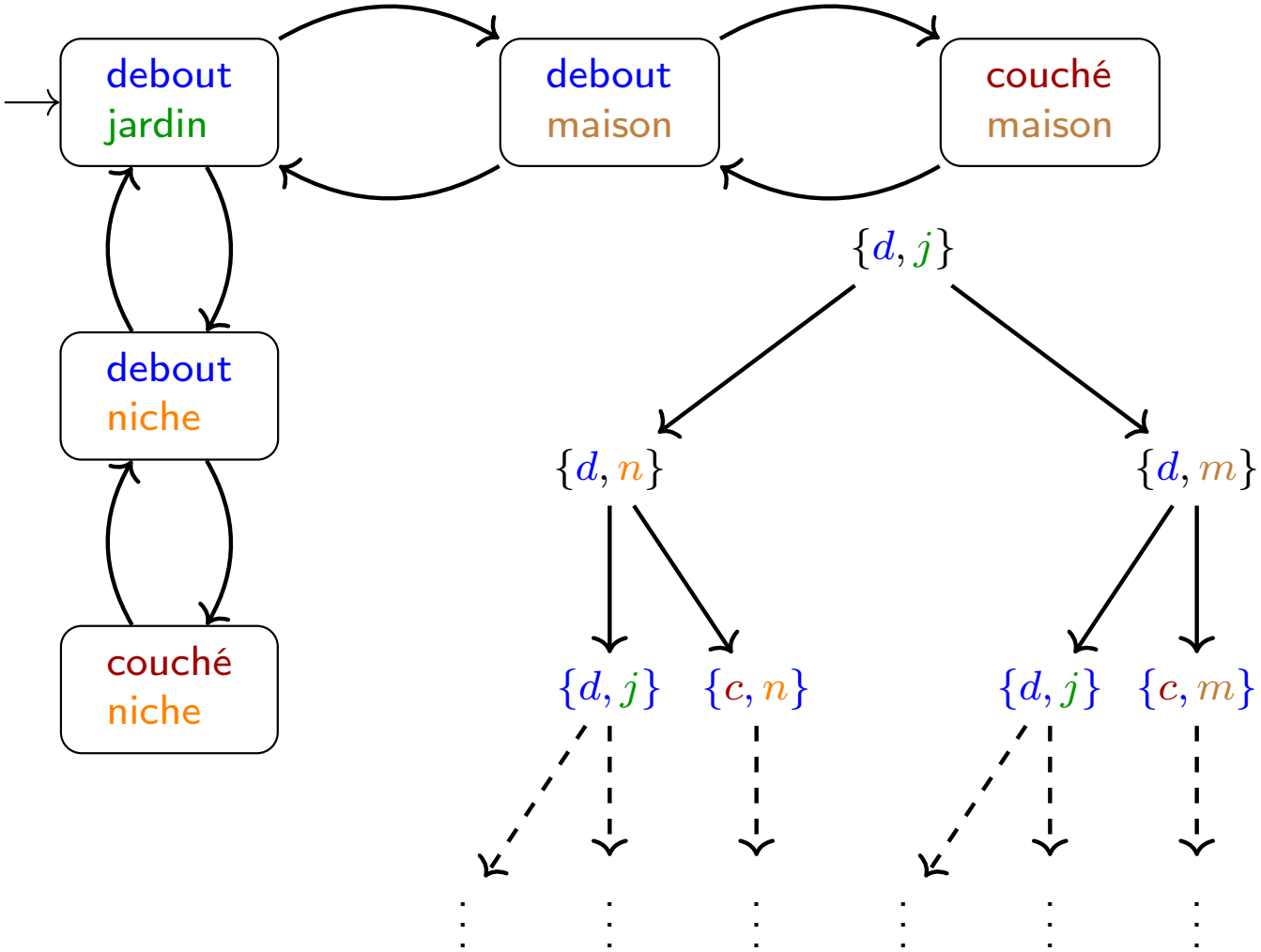
Dépliement d'un graphe



Dépliement d'un graphe



Dépliement d'un graphe



CTL : syntaxe

Une formule CTL sur un alphabet Σ est

- une lettre de Σ, ou
- $\top$ ou $\bot$, ou
- une combinaison booléenne de formules CTL, ou
- une formule du type
  - $\forall \circ \varphi$,
  - $\exists \circ \varphi$,
  - $\forall(\varphi \, \mathcal{U} \, \psi)$,
  - $\exists(\varphi \, \mathcal{U} \, \psi)$,

  où φ et ψ sont des formules CTL.

$$\forall \circ (a \wedge b)$$

$$\exists(\forall \circ (a \wedge b) \, \mathcal{U} c)$$

Nouveaux opérateurs

On définit (comme pour LTL) les opérateurs

- $\forall\Box\varphi := \forall(\top\ \mathcal{U}\ \varphi)$,
- $\exists\Box\varphi := \exists(\top\ \mathcal{U}\ \varphi)$,
- $\forall\Diamond\varphi := \neg\forall\Box\neg\varphi$,
- $\exists\Diamond\varphi := \neg\exists\Box\neg\varphi$,

$$\forall\Box(a \wedge b)$$

Comme LTL mais avec un quantificateur associé à chaque opérateur temporel

Sémantique

Une formule CTL φ s'évalue sur les positions s (les sommets) d'un arbre $\mathcal{A}$.

- $(\mathcal{A}, s) \models a$ si $a \in \mathcal{A}_s$,
- $(\mathcal{A}, s) \models \varphi \wedge \psi$ si $(\mathcal{A}, s) \models \varphi$ et $(\mathcal{A}, s) \models \psi$.
- $(\mathcal{A}, s) \models \varphi \vee \psi$ si $(\mathcal{A}, s) \models \varphi$ ou $(\mathcal{A}, s) \models \psi$.
- $(\mathcal{A}, s) \models \varphi \Rightarrow \psi$ si $(\mathcal{A}, s) \not\models \varphi$ ou $(\mathcal{A}, s) \models \psi$.

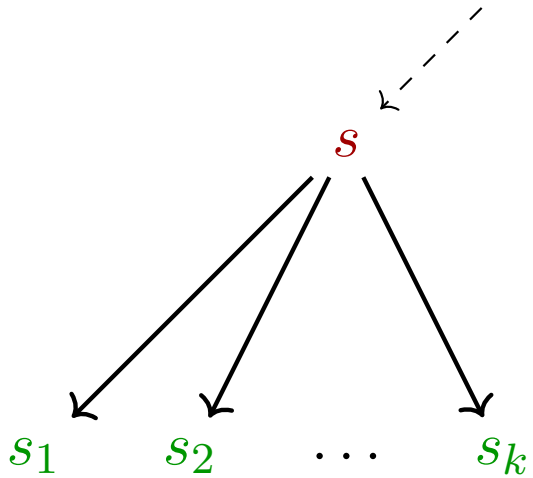
Classique pour les opérateur booléens

Sémantique

Une formule CTL φ s'évalue sur les positions s (les sommets) d'un arbre $\mathcal{A}$.

Si les fils de du sommet à la positions s sont $s_1, \ldots, s_k$, alors

- $(\mathcal{A}, s) \models \exists \circ \varphi$ si **au moins un** des s_i vérifie $(\mathcal{A}, s_i) \models \varphi$
- $(\mathcal{A}, s) \models \exists \circ \varphi$ si **tous les** s_i vérifient $(\mathcal{A}, s_i) \models \varphi$

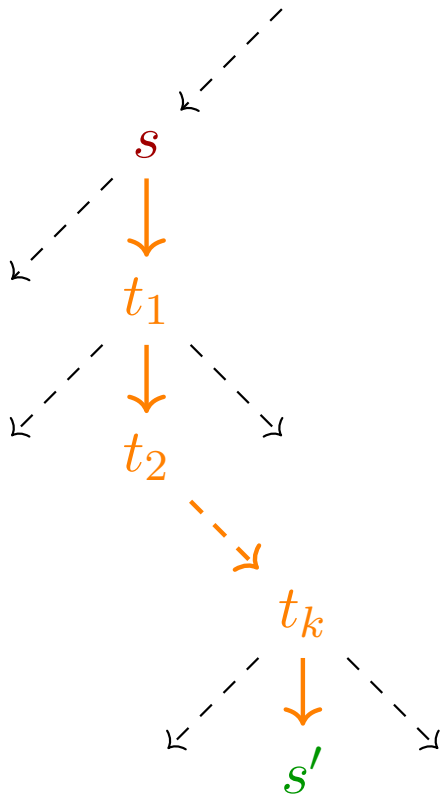


Sémantique

Une formule CTL φ s'évalue sur les positions s (les sommets) d'un arbre $\mathcal{A}$.

$(\mathcal{A}, s) \models \exists(\varphi \mathcal{U}\ \psi)$ s'il existe une position s' descendante de s telle que

- $(\mathcal{A}, s') \models \psi$
- pour toute position t entre s (comprise) et s' (non comprise), $(\mathcal{A}, t) \models \varphi$



Sémantique

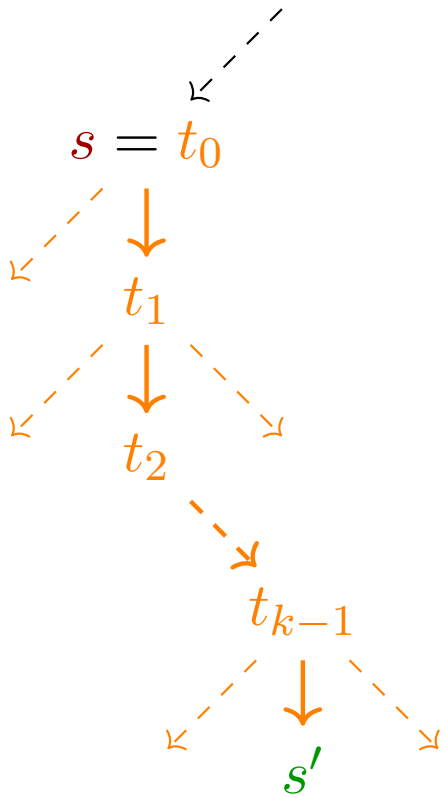
Une formule CTL φ s'évalue sur les positions s (les sommets) d'un arbre $\mathcal{A}$.

$(\mathcal{A}, s) \models \forall(\varphi \mathcal{U}\ \psi)$ si **pour toute** suite infinie de positions consécutives

$$t_0 = s, t_1, \ldots, t_i, \ldots$$

il existe une position $s' = t_k$ telle que

- $(\mathcal{A}, s') \models \psi$
- pour toute position t_i pour $i < k$, $(\mathcal{A}, t_i) \models \varphi$



Sémantique

Une formule CTL φ s'évalue sur les positions s (les sommets) d'un arbre $\mathcal{A}$.

- $(\mathcal{A}, s) \models \exists\Box\varphi$ s'**il existe une branche** partant de s sur laquelle, à chaque position s', $(\mathcal{A}, s') \models \varphi$

Sémantique

Une formule CTL φ s'évalue sur les positions s (les sommets) d'un arbre $\mathcal{A}$.

- $(\mathcal{A}, s) \models \exists\Box\varphi$ s'**il existe une branche** partant de s sur laquelle, à chaque position s', $(\mathcal{A}, s') \models \varphi$
- $(\mathcal{A}, s) \models \forall\Box\varphi$ si **pour toute branche** partant de s à chaque position s', $(\mathcal{A}, s') \models \varphi$

Sémantique

Une formule CTL φ s'évalue sur les positions s (les sommets) d'un arbre $\mathcal{A}$.

- $(\mathcal{A}, s) \models \exists\Box\varphi$ s'**il existe une branche** partant de s sur laquelle, à chaque position s', $(\mathcal{A}, s') \models \varphi$
- $(\mathcal{A}, s) \models \forall\Box\varphi$ si **pour toute branche** partant de s à chaque position s', $(\mathcal{A}, s') \models \varphi$
- $(\mathcal{A}, s) \models \exists \diamond \varphi$ s'**il existe une branche** partant de s sur laquelle il existe une position s' telle que, $(\mathcal{A}, s') \models \varphi$

Sémantique

Une formule CTL φ s'évalue sur les positions s (les sommets) d'un arbre $\mathcal{A}$.

- $(\mathcal{A}, s) \models \exists\Box\varphi$ s'**il existe une branche** partant de s sur laquelle, à chaque position s', $(\mathcal{A}, s') \models \varphi$
- $(\mathcal{A}, s) \models \forall\Box\varphi$ si **pour toute branche** partant de s à chaque position s', $(\mathcal{A}, s') \models \varphi$
- $(\mathcal{A}, s) \models \exists \diamond \varphi$ s'**il existe une branche** partant de s sur laquelle il existe une position s' telle que, $(\mathcal{A}, s') \models \varphi$
- $(\mathcal{A}, s) \models \forall \diamond \varphi$ si **pour toute branche** partant de s il existe une position s' telle que, $(\mathcal{A}, s') \models \varphi$

Sémantique

Une formule CTL φ s'évalue sur les positions s (les sommets) d'un arbre $\mathcal{A}$.

Sémantique

Un arbre $\mathcal{A}$ satisfait une formule CTL φ s'il l'a satisfait à sa racine.

Exemples

- a : la racine de l'arbre contient a.
- $\exists \circ a$: au moins un des fils de la racine contient a.
- $\forall \circ a$: tous les fils de la racine contiennent a.
- $\forall \Box a$: tous les sommets de l'arbre contiennent a.
- $\forall \diamond a$: toutes les branche de l'arbre contiennent au moins un a.
- $\exists \diamond \forall \Box \neg b$: L'arbre contient un sous arbre dont aucun sommet ne contient b.

Plan

Preuve de programme

Prouver un programme, c'est donner une preuve mathématique qu'il satisfait une propriété donnée :

- Preuve de terminaison : le programme termine
- Preuve de correction : le programme fait bien ce qu'il est sensé faire.

Définition

Un algorithme *termine* pour une entrée donnée si un *return* est exécuté après un nombre fini d'instructions.

Terminaison

La terminaison est un problème qui peut être très difficile.

Exemple

```
def Syracuse(n):
    while (n !=1):
        if n%2 == 0 :
            n=n/2
        else :
            n=3*n+1
    return True
```

On ne sait pas si cet algorithme termine toujours !

Primalité : terminaison vs correction

Un entier $n > 2$ est non premier si et seulement s'il admet un diviseur différent de 1 et de lui-même.

```
def IsNotPrime(n):# n est un entier plus grand que 2
    x=1
    while (n%x !=0 or x==1 or x==n):
        x=x+1
    if x < n :
        return True
    else :
        return False
```

Ce code retourne `True` ssi l'entier n n'est pas premier. Dans le cas contraire il ne peut retourner que `False`.

Prouver un programme : la logique de Hoare

Les formules sont du type

$$HPC$$

qui signifient intuitivement que si les conditions H sont vérifié avant l'exécution du programme P alors les conditions C le sont après l'exécution du programme.

Exemple

`x=41, l=[1,2,3]`

```
x=x+1
l.append(3)
```

`x=42, len(l)=4`

Prouver un programme : la logique de Hoare

On dispose de règles de déduction du type :

$$\frac{H \ P_1 \ C \qquad C \ P_2 \ D}{H \ P_1; P_2 \ D} \text{ (sequ.)}$$

Pour les appels successifs de code.

Prouver un programme : la logique de Hoare

On dispose de règles de déduction du type :

$$\frac{H \wedge B \ P_1 \ C \qquad \neg H \wedge B \ P_2 \ C}{B \text{ Si } H \text{ alors } P_1 \text{ sinon } P_2 \ C} \text{ (séqu.)}$$

Pour les conditions.

Prouver un programme : la logique de Hoare

On dispose de règles de déduction du type :

$$\frac{I \wedge B \ P \ I}{I \text{ Tant que } B \text{ faire } P \ I} \text{ (séqu.)}$$

Pour les boucles ; I s'appelle un **invariant**.

Plan

Conclusion

La logique est un outil important en informatique

- pour la conception et la spécification de programme,
- pour la preuve de correction,
- pour le test,
- ...