Améliorer les preuves de terminaison en coopérant

Marc Brockschmidt [1,2] Byron Cook [2,3] **Carsten Fuhs** [3]

[1]Université d'Aix-la-Chapelle (RWTH Aachen University)

[2]Microsoft Research

[3]University College London

Septièmes rencontres de la communauté française de compilation

Dammaire-les-Lys, France

5 décembre 2013

La terminaison des programmes, pourquoi l'analyser ?

La terminaison des programmes, pourquoi l'analyser ?

1. **Programme** : produit un résultat

La terminaison des programmes, pourquoi l'analyser ?

1. **Programme** : produit un résultat
2. **Input handler** : le système réagit

La terminaison des programmes, pourquoi l'analyser ?

1. **Programme** : produit un résultat
2. **Input handler** : le système réagit
3. **Preuve mathématique** : l'induction est valide

La terminaison des programmes, pourquoi l'analyser ?

1. **Programme** : produit un résultat
2. **Input handler** : le système réagit
3. **Preuve mathématique** : l'induction est valide
4. **Processus biologique** : atteint un état stable

La terminaison des programmes, pourquoi l'analyser ?

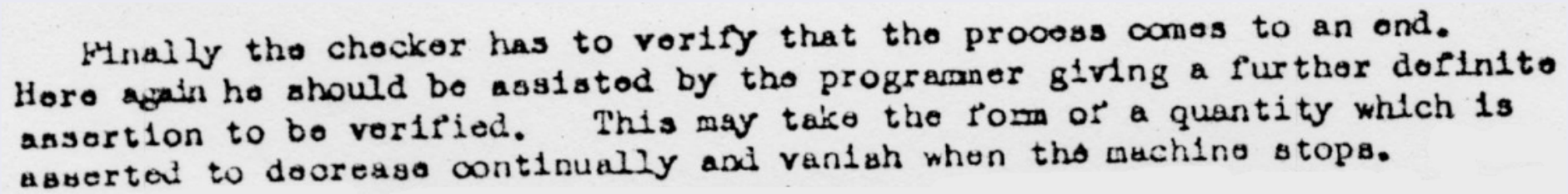
1. **Programme** : produit un résultat
2. **Input handler** : le système réagit
3. **Preuve mathématique** : l'induction est valide
4. **Processus biologique** : atteint un état stable

Des variations du même problème :

- 2 est un cas spécial de 1
- 3 peut être interprété comme 1
- 4 est une version probabiliste de 1

L'analyse de terminaison : La méthode classique

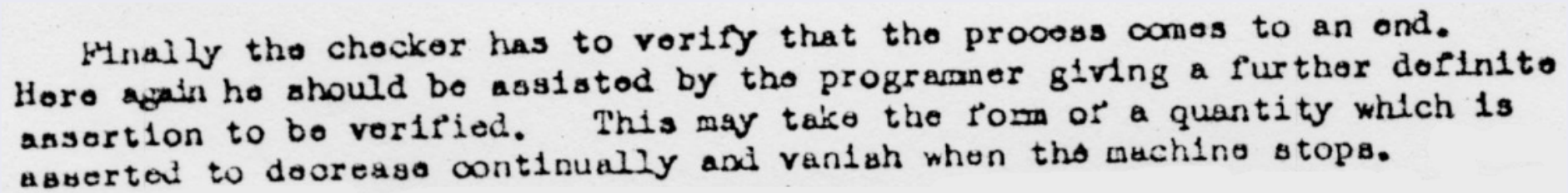
Turing 1949


Finally the checker has to verify that the process comes to an end. Here again he should be assisted by the programmer giving a further definite assertion to be verified. This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops.

« Enfin le contrôleur doit vérifier que le processus arrive à une fin. [...] Cela peut prendre la forme d'une quantité qui est affirmée à diminuer continûment et à disparaître quand la machine s'arrête. »

L'analyse de terminaison : La méthode classique

Turing 1949

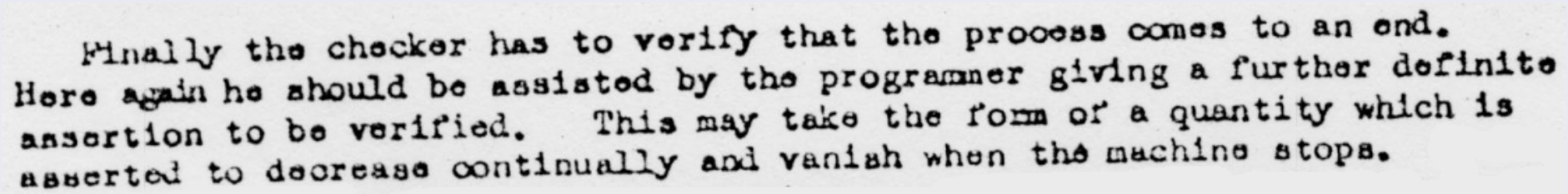

Finally the checker has to verify that the process comes to an end. Here again he should be assisted by the programmer giving a further definite assertion to be verified. This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops.

≪ Enfin le contrôleur doit vérifier que le processus arrive à une fin. [...] Cela peut prendre la forme d'une quantité qui est affirmée à diminuer continûment et à disparaître quand la machine s'arrête. ≫

1. Trouve **fonction de rang** f (≪ quantité ≫)

L'analyse de terminaison : La méthode classique

Turing 1949

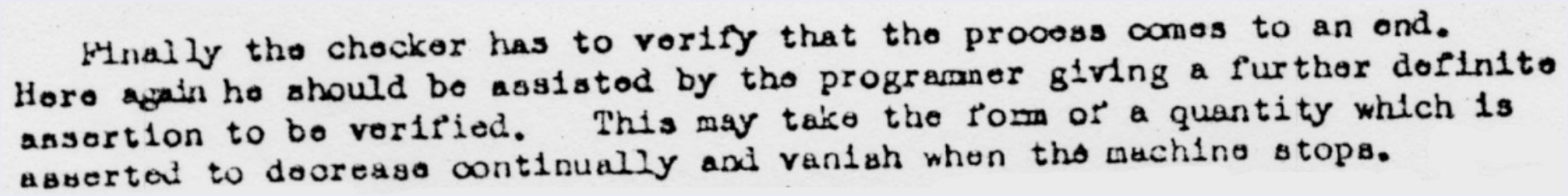

Finally the checker has to verify that the process comes to an end. Here again he should be assisted by the programmer giving a further definite assertion to be verified. This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops.

« Enfin le contrôleur doit vérifier que le processus arrive à une fin. [...] Cela peut prendre la forme d'une quantité qui est affirmée à diminuer continûment et à disparaître quand la machine s'arrête. »

1. Trouve **fonction de rang** f (« quantité »)
2. Prouve que f a une **borne inférieure** (« disparaît quand la machine s'arrête »)

L'analyse de terminaison : La méthode classique

Turing 1949



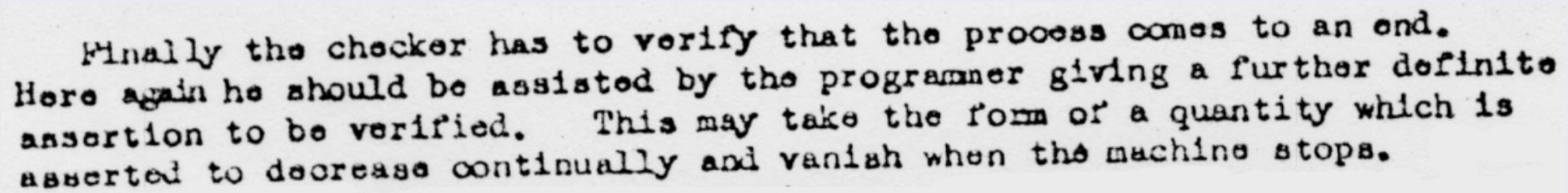
Finally the checker has to verify that the process comes to an end. Here again he should be assisted by the programmer giving a further definite assertion to be verified. This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops.

« Enfin le contrôleur doit vérifier que le processus arrive à une fin. [...] Cela peut prendre la forme d'une quantité qui est affirmée à diminuer continûment et à disparaître quand la machine s'arrête. »

1. Trouve **fonction de rang** f (« quantité »)
2. Prouve que f a une **borne inférieure** (« disparaît quand la machine s'arrête »)
3. Prouve que f **diminue** au fil du temps

L'analyse de terminaison : La méthode classique

Turing 1949



Finally the checker has to verify that the process comes to an end. Here again he should be assisted by the programmer giving a further definite assertion to be verified. This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops.

« Enfin le contrôleur doit vérifier que le processus arrive à une fin. [...] Cela peut prendre la forme d'une quantité qui est affirmée à diminuer continûment et à disparaître quand la machine s'arrête. »

1. Trouve **fonction de rang** f (« quantité »)
2. Prouve que f a une **borne inférieure** (« disparaît quand la machine s'arrête »)
3. Prouve que f **diminue** au fil du temps

Exemple (C'est facile, la terminaison)

while $x > 0$ **do**
    $x := x - 1$;
done

L'analyse de terminaison : Des approches automatiques

L'analyse de terminaison : Des approches automatiques

1. Approches monolithiques :
   - Fonctions de rang (FR) linéaires ou polynomiales
   - Size-change termination

L'analyse de terminaison : Des approches automatiques

1. Approches monolithiques :
   - Fonctions de rang (FR) linéaires ou polynomiales
   - Size-change termination
2. Simplification itérative :
   - Combinaisons lexicographiques
     $\rightarrow$ RANK [Alias, Darte, Feautrier, Gonnord, *équipe Compsys, Lyon*]
   - Dependency Pair Framework (de la réécriture des termes)
     $\rightarrow$ AProVE, CiME, Matchbox, Mu-Term, TTT2, VMTL, ...
   - pour : On peut réutiliser 1
   - pour : Chaque pas avance la preuve (efface une partie du problème)
   - contre : Ignore le point de départ du programme, pas de génération d'invariants

L'analyse de terminaison : Des approches automatiques

1. Approches monolithiques :
   - Fonctions de rang (FR) linéaires ou polynomiales
   - Size-change termination
2. Simplification itérative :
   - Combinaisons lexicographiques
     $\rightarrow$ RANK [Alias, Darte, Feautrier, Gonnord, *équipe Compsys, Lyon*]
   - Dependency Pair Framework (de la réécriture des termes)
     $\rightarrow$ AProVE, C*i*ME, MATCHBOX, MU-TERM, TTT2, VMTL, ...
   - pour : On peut réutiliser 1
   - pour : Chaque pas avance la preuve (efface une partie du problème)
   - contre : Ignore le point de départ du programme, pas de génération d'invariants
3. Renforcement itératif de l'argument de la terminaison (à la CEGAR) :
   - Invariants de transition $\rightarrow$ TERMINATOR [Cook, Podelski, Rybalchenko]
   - Combinaisons lexicographiques $\rightarrow$ T2 [Cook et al.]
   - pour : On peut réutiliser 1
   - pour : Le prouveur de sûreté *(safety)* raisonne avec les invariants
   - contre : N'avance pas toujours rééllement

L'analyse de terminaison : Invariants et fonctions de rang

Exemple

```
y := 1;
while x > 0 do
    x := x − y;
    y := y + 1;
done
```

- L'invariant $y > 0$ et la fonction de rang x prouvent la terminaison
- Comment sait-on qu'on a besoin de $y > 0$? $\curvearrowright$ x le requiert

L'analyse de terminaison : Invariants et fonctions de rang

Exemple

```
y := 1;
while x > 0 do
    x := x − y;
    y := y + 1;
done
```

- L'invariant $y > 0$ et la fonction de rang x prouvent la terminaison
- Comment sait-on qu'on a besoin de $y > 0$? $\curvearrowright$ x le requiert
- Comment sait-on que x est une fonction de rang ?

  $\curvearrowright$ $y > 0$ le prouve

Terminaison par renforcement itératif : L'idée

On codifie la vérification de l'argument (présumé) de terminaison comme un problème de sûreté (inatteignabilité)

Terminaison par renforcement itératif : L'idée

On codifie la vérification de l'argument (présumé) de terminaison comme un problème de sûreté (inatteignabilité)

1. Prouveur de sûreté : Fournit des échantillons (contre-exemples)
2. Outil de rang : Trouve un argument de terminaison **spécifique**
3. Proveur de sûreté : Prouve la **généralité**, ou 1

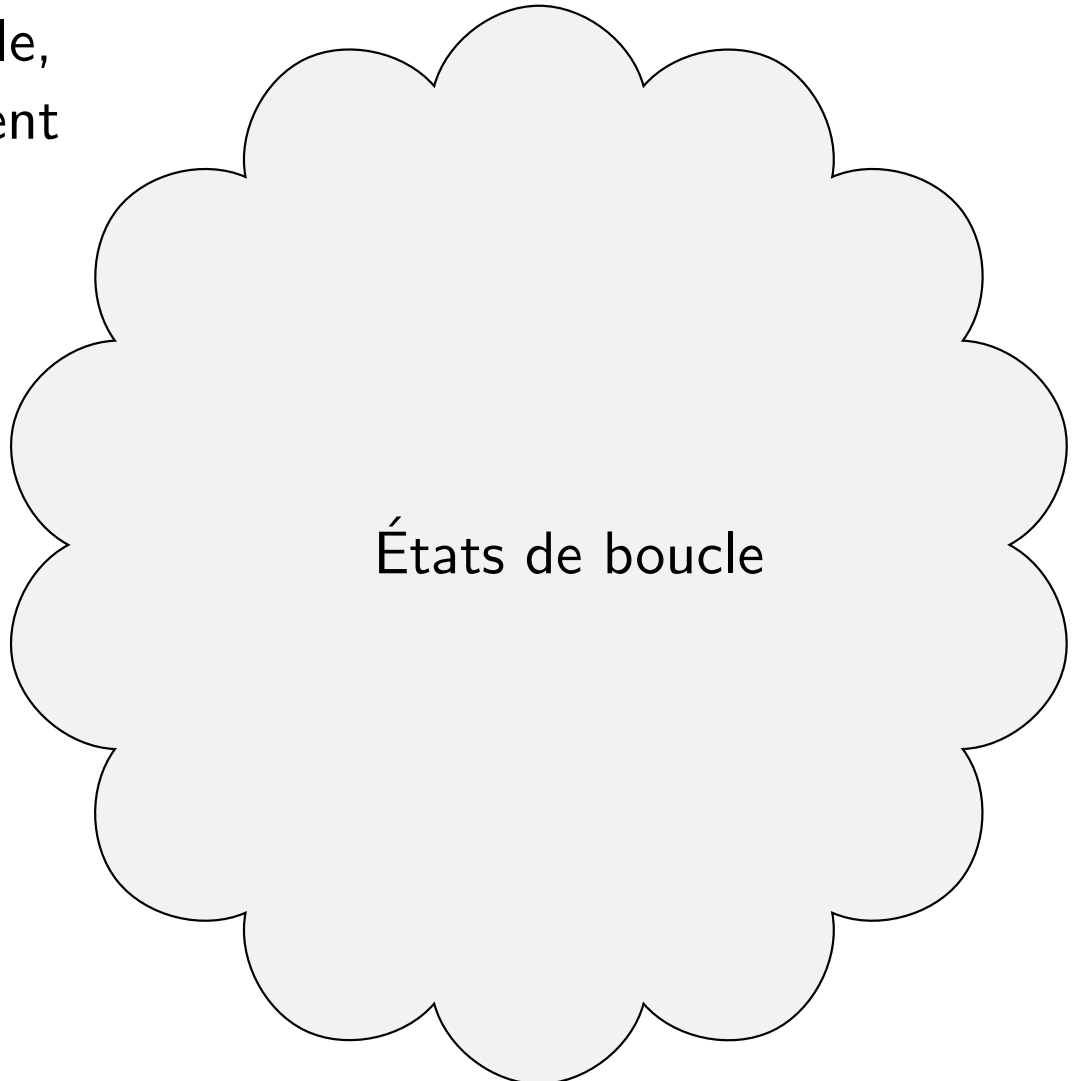
Terminaison par renforcement itératif : L'idée

On codifie la vérification de l'argument (présumé) de terminaison comme un problème de sûreté (inatteignabilité)

1. Prouveur de sûreté : Fournit des échantillons (contre-exemples)
2. Outil de rang : Trouve un argument de terminaison **spécifique**
3. Proveur de sûreté : Prouve la **généralité**, ou 1

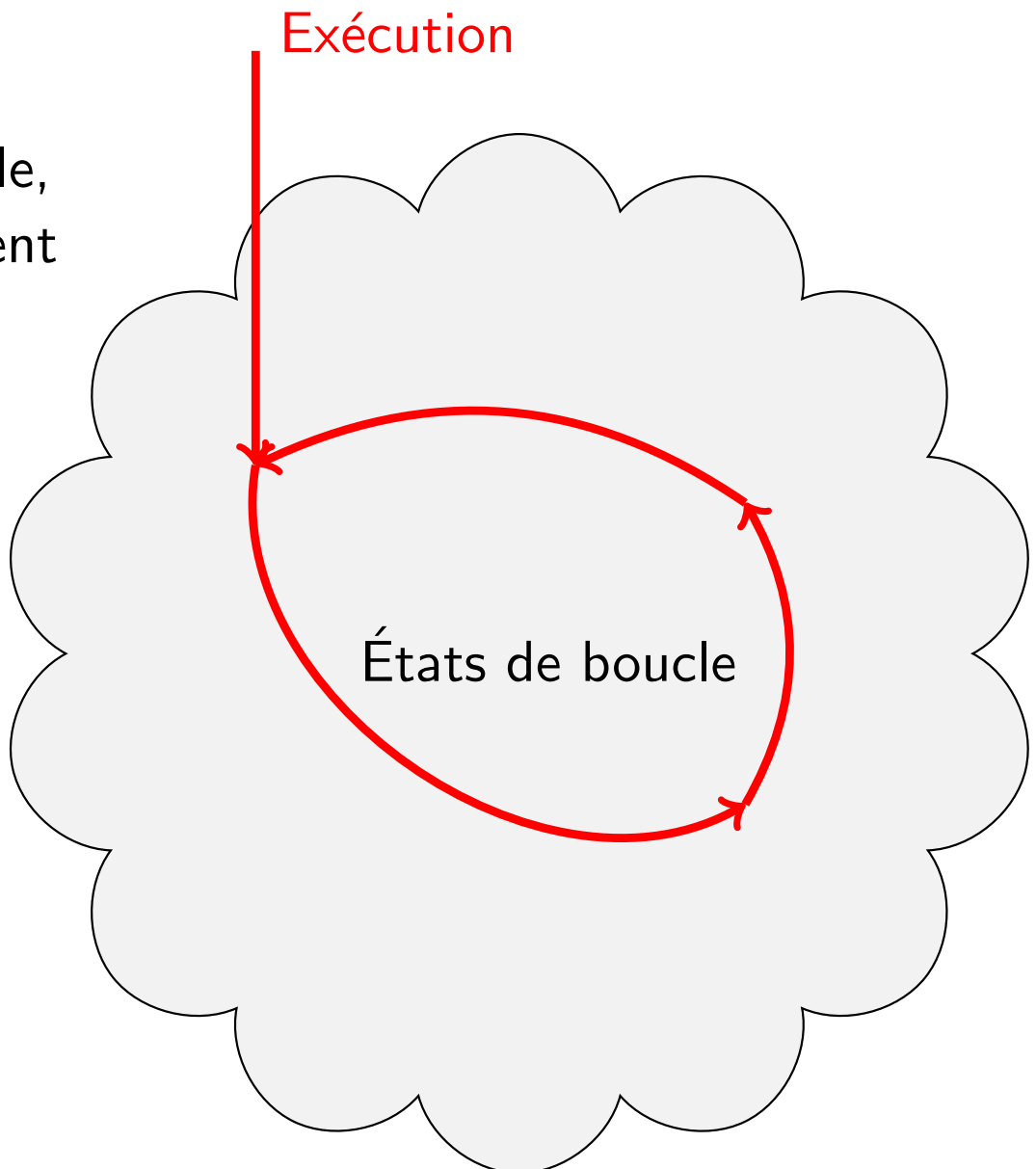
Terminaison par renforcement itératif

Trouve contre-exemple,
donc renforce argument



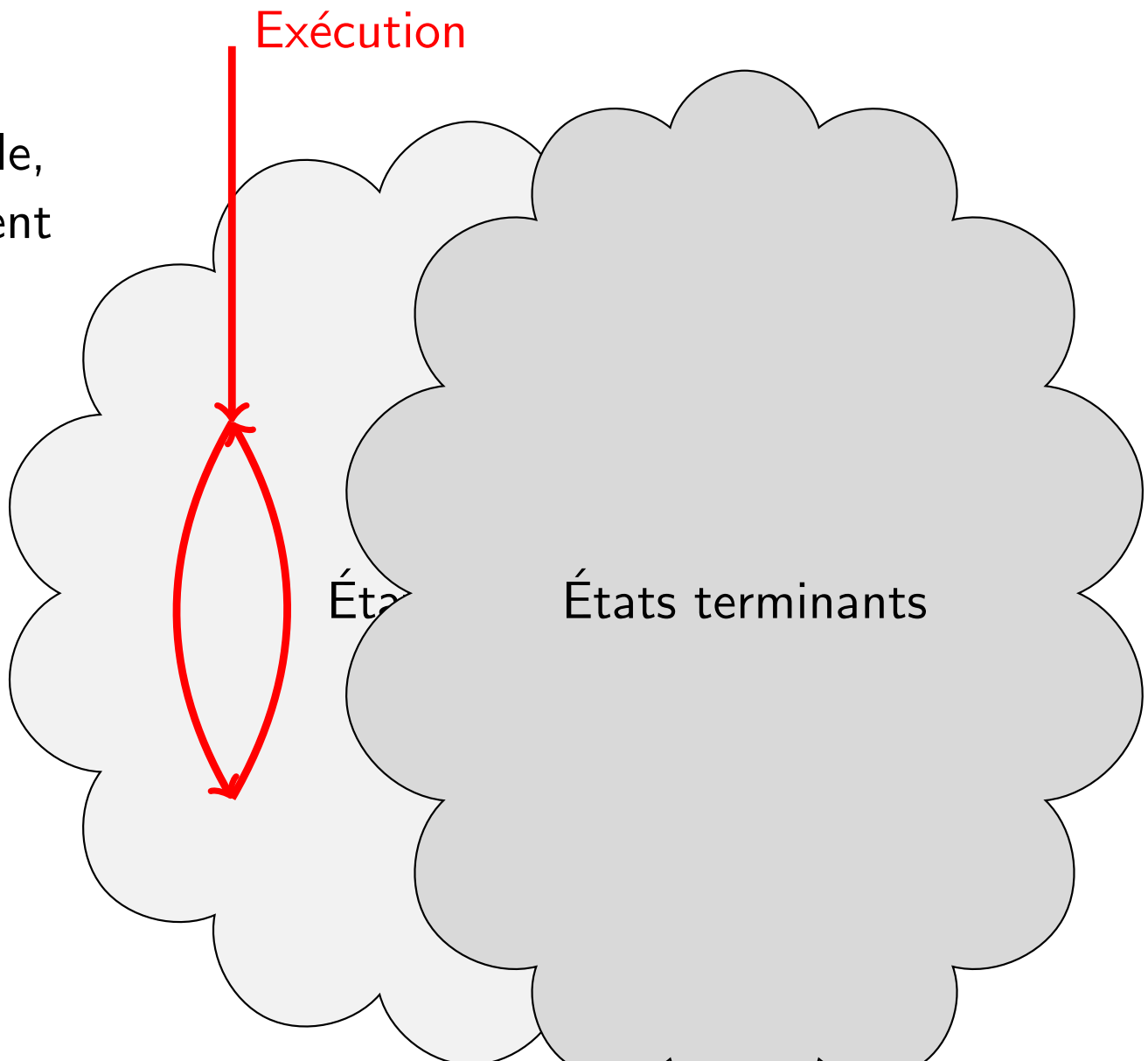
Terminaison par renforcement itératif

Trouve contre-exemple,
donc renforce argument



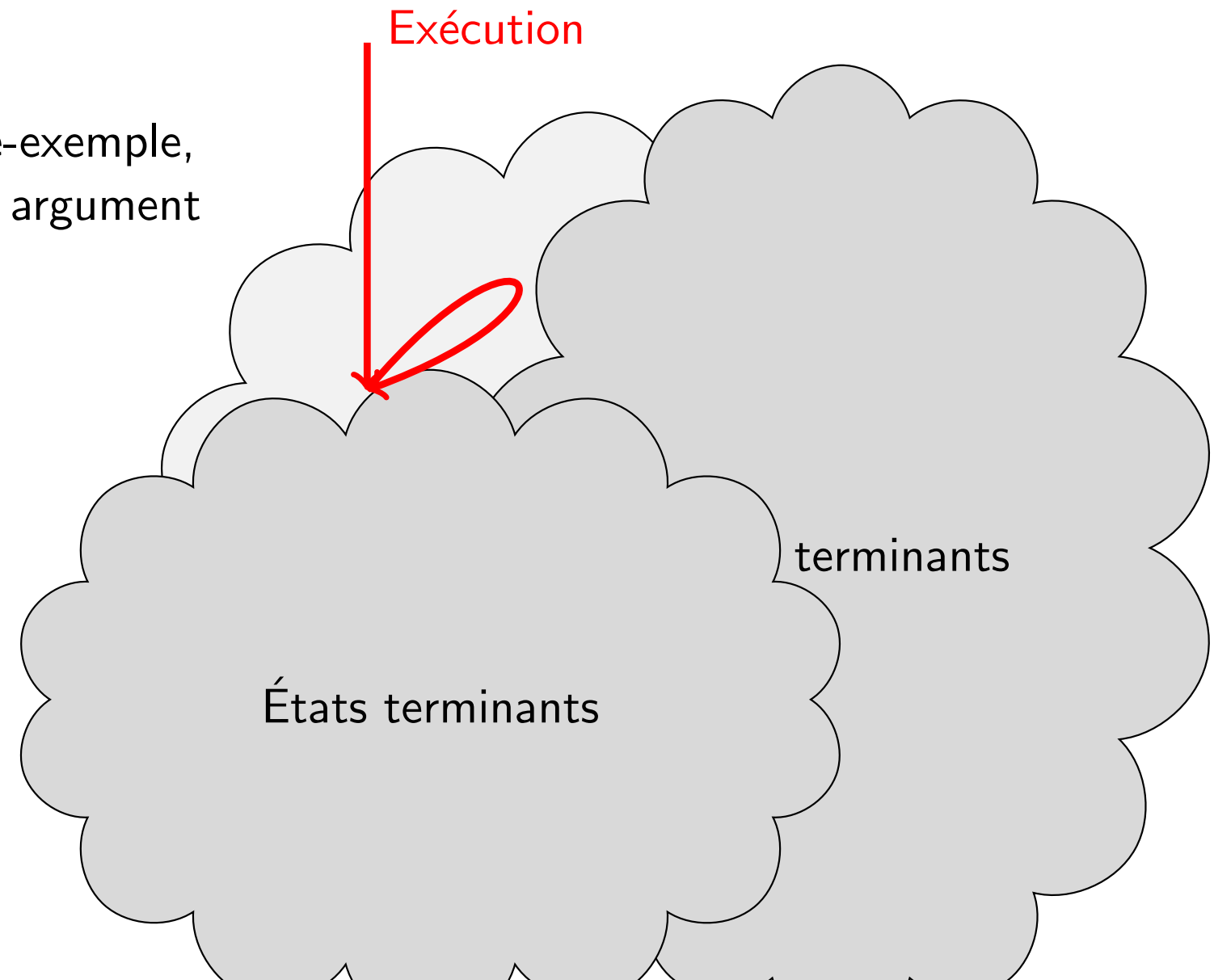
Terminaison par renforcement itératif

Trouve contre-exemple,
donc renforce argument



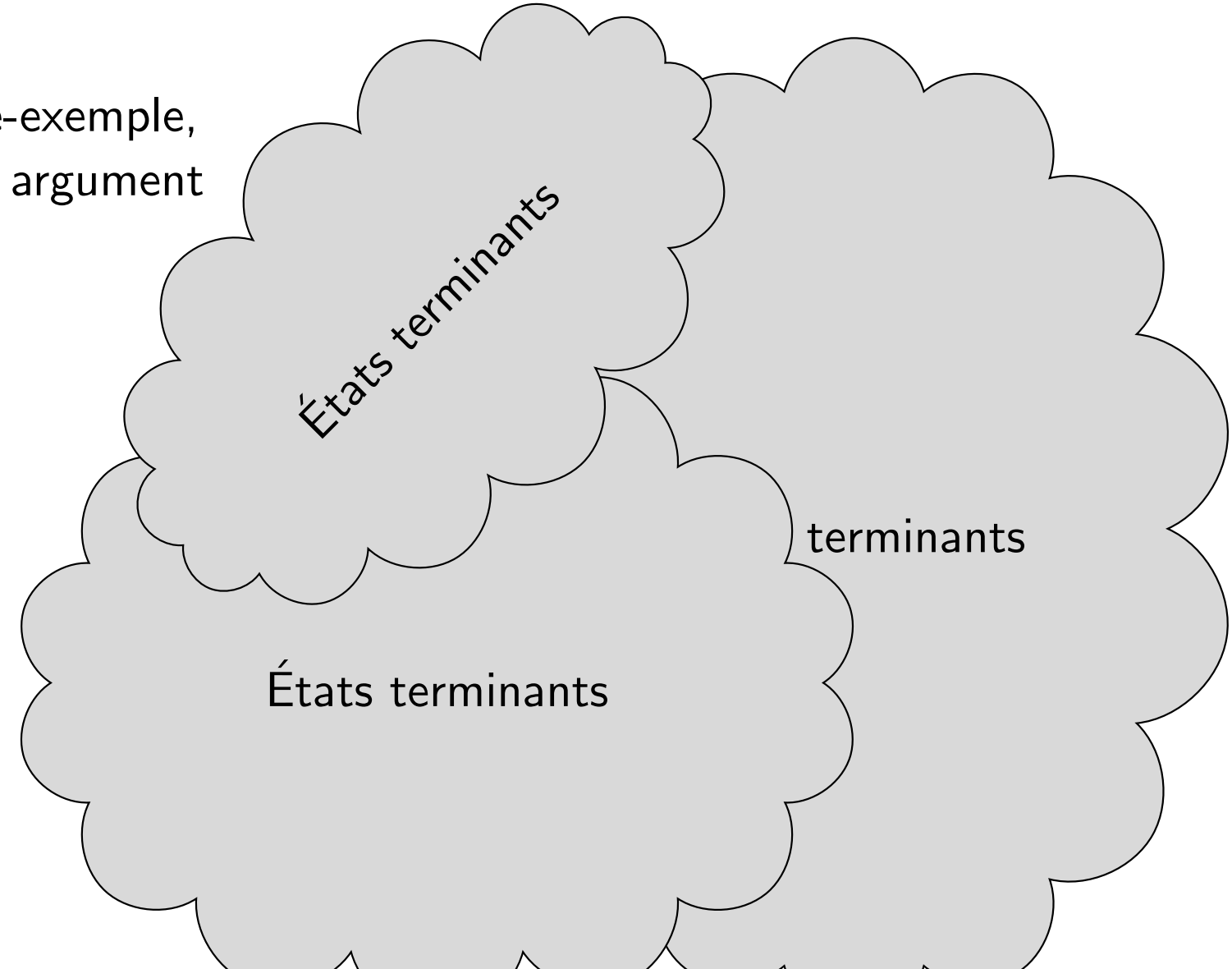
Terminaison par renforcement itératif

Trouve contre-exemple,
donc renforce argument



Terminaison par renforcement itératif

Trouve contre-exemple,
donc renforce argument



Terminaison par renforcement : Le pire cas

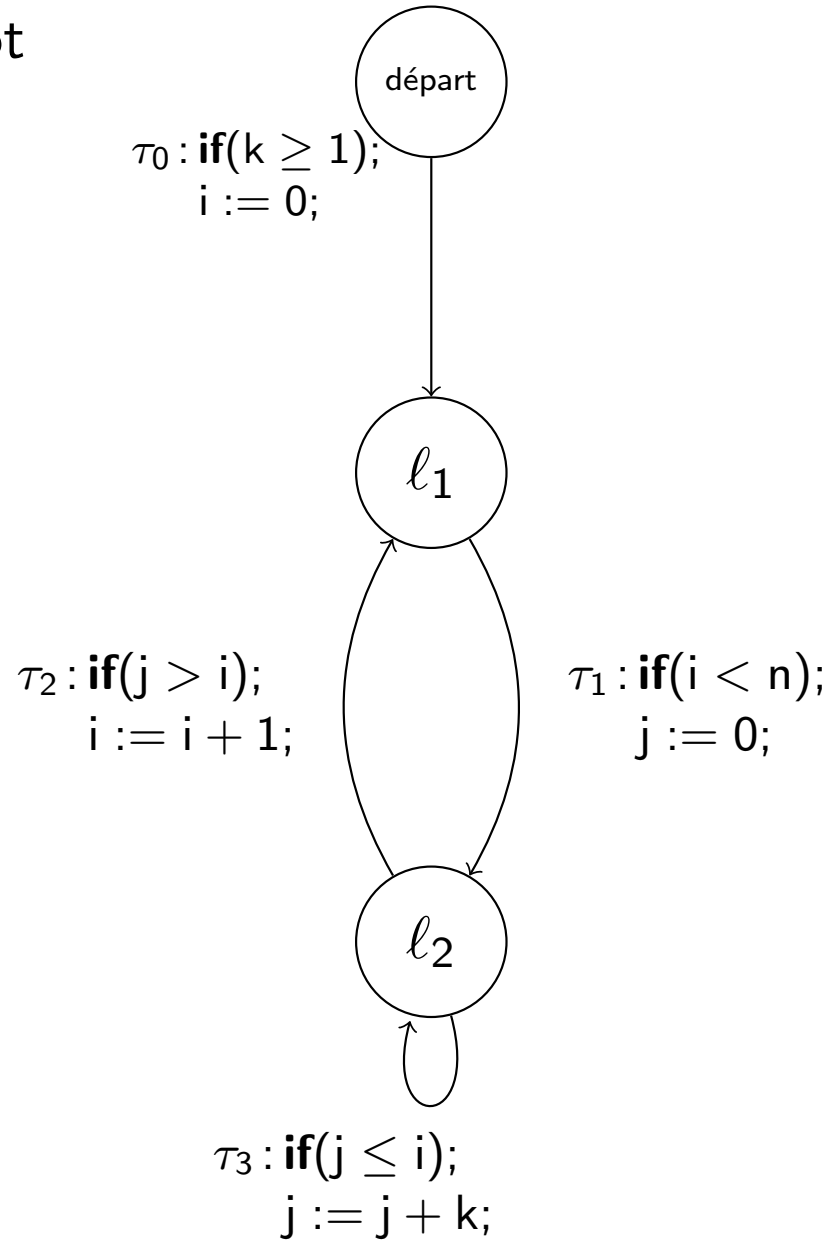
1. Prouveur de sûreté :
   Examine tout ... et retourne essentiellement l'échantillon antérieur
2. Outil de rang : Trouve argument de terminaison **trop** spécifique
3. Prouveur de sûreté : Ne peut pas prouver la généralité, répète 1

Terminaison par renforcement : Le pire cas

1. Prouveur de sûreté :
   Examine tout ... et retourne essentiellement l'échantillon antérieur
2. Outil de rang : Trouve argument de terminaison **trop** spécifique
3. Prouveur de sûreté : Ne peut pas prouver la généralité, répète 1

Des boucles imbriquées

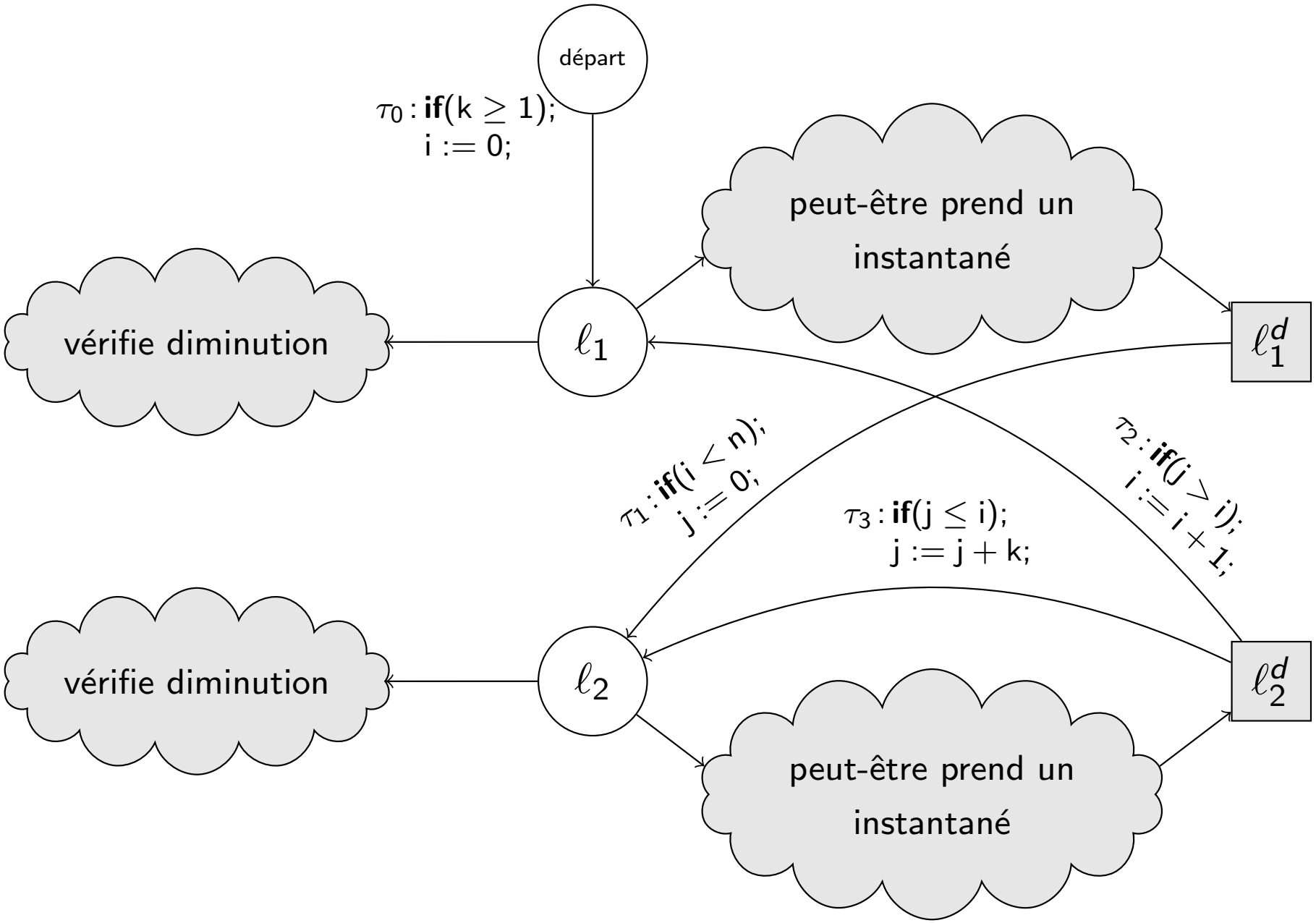
Graphe de flot de contrôle :



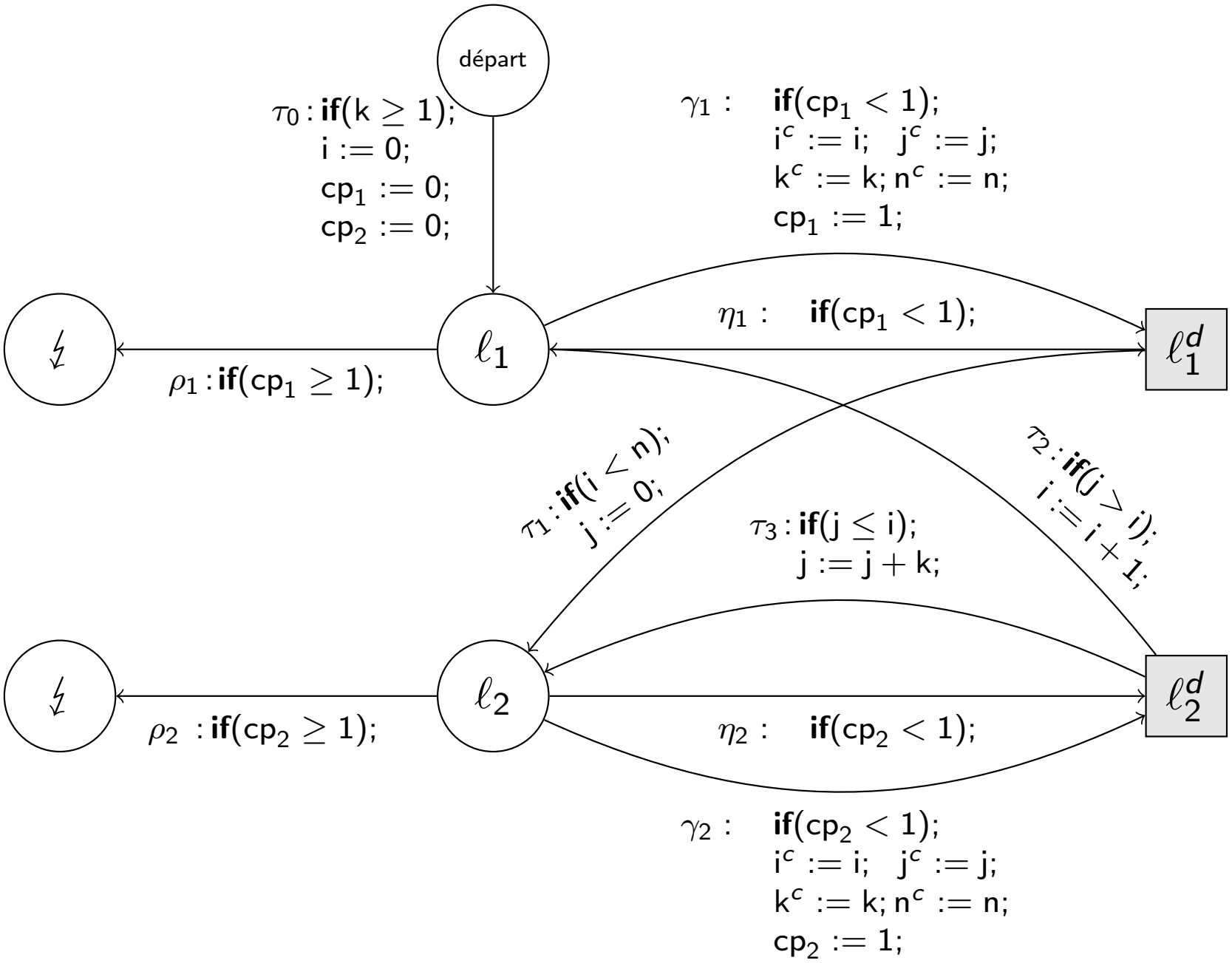
Exemple (Source)

```
   if k ≥ 1 then
      i := 0 ;
ℓ1 while i < n do
      j := 0 ;

ℓ2    while j ≤ i do
         j := j + k ;
      done

      i := i + 1 ;
   done
fi
```

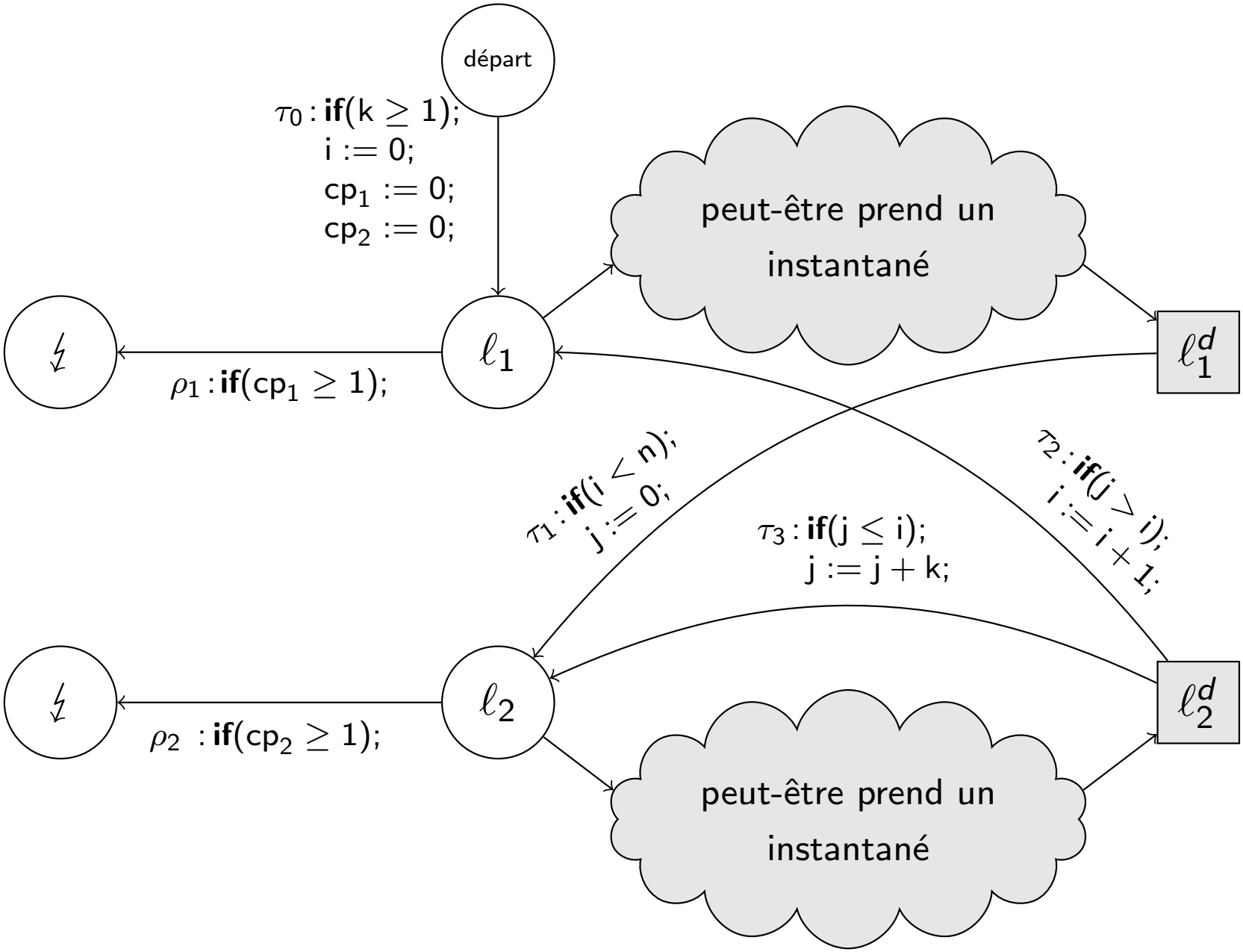
Des boucles imbriquées : Instrumentation



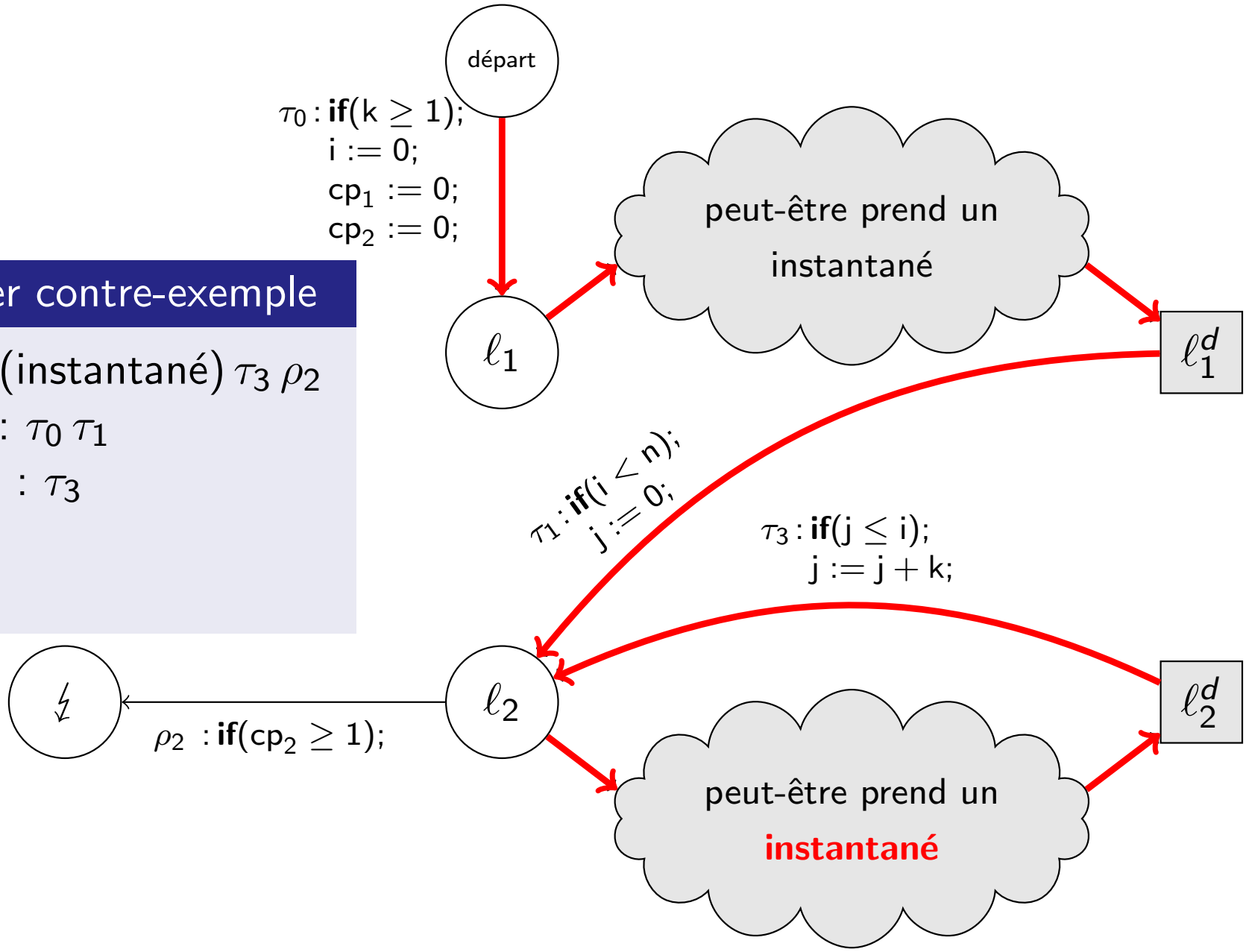
Des boucles imbriquées : Instrumentation



Des boucles imbriquées : Instrumentation



Des boucles imbriquées : Renforcement



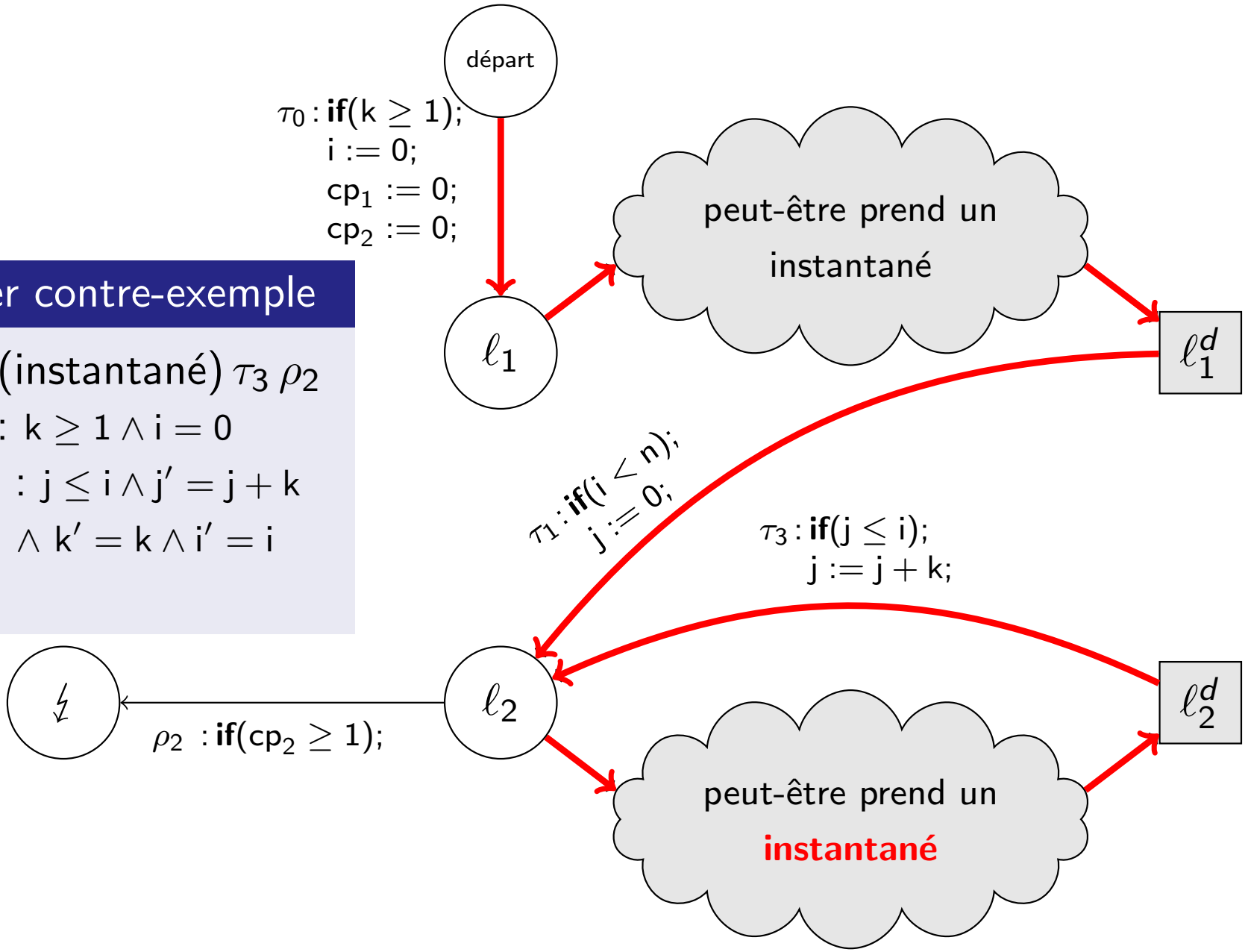
Premier contre-exemple

$\tau_0\ \tau_1$ (instantané) $\tau_3\ \rho_2$

Tronc : $\tau_0\ \tau_1$

Boucle : τ_3

Des boucles imbriquées : Renforcement



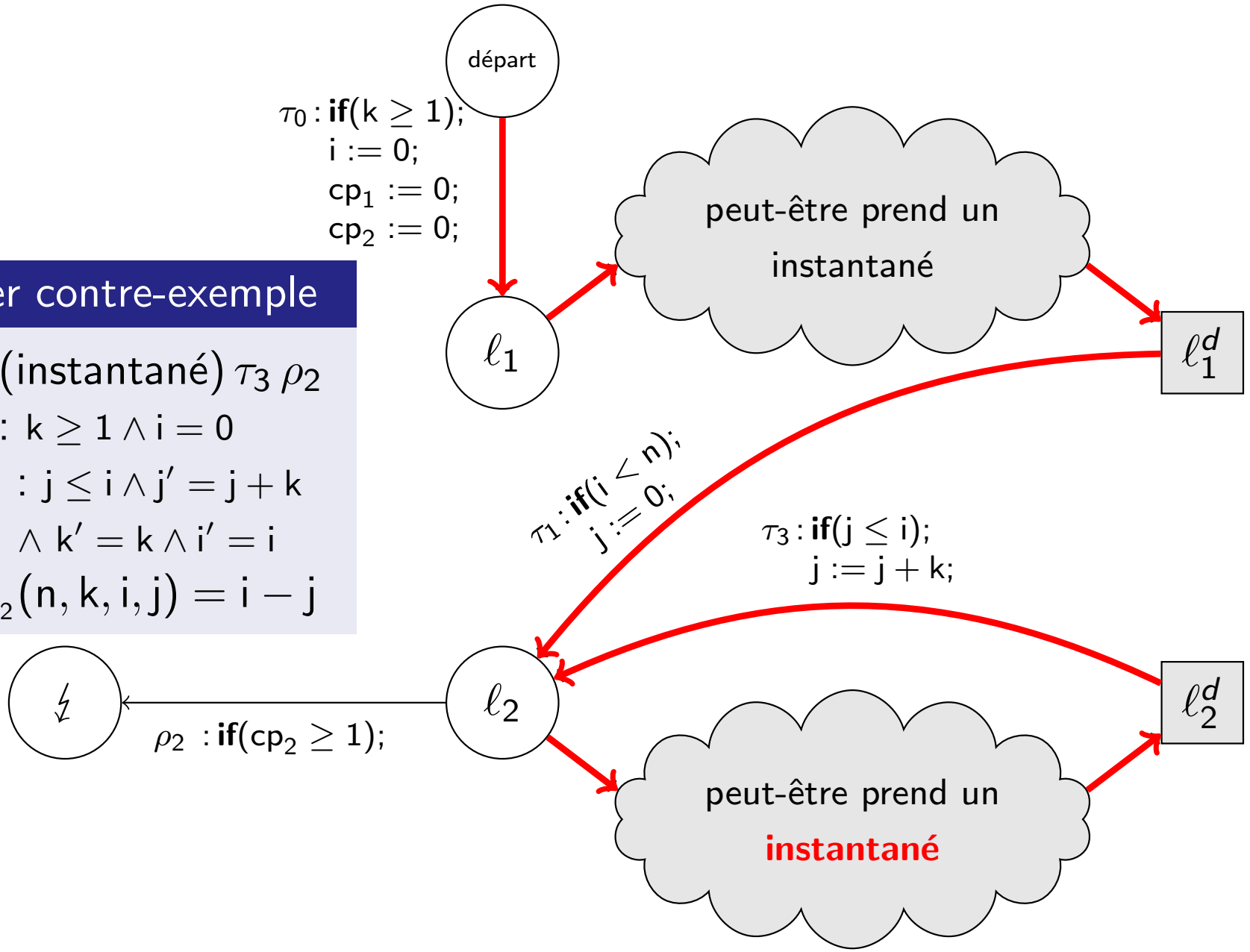
Premier contre-exemple

$\tau_0\ \tau_1$ (instantané) $\tau_3\ \rho_2$

Tronc : $k \geq 1 \wedge i = 0$

Boucle : $j \leq i \wedge j' = j + k$
$\wedge\ k' = k \wedge i' = i$

Des boucles imbriquées : Renforcement



Premier contre-exemple

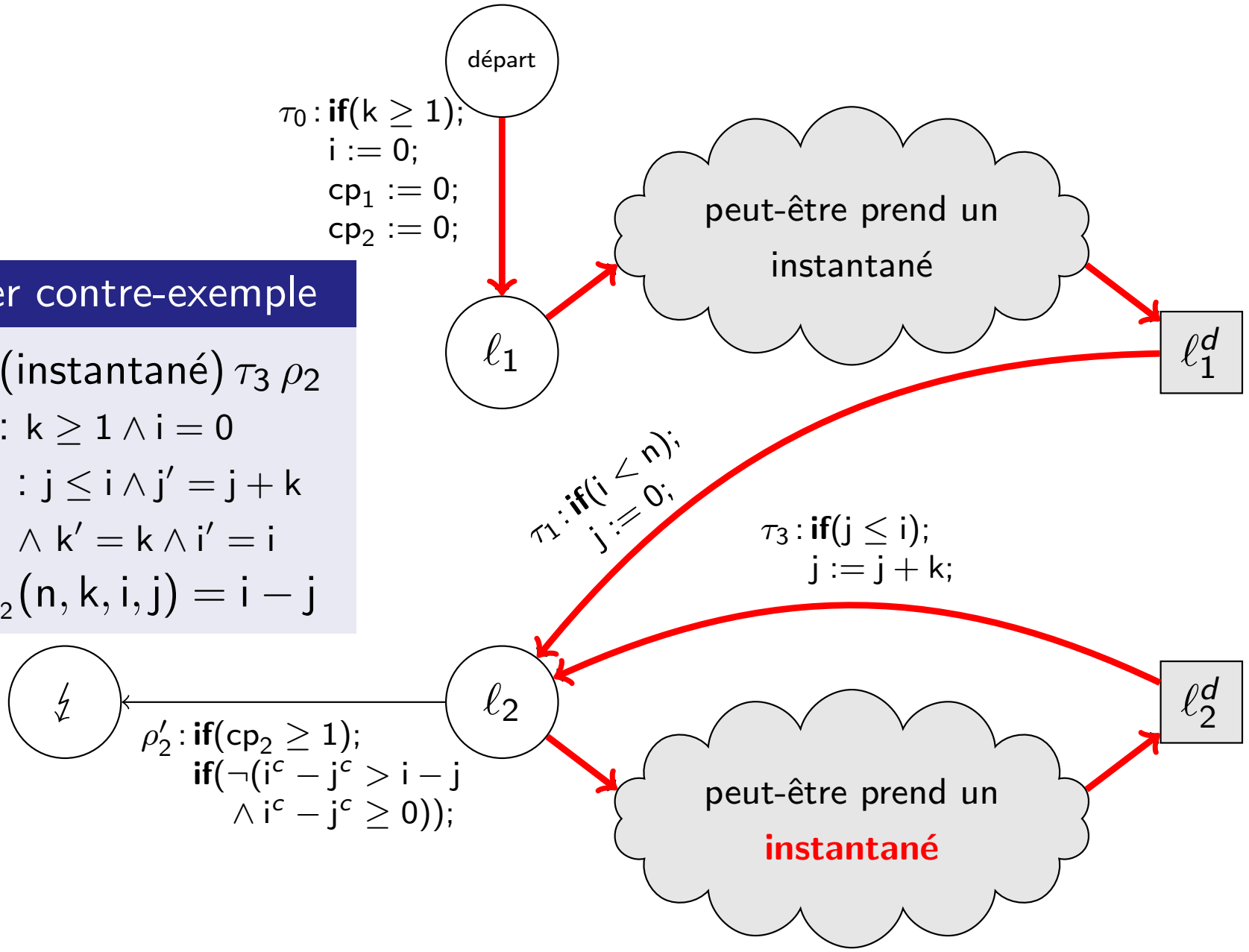
$\tau_0\ \tau_1$ (instantané) $\tau_3\ \rho_2$

Tronc : $k \geq 1 \wedge i = 0$

Boucle : $j \leq i \wedge j' = j + k$
$\wedge\ k' = k \wedge i' = i$

FR : $f_{\ell_2}(n, k, i, j) = i - j$

Des boucles imbriquées : Renforcement



Premier contre-exemple

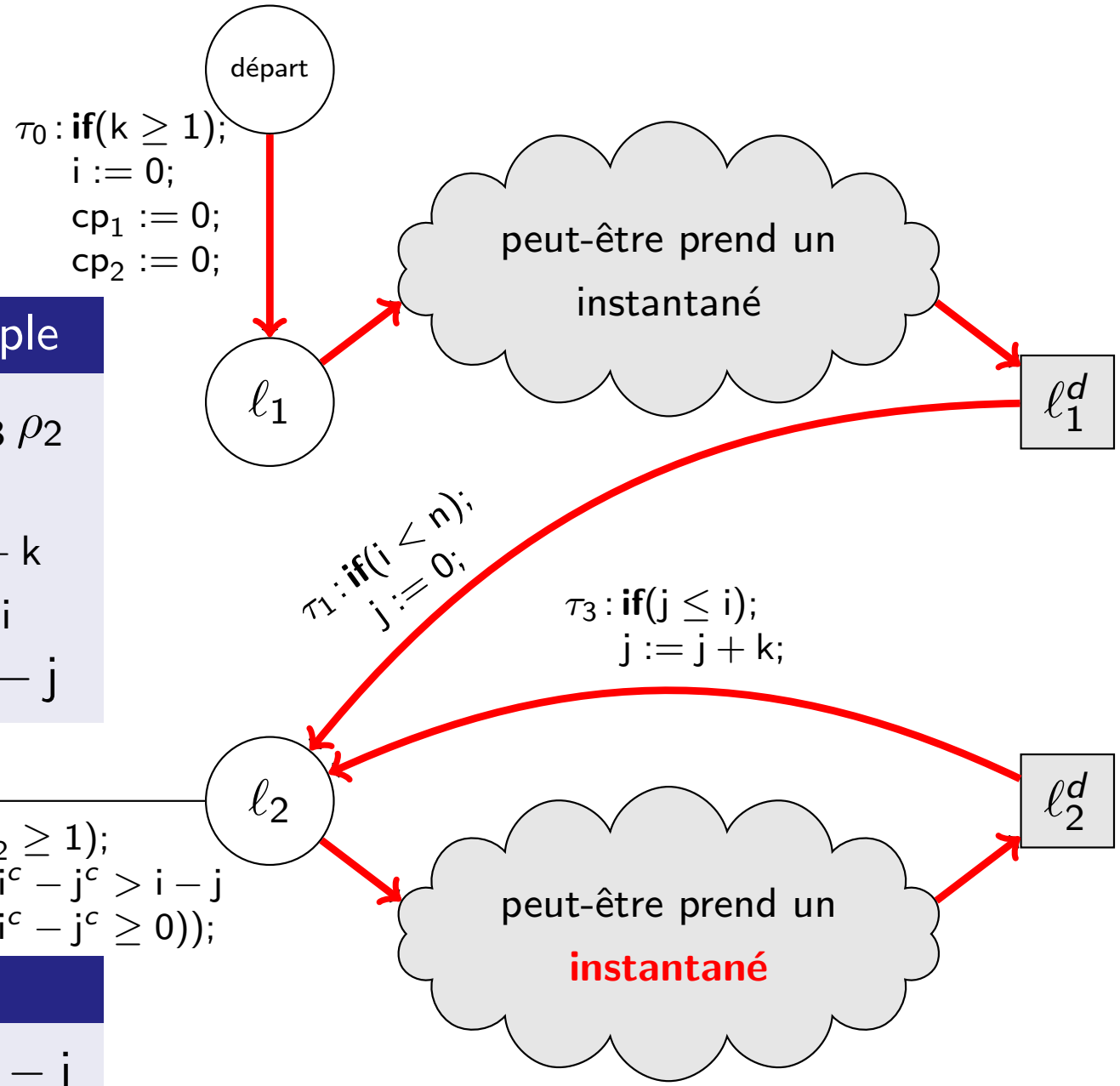
$\tau_0\ \tau_1$ (instantané) $\tau_3\ \rho_2$

Tronc : $k \geq 1 \wedge i = 0$

Boucle : $j \leq i \wedge j' = j + k$
$\wedge\ k' = k \wedge i' = i$

FR : $f_{\ell_2}(n, k, i, j) = i - j$

Des boucles imbriquées : Renforcement



Premier contre-exemple

$\tau_0\ \tau_1$ (instantané) $\tau_3\ \rho_2$

Tronc : $k \geq 1 \wedge i = 0$

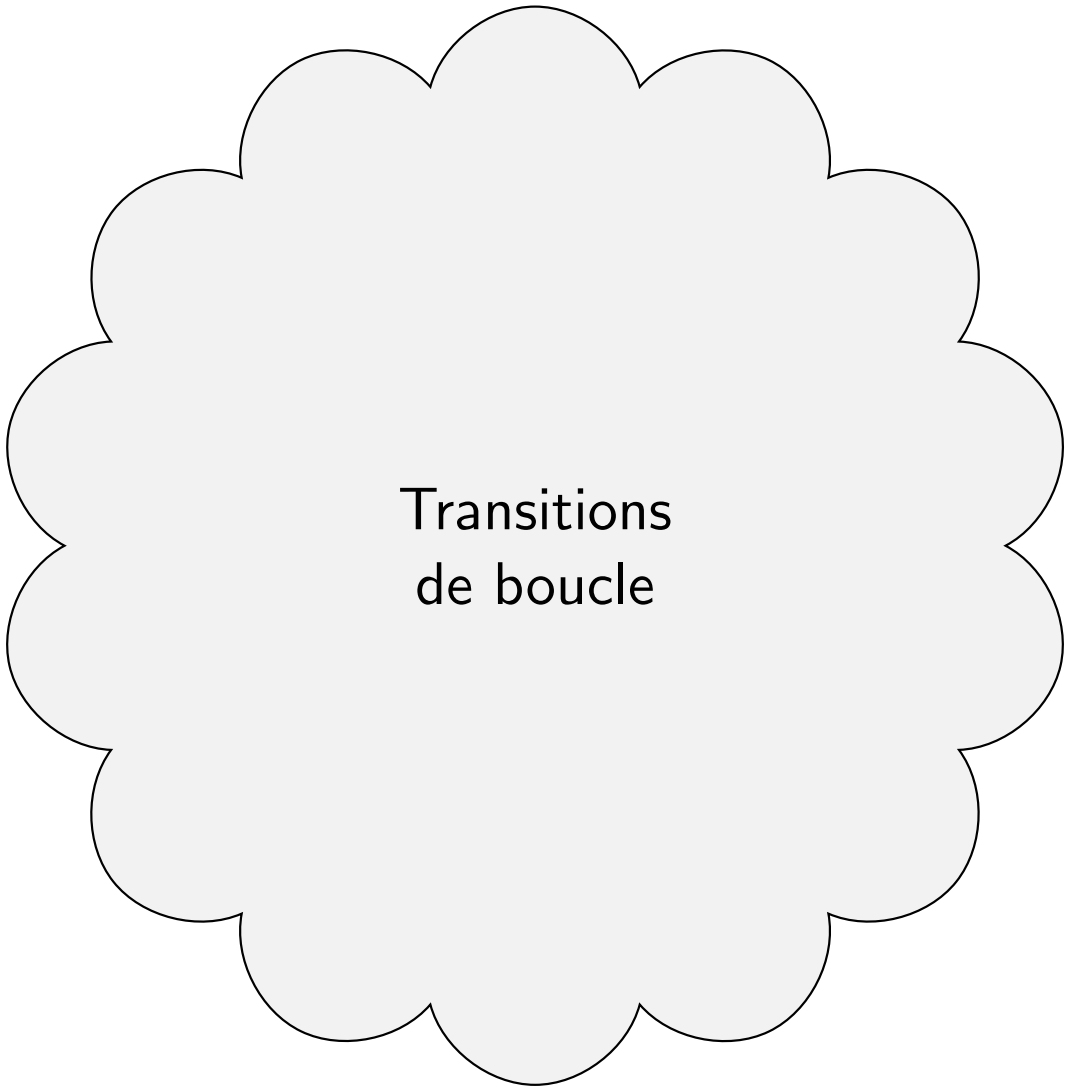
Boucle : $j \leq i \wedge j' = j + k$
$\wedge\ k' = k \wedge i' = i$

FR : $f_{\ell_2}(n, k, i, j) = i - j$

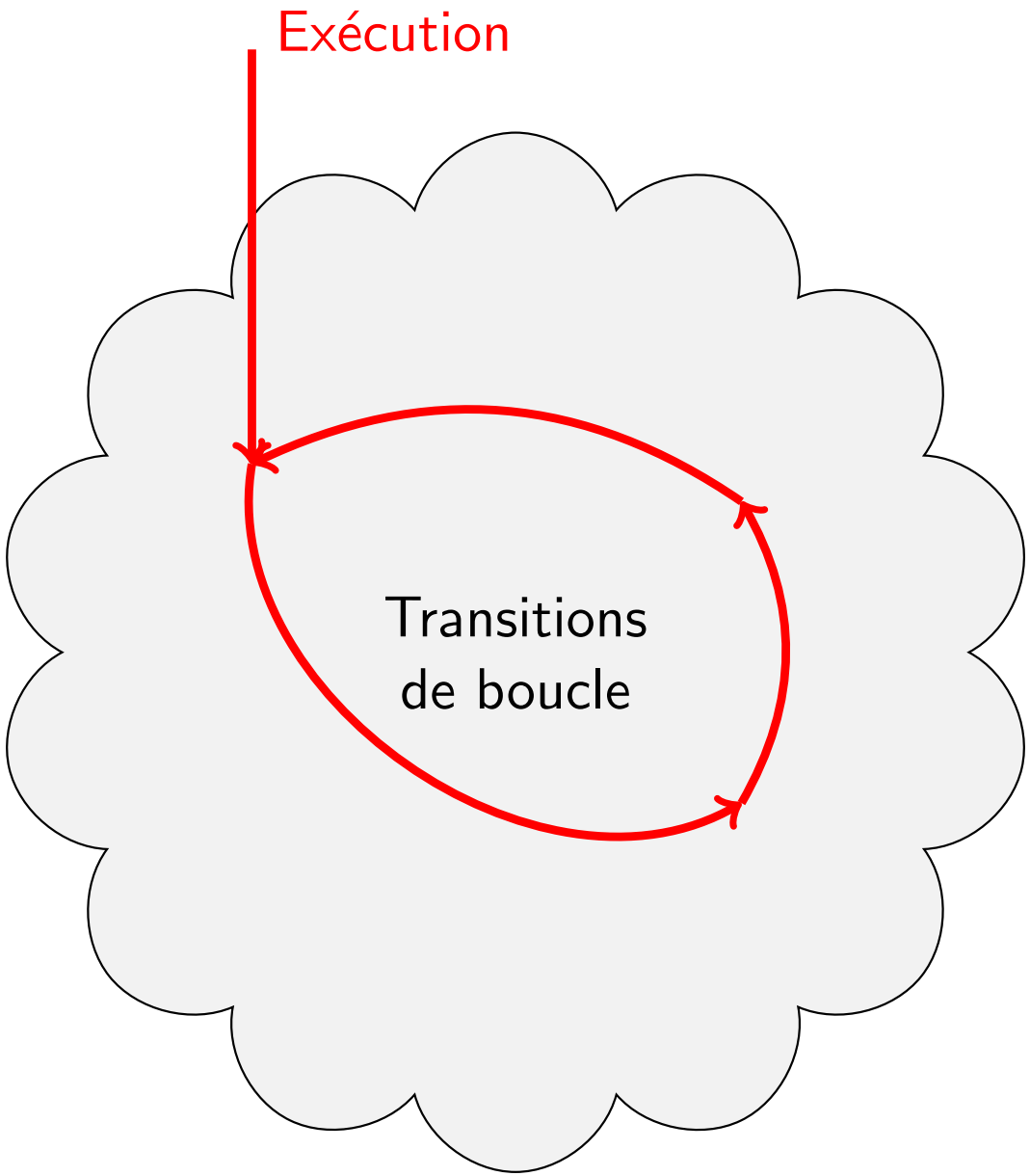
FR alternative

FR : $f'_{\ell_2}(n, k, i, j) = 1 - j$

Terminaison par simplification itérative

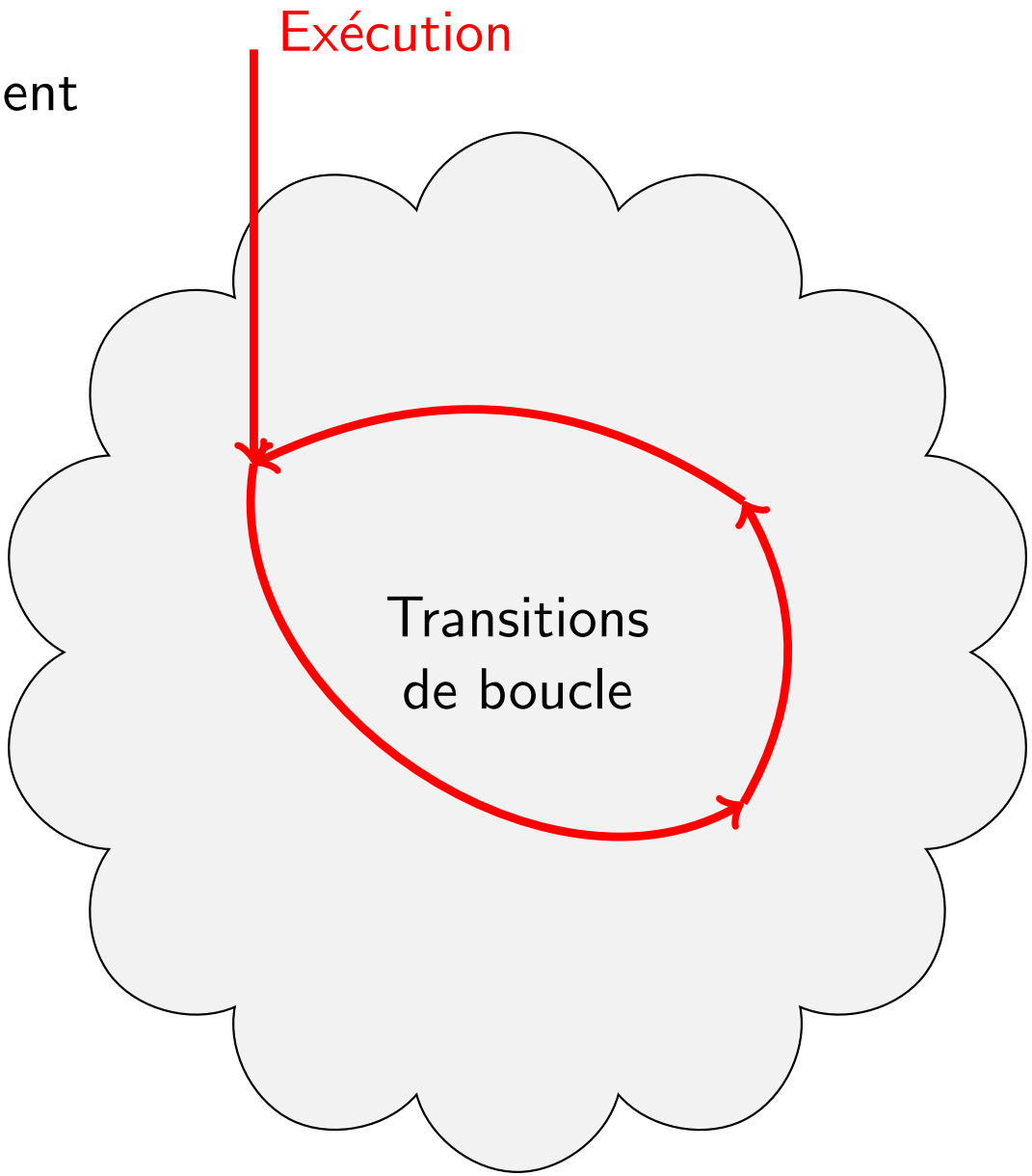


Terminaison par simplification itérative



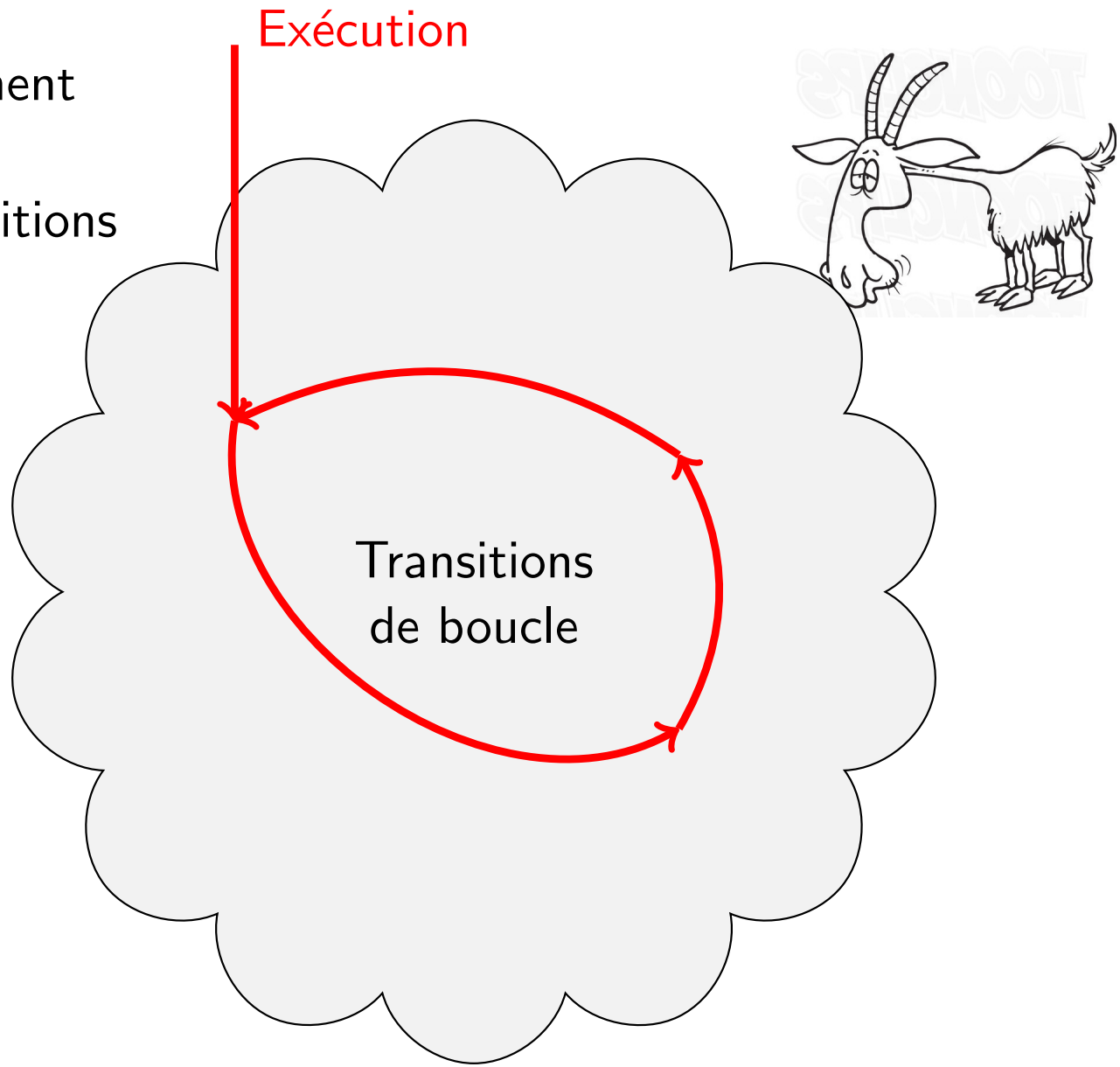
Terminaison par simplification itérative

Trouve fonction de rang pour composante fortement connexe (CFC),



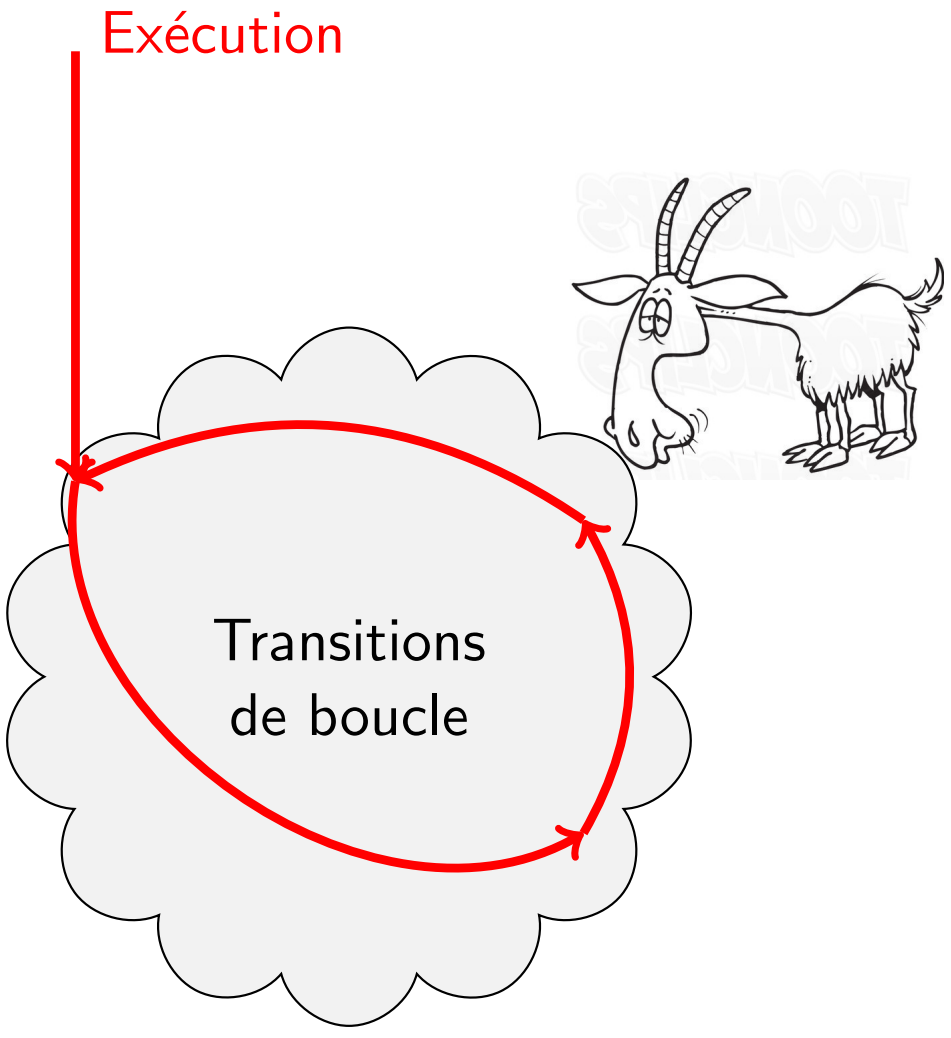
Terminaison par simplification itérative

Trouve fonction de rang
pour composante fortement
connexe (CFC),
donc supprime des transitions



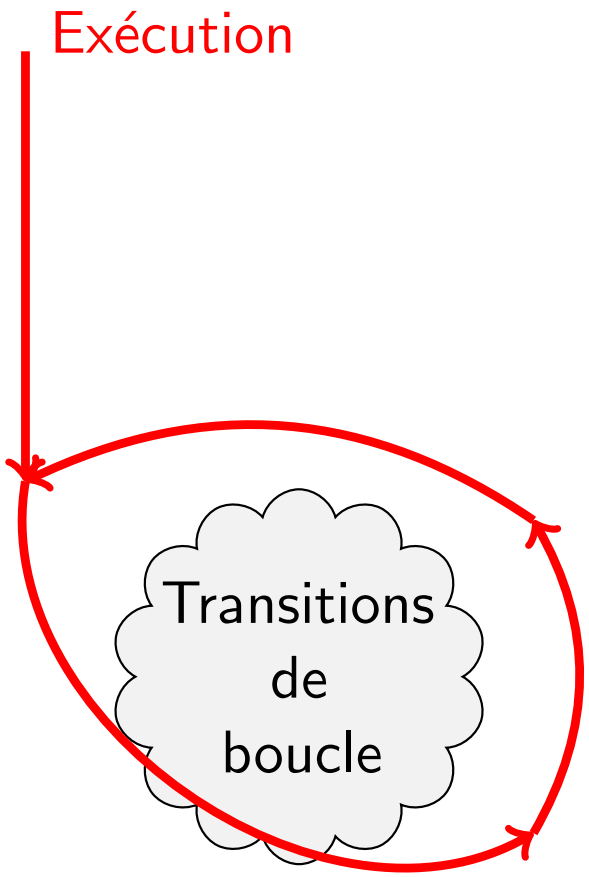
Terminaison par simplification itérative

Trouve fonction de rang
pour composante fortement
connexe (CFC),
donc supprime des transitions



Terminaison par simplification itérative

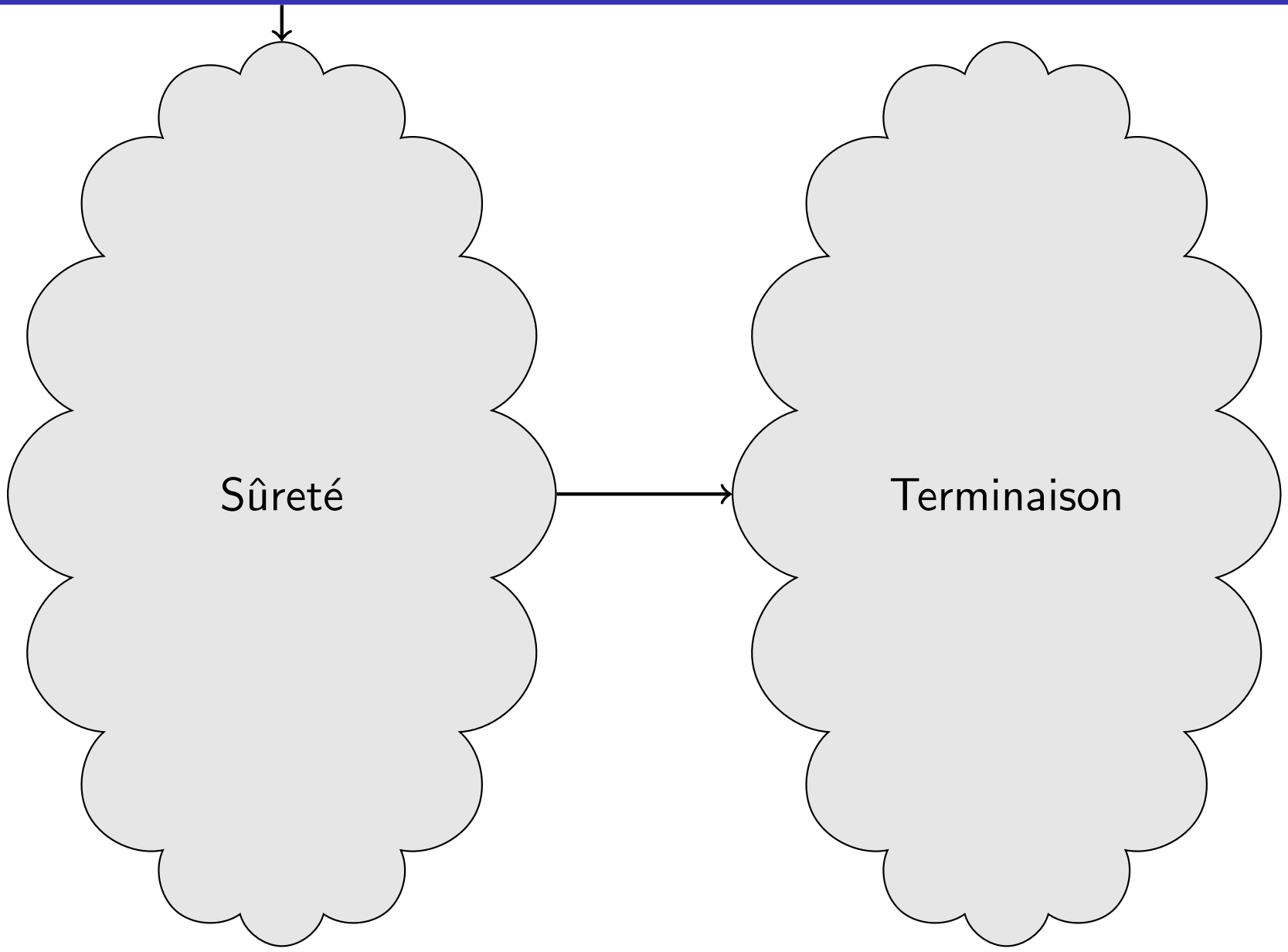
Trouve fonction de rang
pour composante fortement
connexe (CFC),
donc supprime des transitions



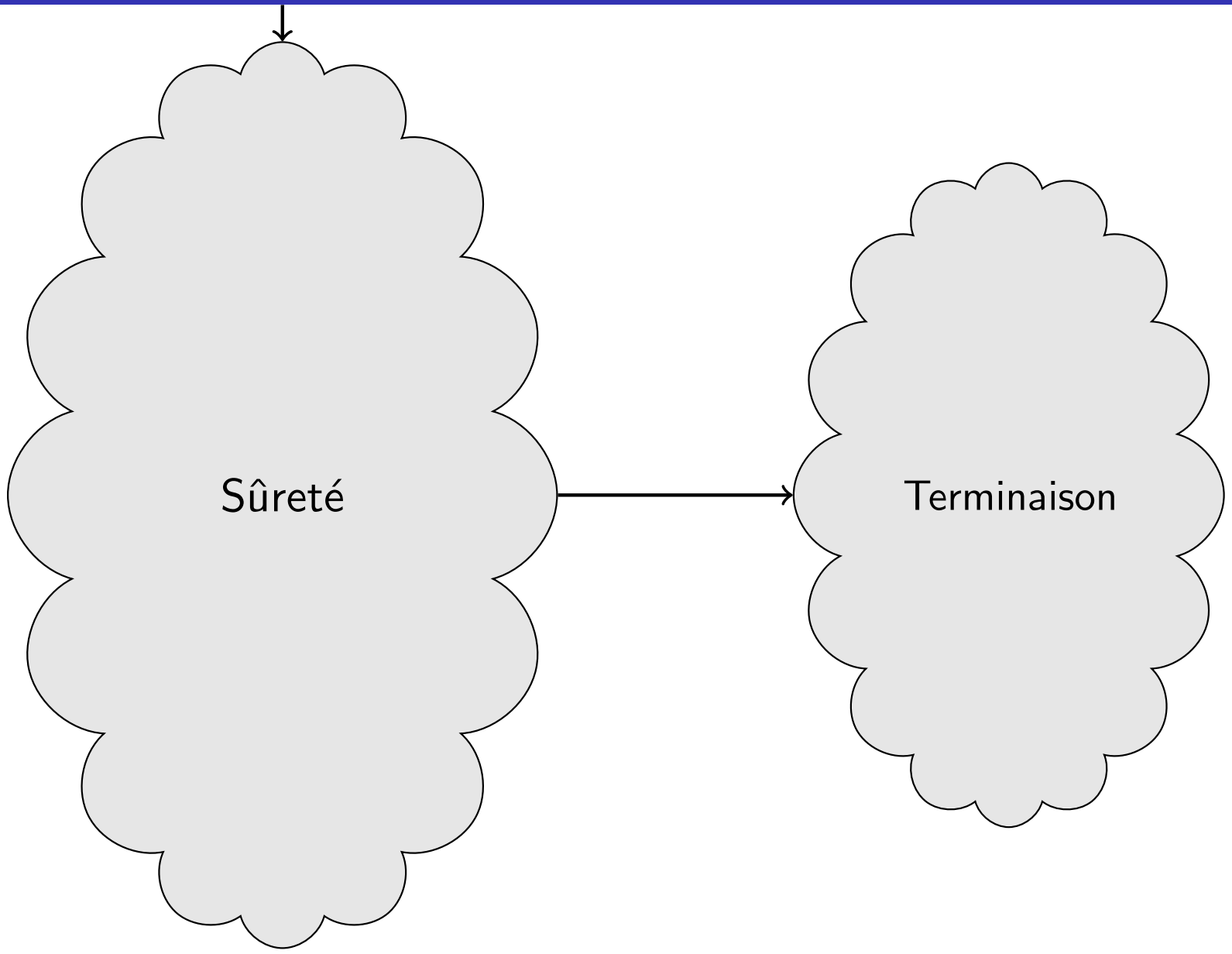
Terminaison par coopération

1. Prouveur de sûreté : Fournit des échantillons (contre-exemples)
2. Outil de rang :
   Trouve un argument de terminaison **dans le contexte**
3. Outil de rang : Marque des parties définitivement terminantes
4. Prouveur de sûreté : Prouve la généralité pour le reste, ou 1

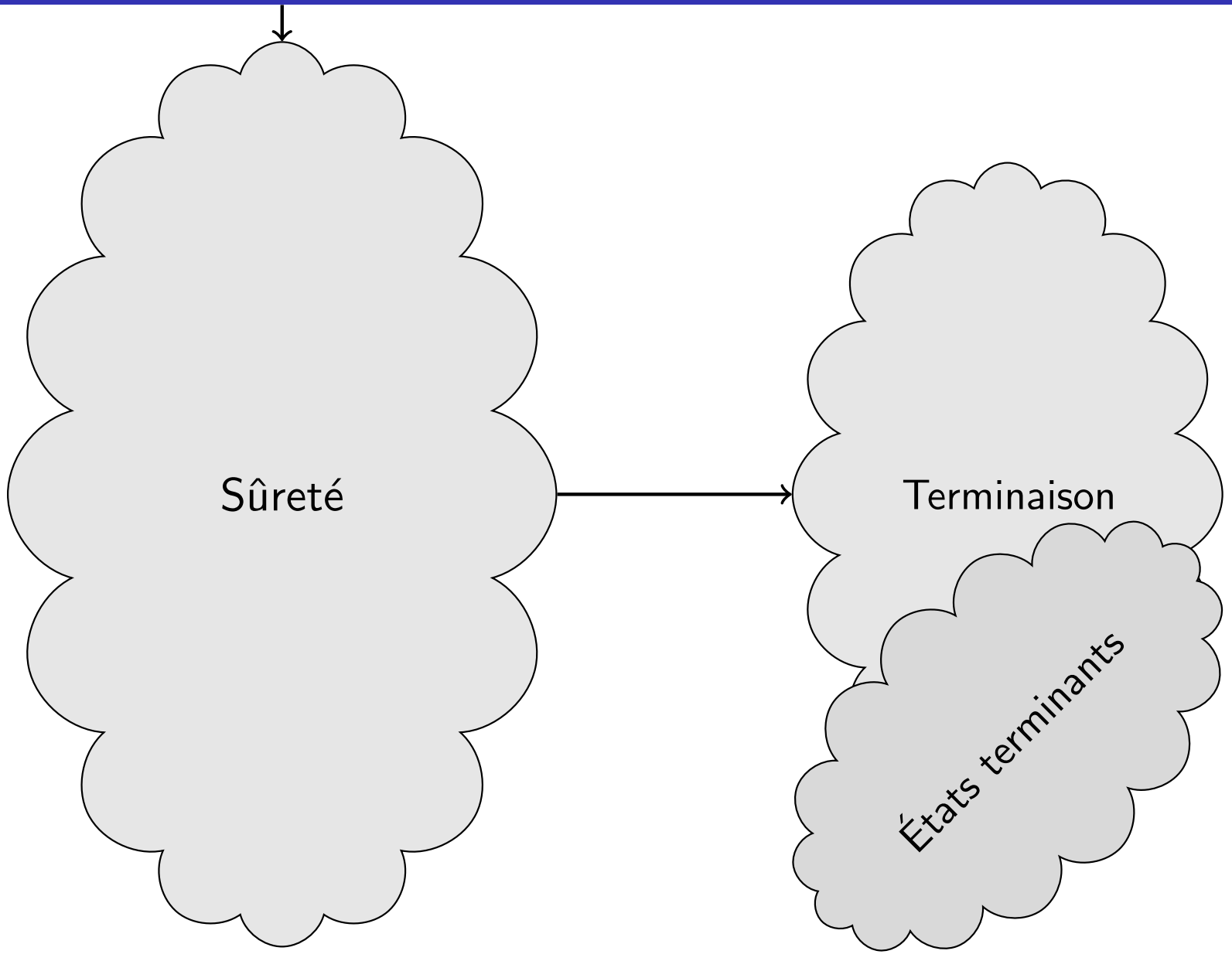
Coopération : Vue d'ensemble



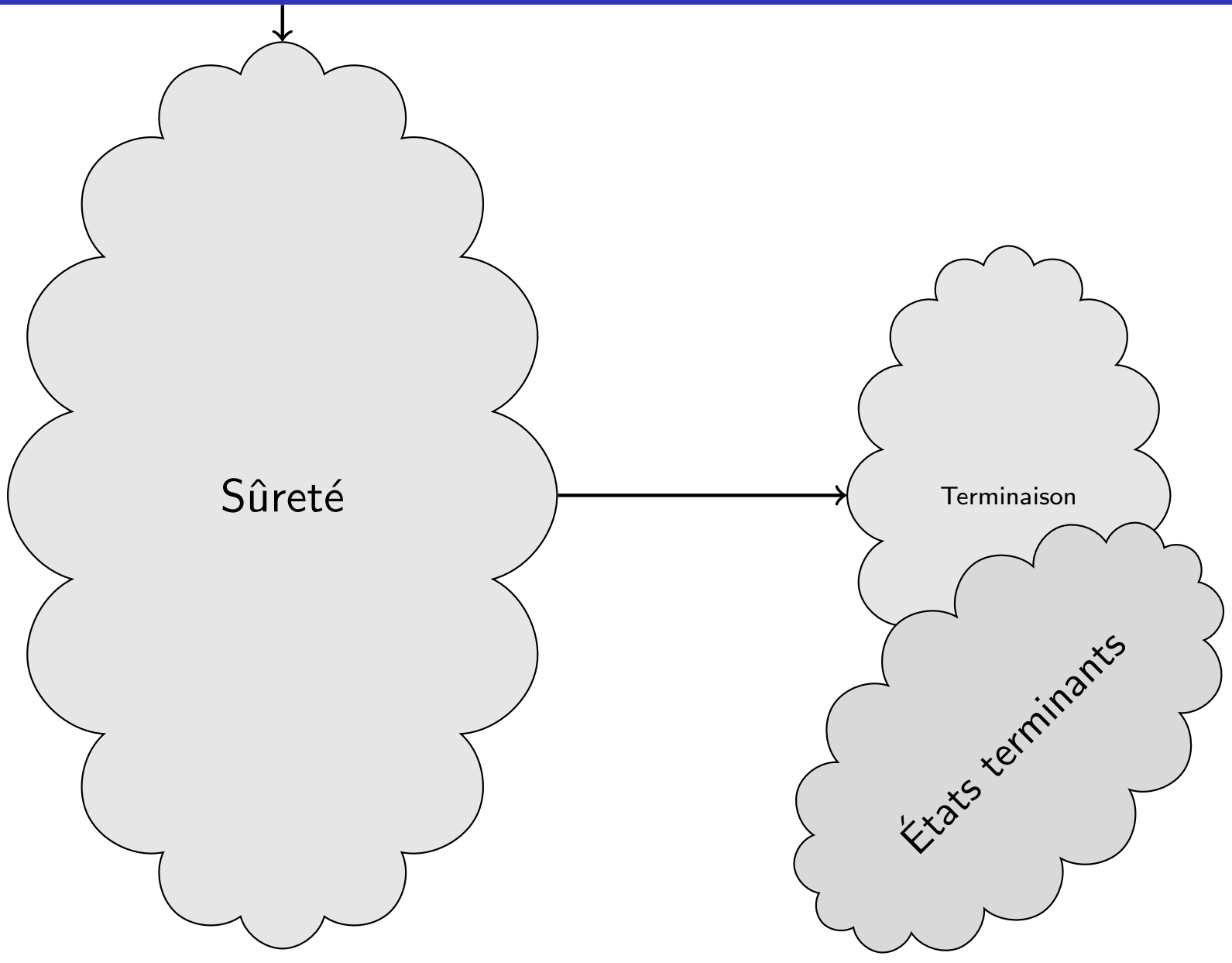
Coopération : Vue d'ensemble



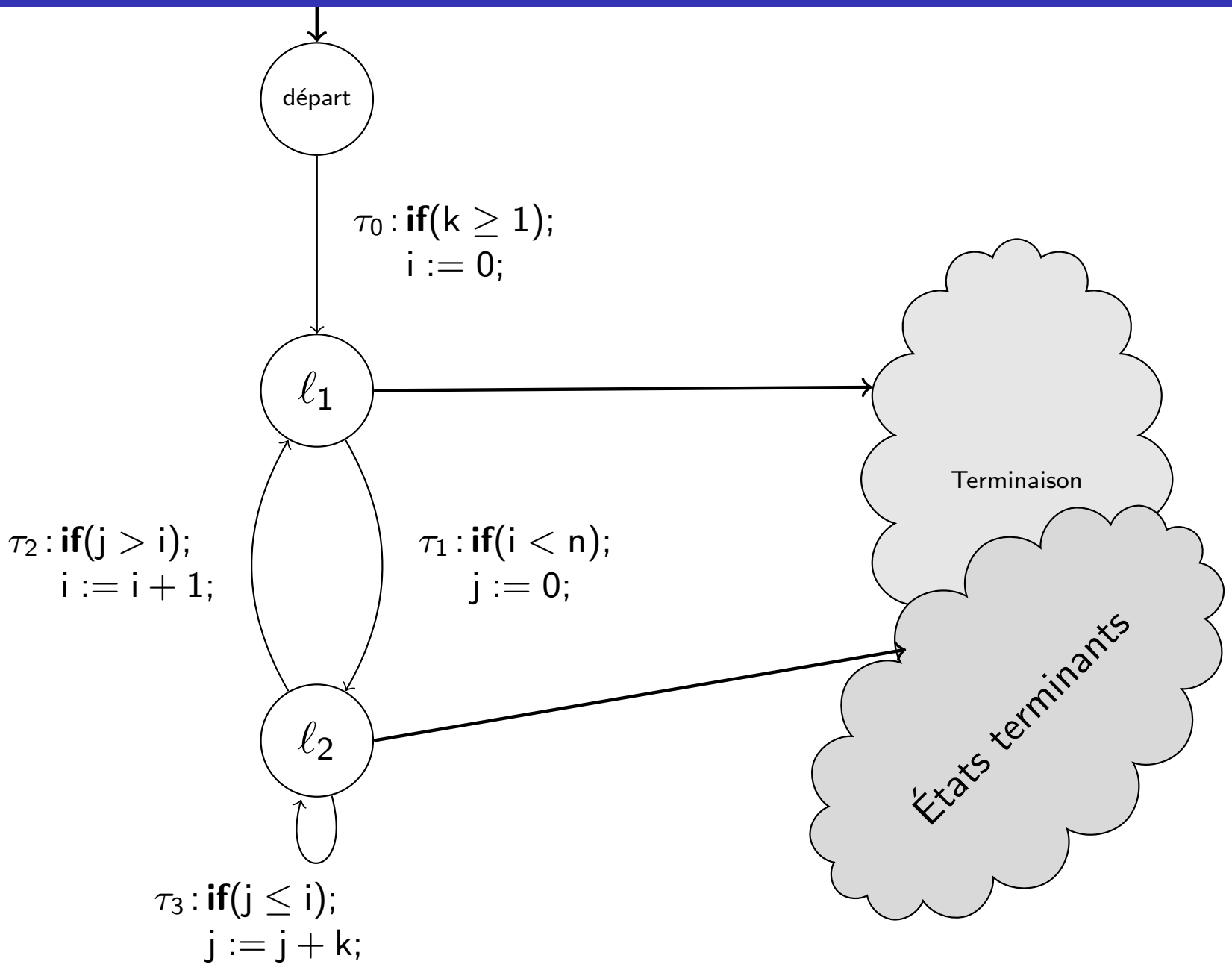
Coopération : Vue d'ensemble



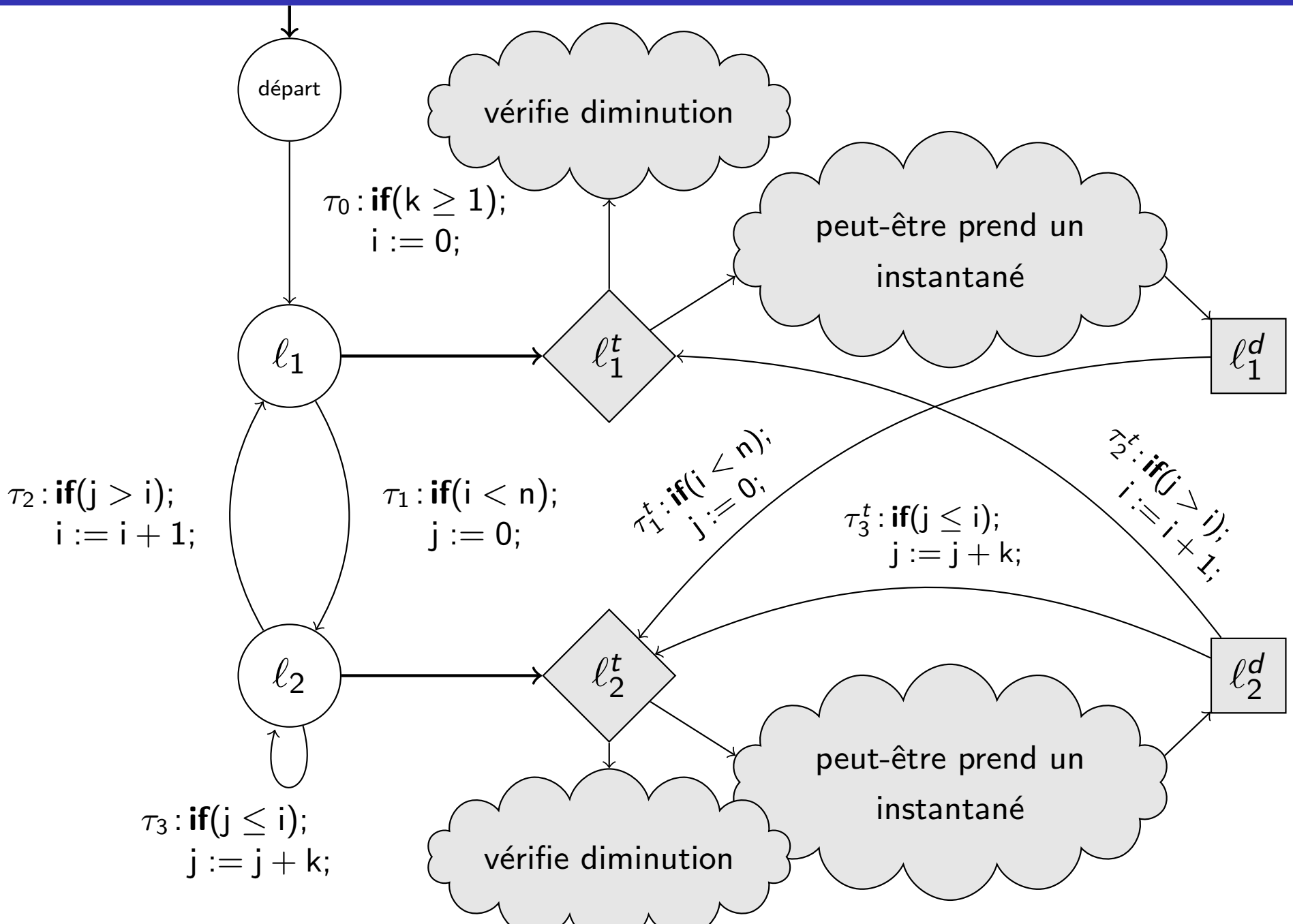
Coopération : Vue d'ensemble



Coopération : Vue d'ensemble



Coopération : Vue d'ensemble



Coopération

Intuition :

- **Sous-graphe de sûreté** : le programme original
- **Sous-graphe de terminaison** : copie instrumentée

Coopération

Intuition :

- **Sous-graphe de sûreté** : le programme original
- **Sous-graphe de terminaison** : copie instrumentée

- **Rang** : Simplifie problème, ≪ précise des parties difficiles ≫

Coopération

Intuition :

- **Sous-graphe de sûreté** : le programme original
- **Sous-graphe de terminaison** : copie instrumentée

- **Rang** : Simplifie problème, « précise des parties difficiles »
- **Sûreté** : Analyse le programme entier, « précise des invariants »

Coopération

Intuition :

- **Sous-graphe de sûreté** : le programme original
- **Sous-graphe de terminaison** : copie instrumentée

- **Rang** : Simplifie problème, « précise des parties difficiles »
- **Sûreté** : Analyse le programme entier, « précise des invariants »

Approche :

- Analyse **CFC entière**,
  pas seulement le contre-exemple (sans contexte de boucle)

Coopération

Intuition :

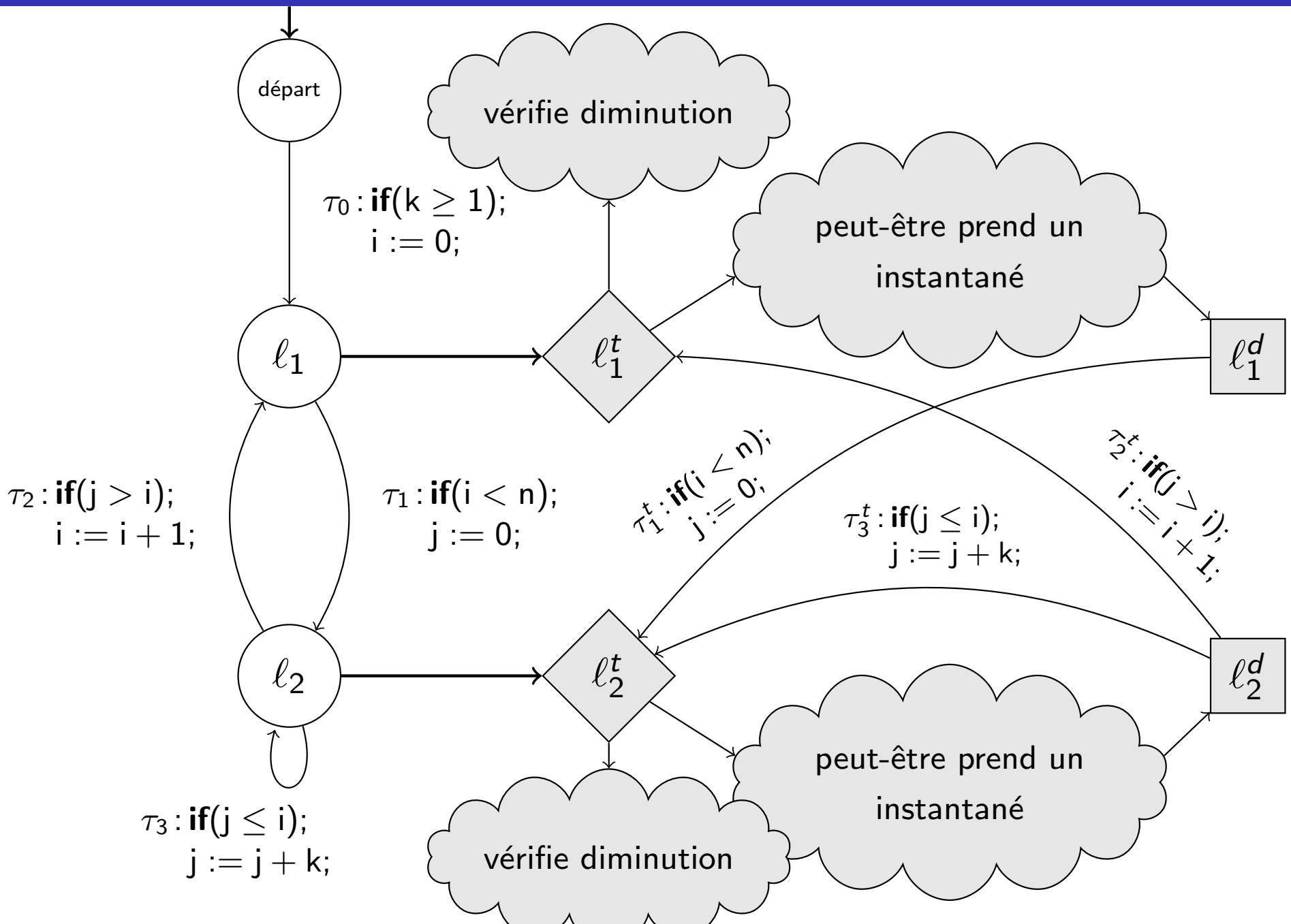
- **Sous-graphe de sûreté** : le programme original
- **Sous-graphe de terminaison** : copie instrumentée

- **Rang** : Simplifie problème, ≪ précise des parties difficiles ≫
- **Sûreté** : Analyse le programme entier, ≪ précise des invariants ≫

Approche :

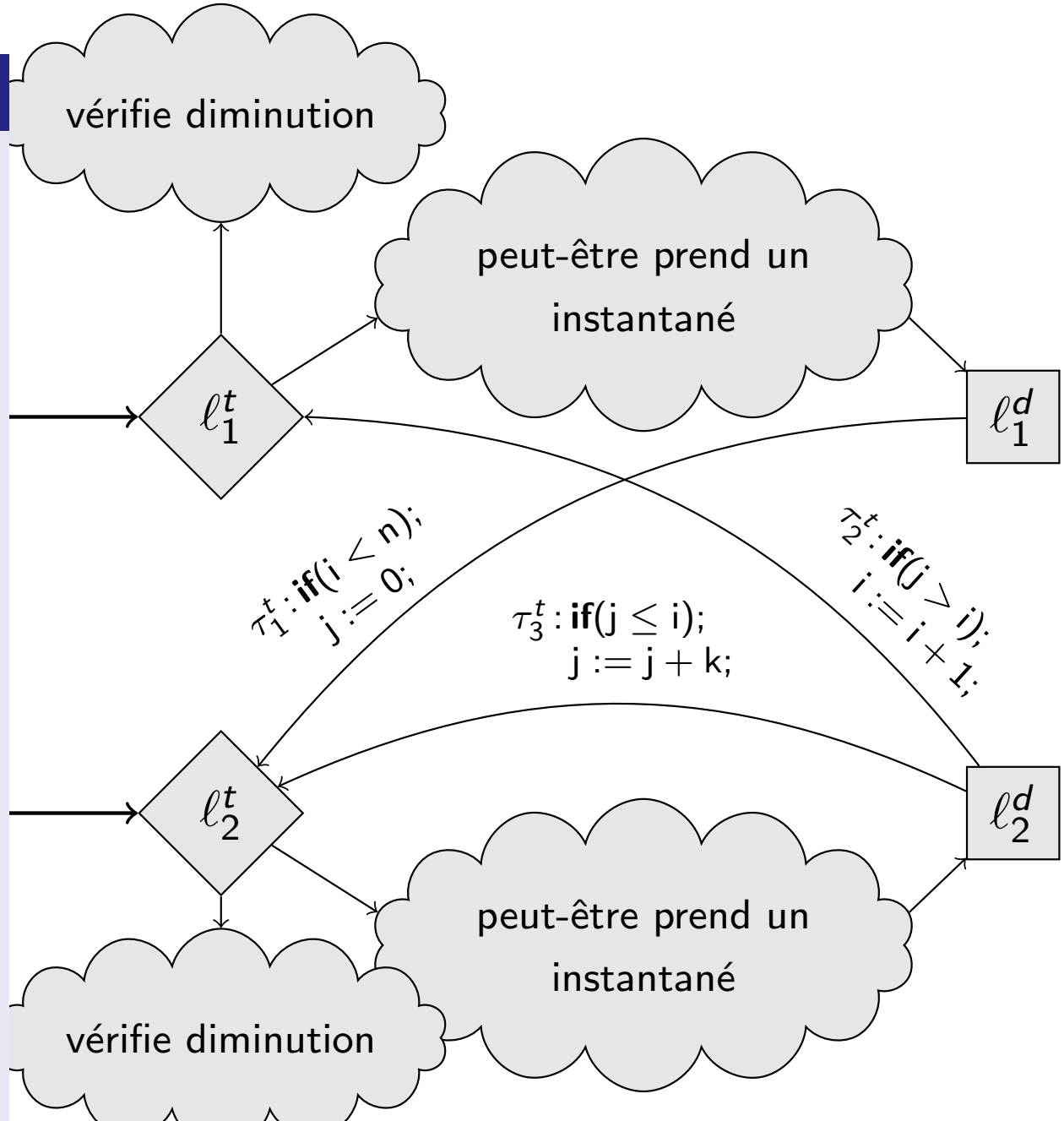
- Analyse **CFC entière**,
  pas seulement le contre-exemple (sans contexte de boucle)
- **Supprime** des transitions pendant la construction de la preuve

Coopération : Simplification



Coopération : Simplification

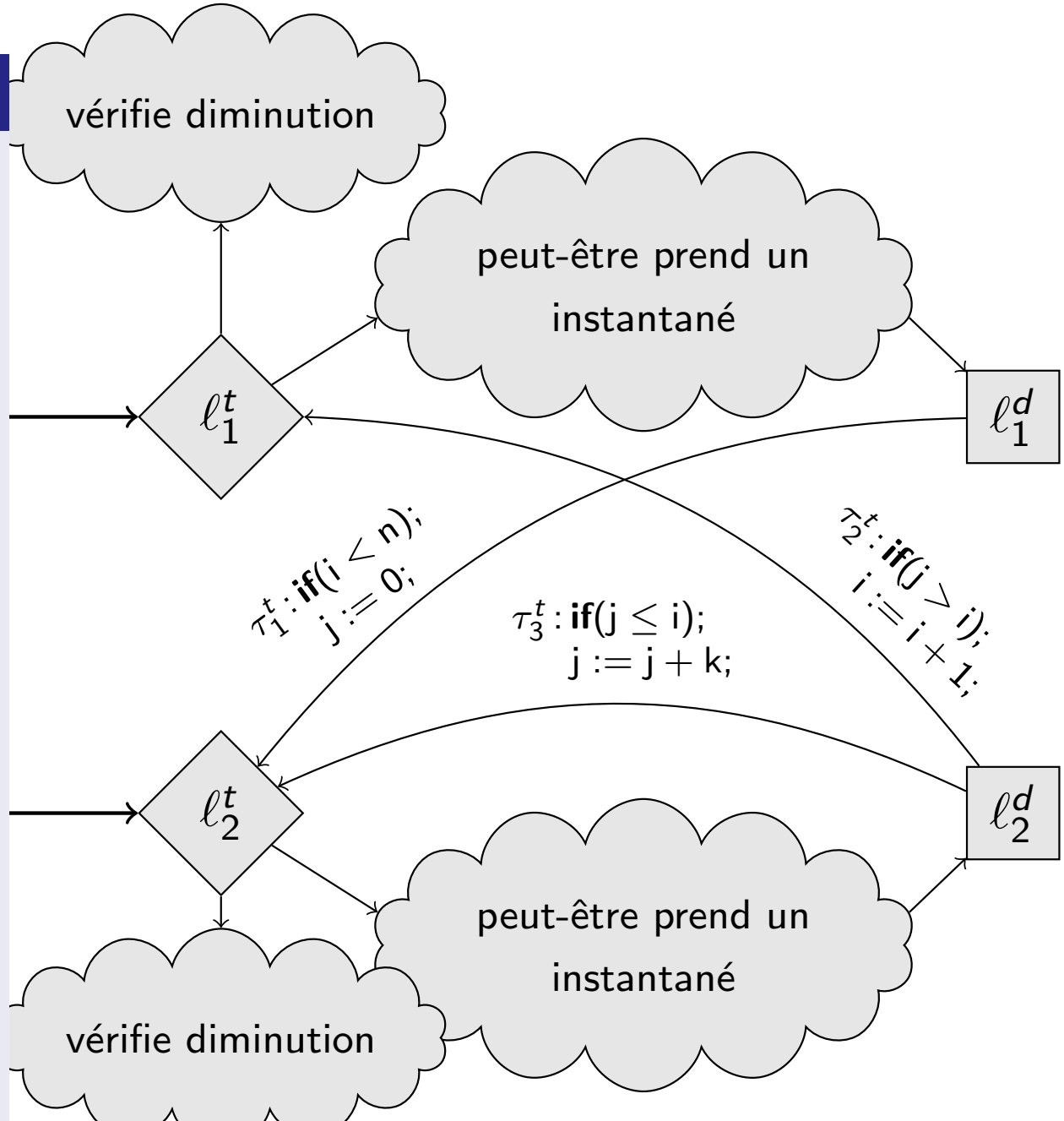
Simplification



Coopération : Simplification

Simplification

1. Trouve CFC $\mathcal{S}$ dans le graphe de terminaison :
   $\ell_1^t, \ell_1^d, \ell_2^t, \ell_2^d$



Coopération : Simplification

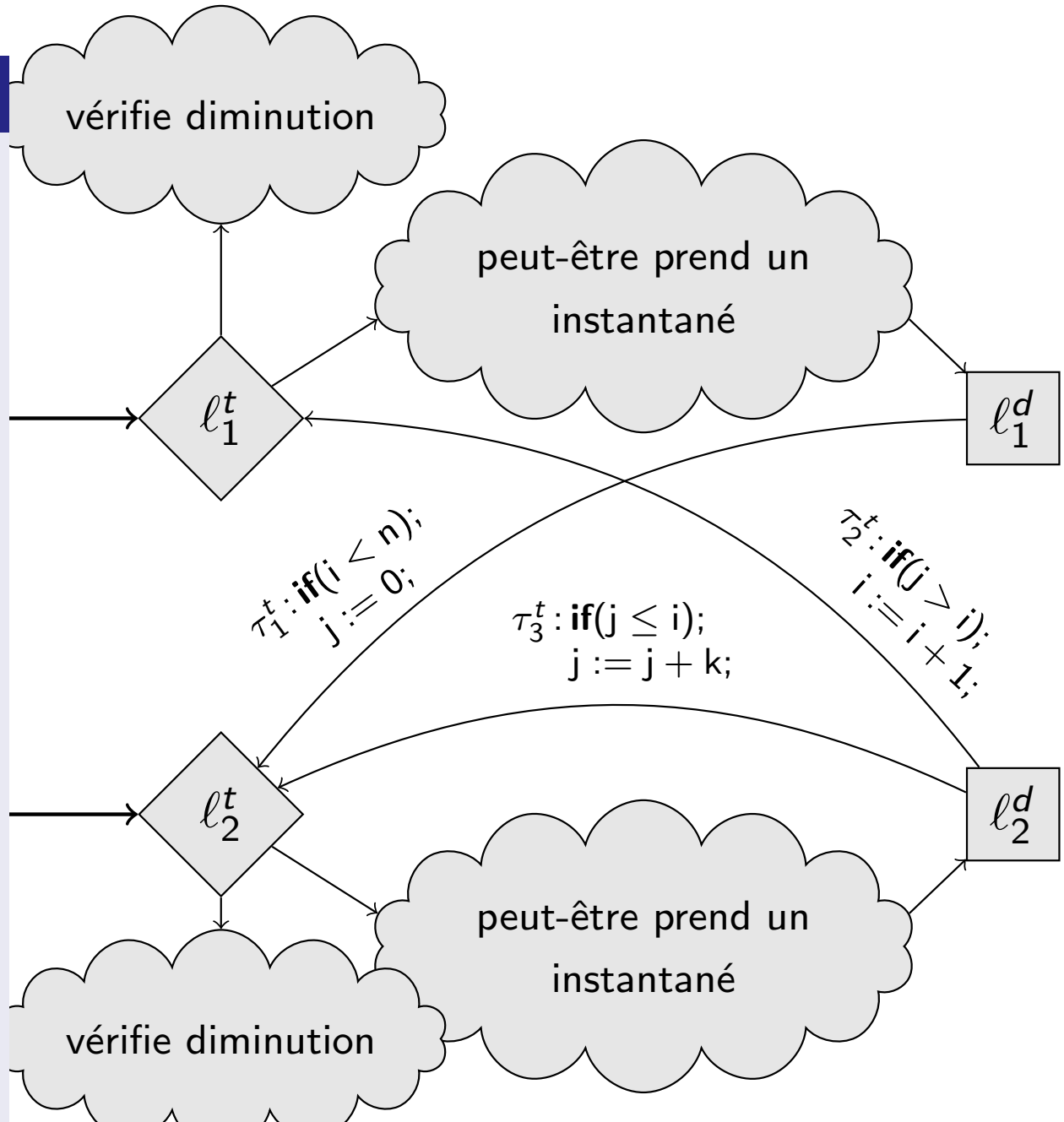
Simplification

1. Trouve CFC $\mathcal{S}$ dans le graphe de terminaison :

$$\ell_1^t, \ell_1^d, \ell_2^t, \ell_2^d$$

2. Trouve FR $\mathcal{S}$-orientante :

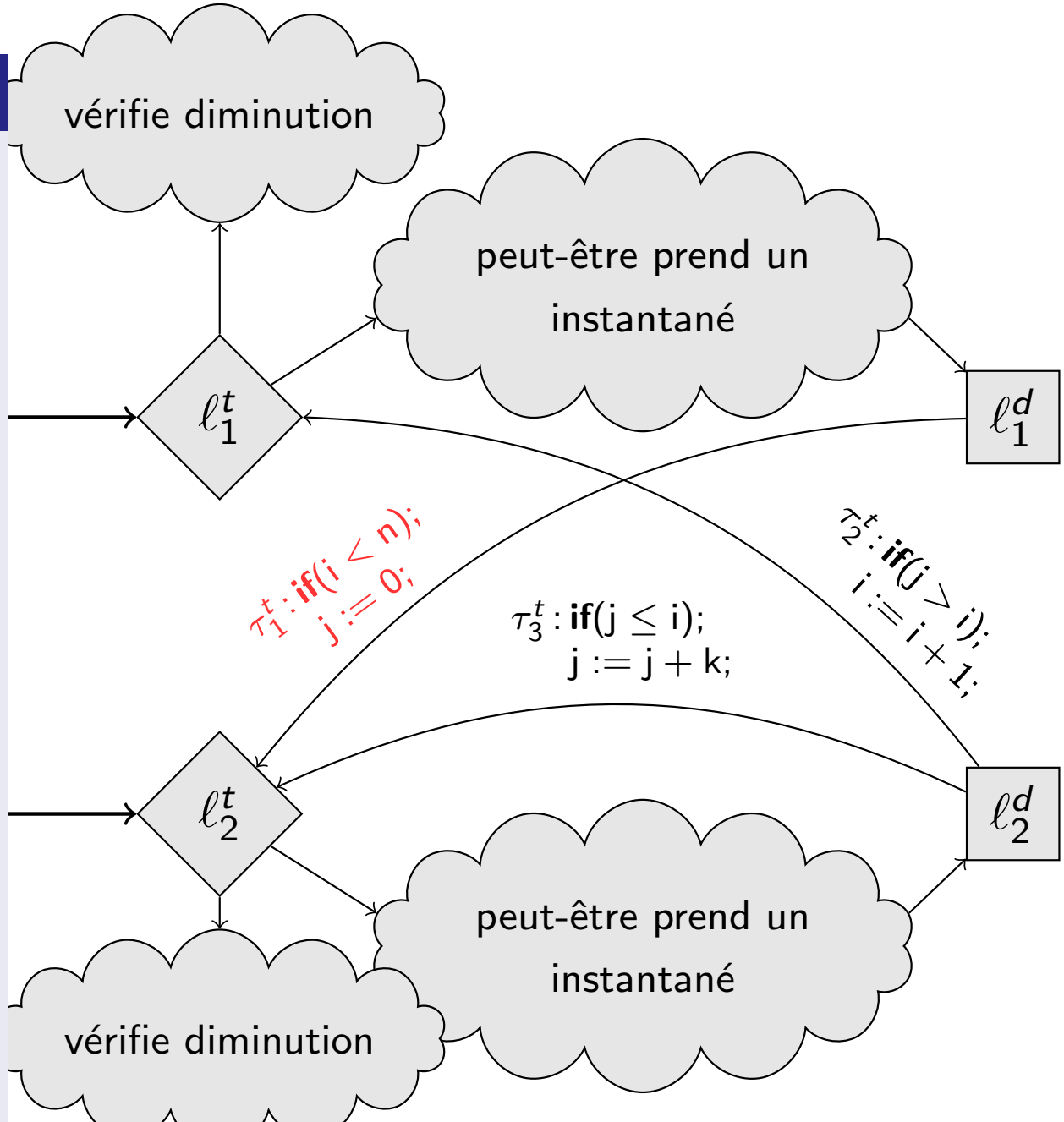
$$f_{\ell_1^t}(i,j,k,n) = n - i + 1$$
$$f_{\ell_1^d}(i,j,k,n) = n - i + 1$$
$$f_{\ell_2^t}(i,j,k,n) = n - i$$
$$f_{\ell_2^d}(i,j,k,n) = n - i$$



Coopération : Simplification

Simplification

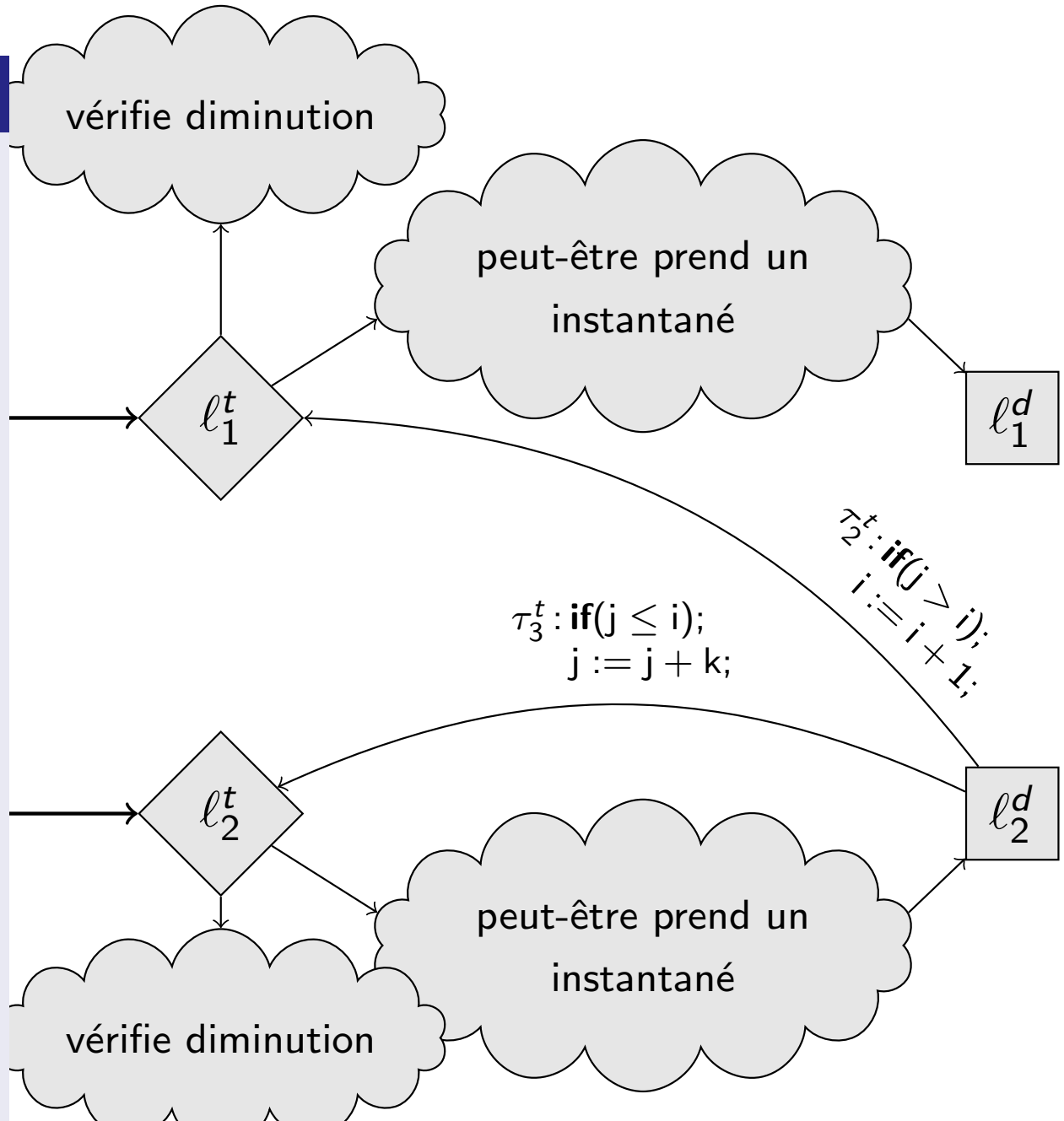
1. Trouve CFC $\mathcal{S}$ dans le graphe de terminaison :
   $$\ell_1^t, \ell_1^d, \ell_2^t, \ell_2^d$$
2. Trouve FR $\mathcal{S}$-orientante :
   $$f_{\ell_1^t}(i,j,k,n) = n - i + 1$$
   $$f_{\ell_1^d}(i,j,k,n) = n - i + 1$$
   $$f_{\ell_2^t}(i,j,k,n) = n - i$$
   $$f_{\ell_2^d}(i,j,k,n) = n - i$$
3. Supprime transitions qui diminuent



Coopération : Simplification

Simplification

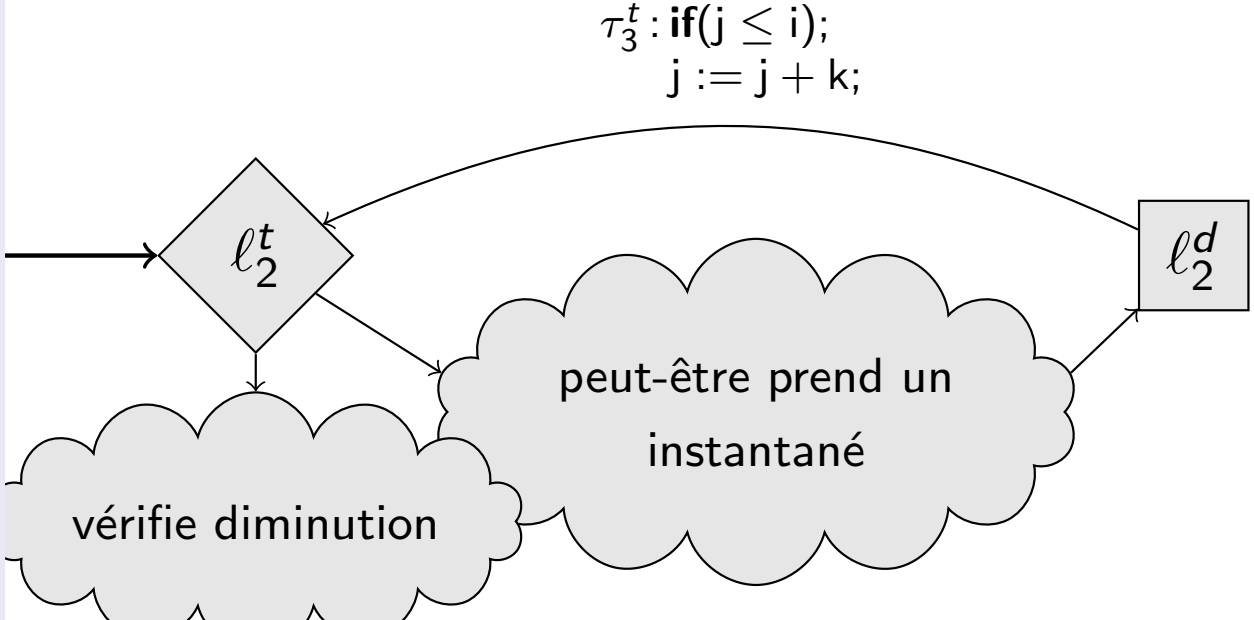
1. Trouve CFC $\mathcal{S}$ dans le graphe de terminaison :
   $$\ell_1^t, \ell_1^d, \ell_2^t, \ell_2^d$$
2. Trouve FR $\mathcal{S}$-orientante :
   $$f_{\ell_1^t}(\mathsf{i},\mathsf{j},\mathsf{k},\mathsf{n}) = \mathsf{n} - \mathsf{i} + 1$$
   $$f_{\ell_1^d}(\mathsf{i},\mathsf{j},\mathsf{k},\mathsf{n}) = \mathsf{n} - \mathsf{i} + 1$$
   $$f_{\ell_2^t}(\mathsf{i},\mathsf{j},\mathsf{k},\mathsf{n}) = \mathsf{n} - \mathsf{i}$$
   $$f_{\ell_2^d}(\mathsf{i},\mathsf{j},\mathsf{k},\mathsf{n}) = \mathsf{n} - \mathsf{i}$$
3. Supprime transitions qui diminuent



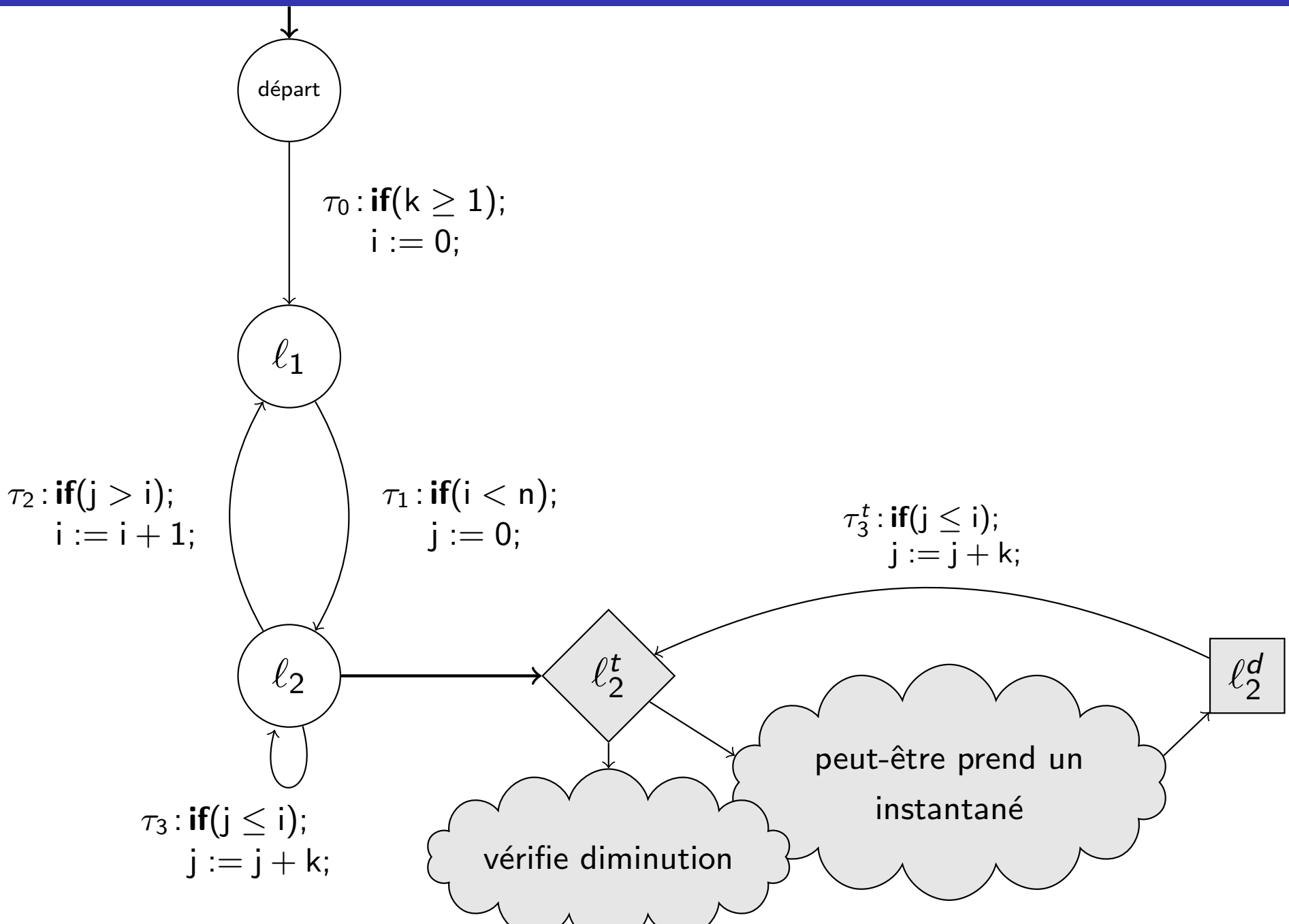
Coopération : Simplification

Simplification

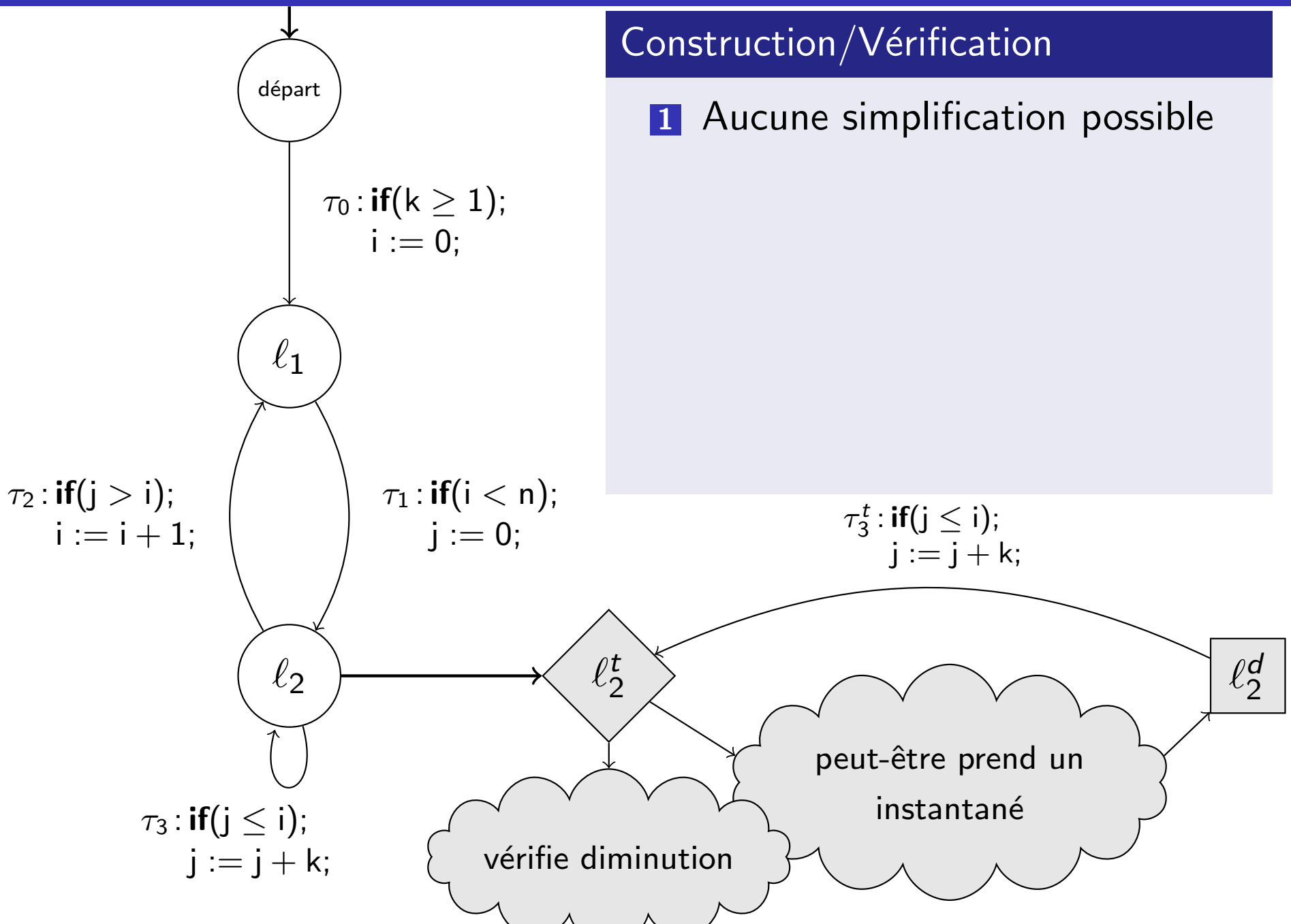
1. Trouve CFC $\mathcal{S}$ dans le graphe de terminaison :
   $$\ell_1^t, \ell_1^d, \ell_2^t, \ell_2^d$$
2. Trouve FR $\mathcal{S}$-orientante :
   $$f_{\ell_1^t}(\mathsf{i},\mathsf{j},\mathsf{k},\mathsf{n}) = \mathsf{n} - \mathsf{i} + 1$$
   $$f_{\ell_1^d}(\mathsf{i},\mathsf{j},\mathsf{k},\mathsf{n}) = \mathsf{n} - \mathsf{i} + 1$$
   $$f_{\ell_2^t}(\mathsf{i},\mathsf{j},\mathsf{k},\mathsf{n}) = \mathsf{n} - \mathsf{i}$$
   $$f_{\ell_2^d}(\mathsf{i},\mathsf{j},\mathsf{k},\mathsf{n}) = \mathsf{n} - \mathsf{i}$$
3. Supprime transitions qui diminuent
4. Nettoie



Coopération



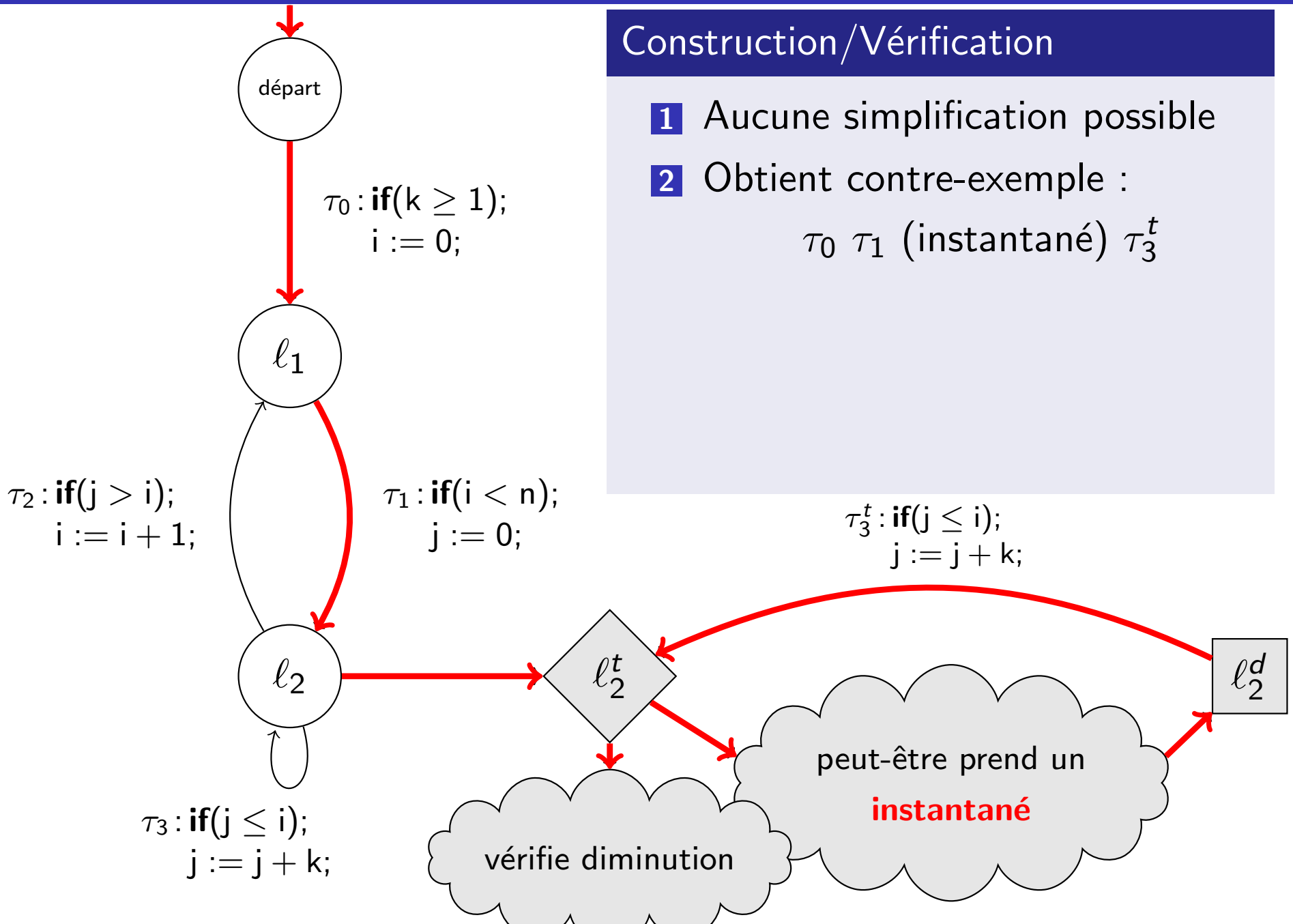
Coopération : Invariants



Construction/Vérification

1. Aucune simplification possible

Coopération : Invariants

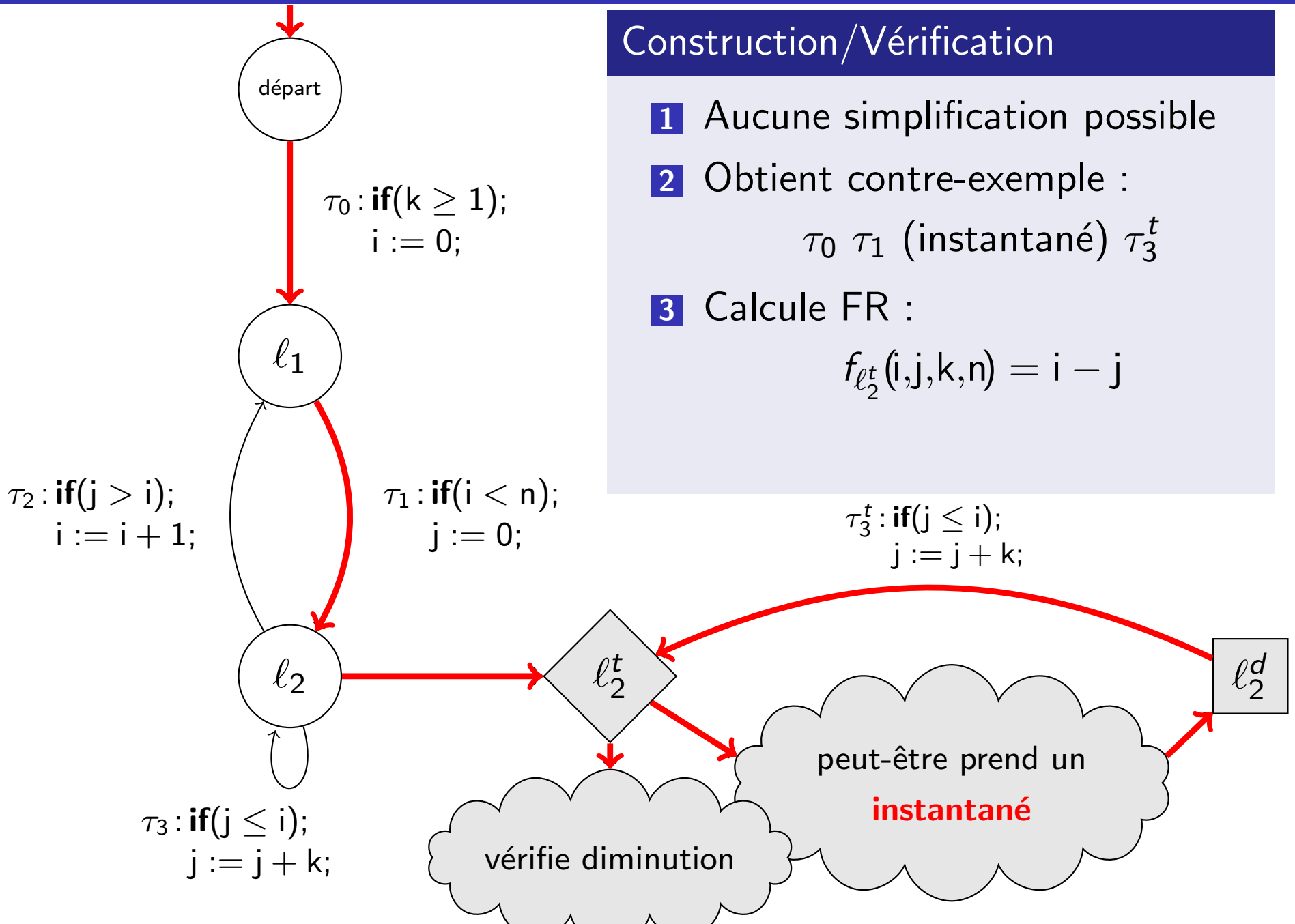


Construction/Vérification

1. Aucune simplification possible
2. Obtient contre-exemple :

$$\tau_0 \ \tau_1 \text{ (instantané) } \tau_3^t$$

Coopération : Invariants



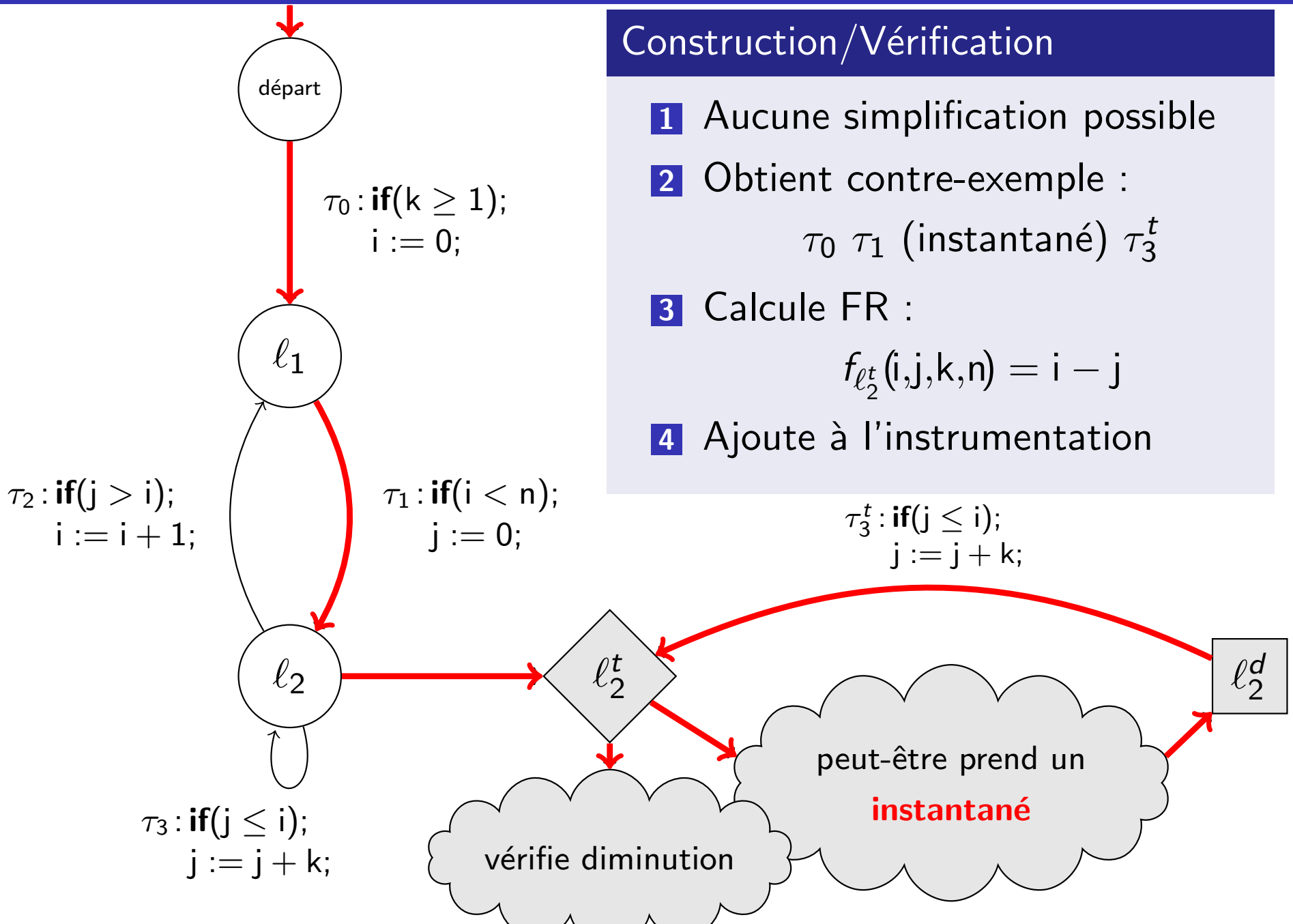
Construction/Vérification

1. Aucune simplification possible
2. Obtient contre-exemple :

   τ_0 τ_1 (instantané) τ_3^t
3. Calcule FR :

   $f_{\ell_2^t}(i,j,k,n) = i - j$

Coopération : Invariants



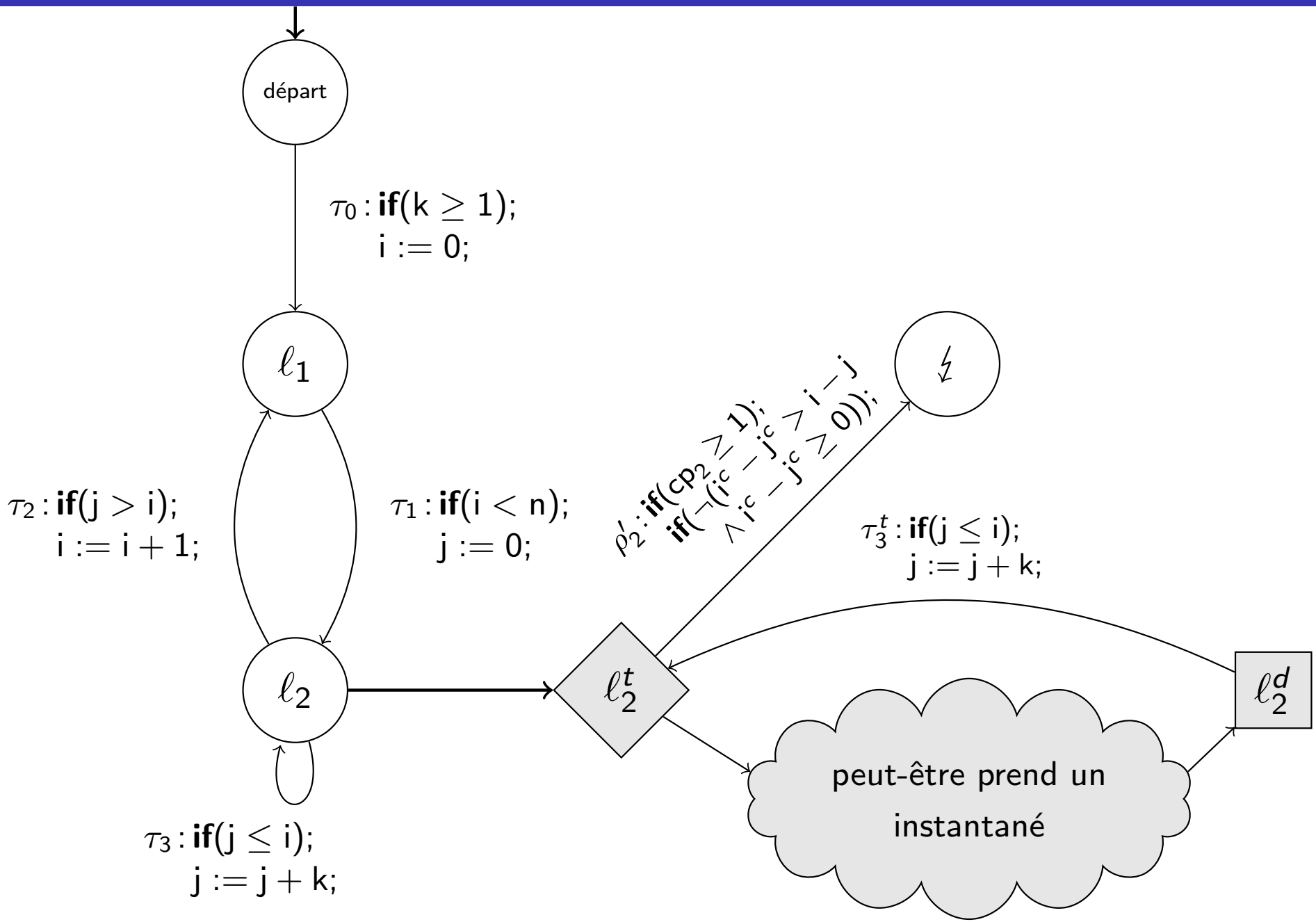
Construction/Vérification

1. Aucune simplification possible
2. Obtient contre-exemple :

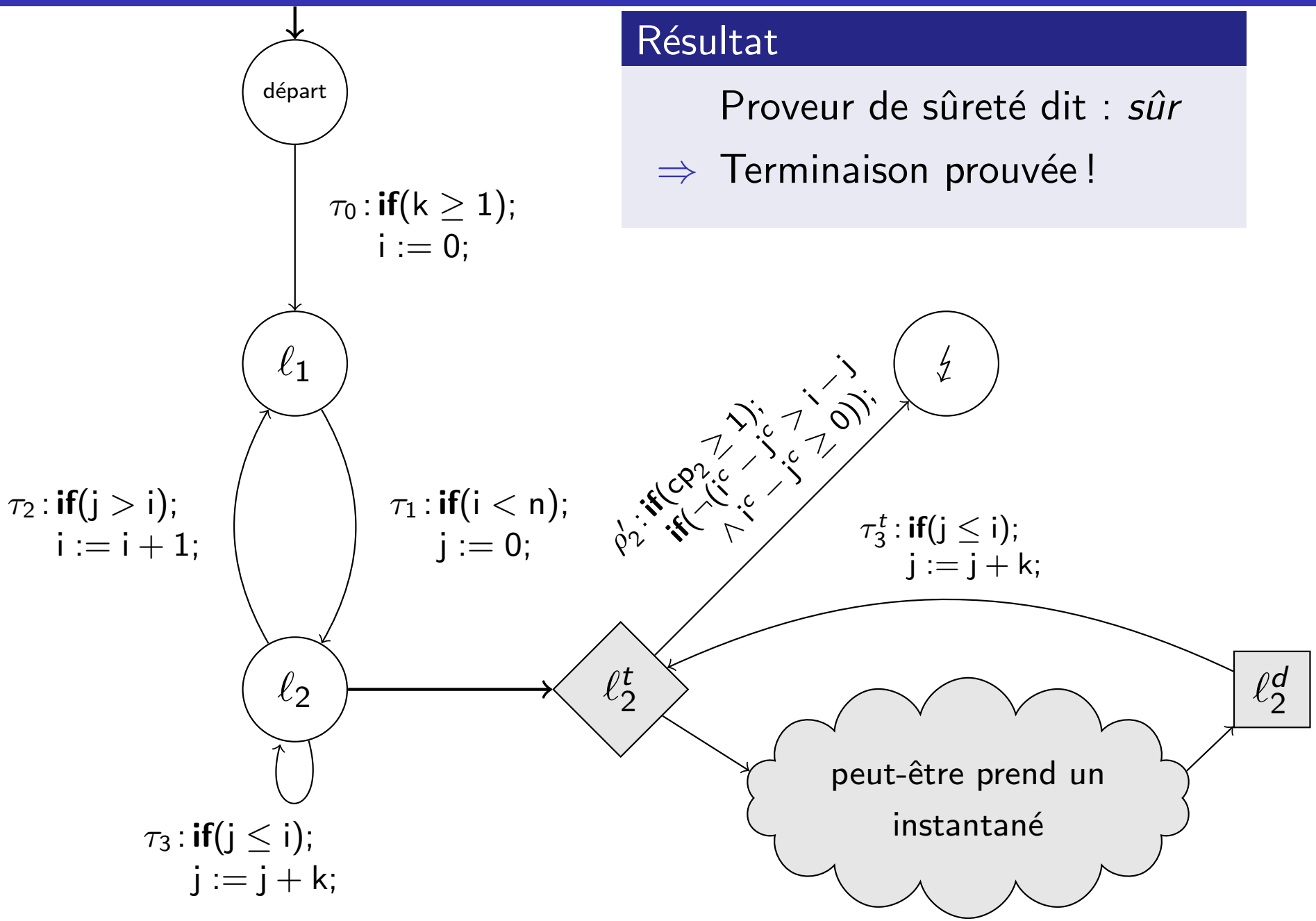
   τ_0 τ_1 (instantané) τ_3^t
3. Calcule FR :

   $f_{\ell_2^t}(\mathrm{i},\mathrm{j},\mathrm{k},\mathrm{n}) = \mathrm{i} - \mathrm{j}$
4. Ajoute à l'instrumentation

Coopération : Invariants



Coopération : Invariants



Résultat

Proveur de sûreté dit : *sûr*

$\Rightarrow$ Terminaison prouvée !

Coopération : Évaluation

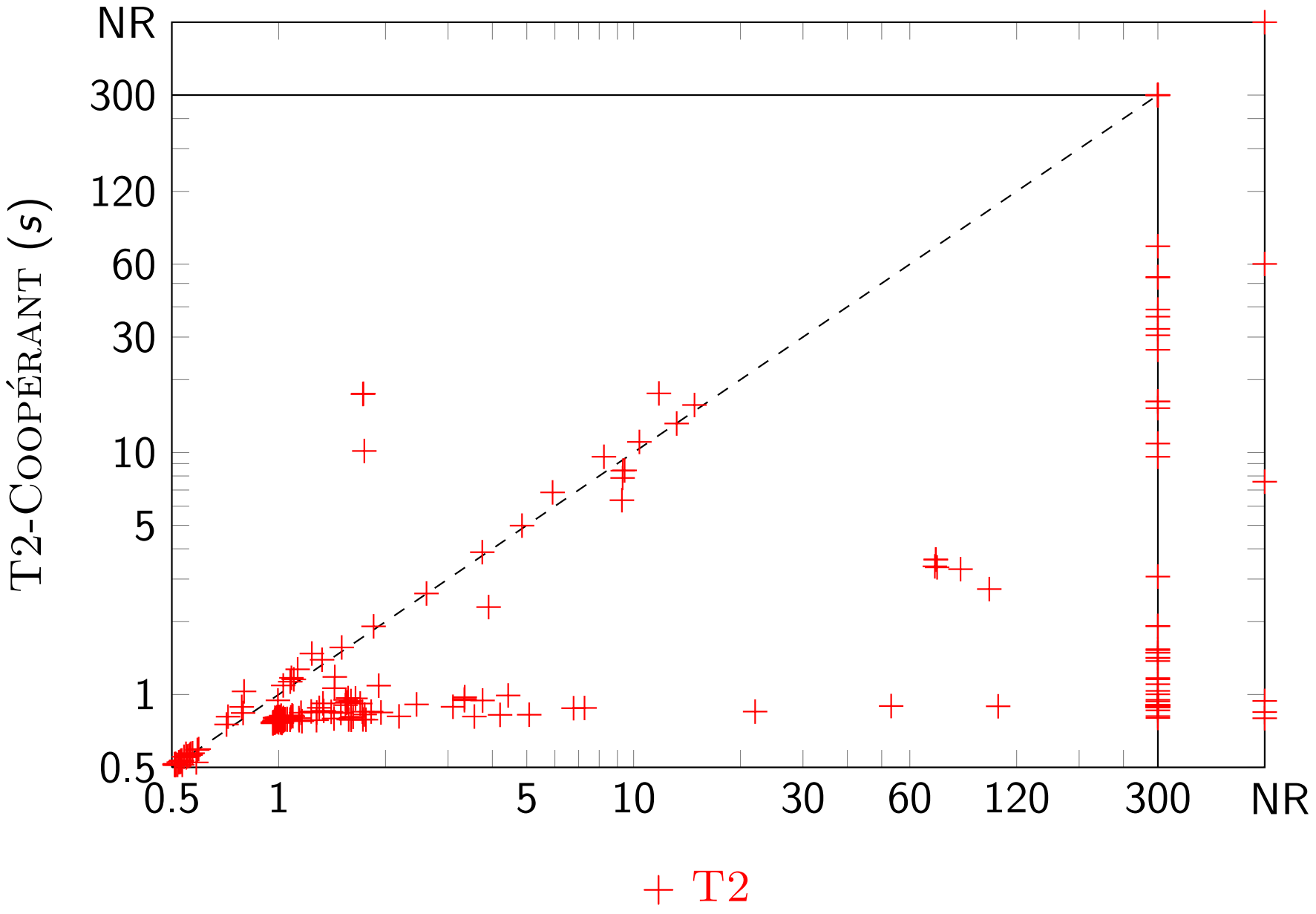
Évalué sur 449 benchmarks pour les preuves de terminaison (300 *s* timeout)
260 connus terminants, 181 connus non-terminants, 8 inconnus
Sources : pilotes de Windows, Apache, PostgreSQL, ...

Coopération : Évaluation

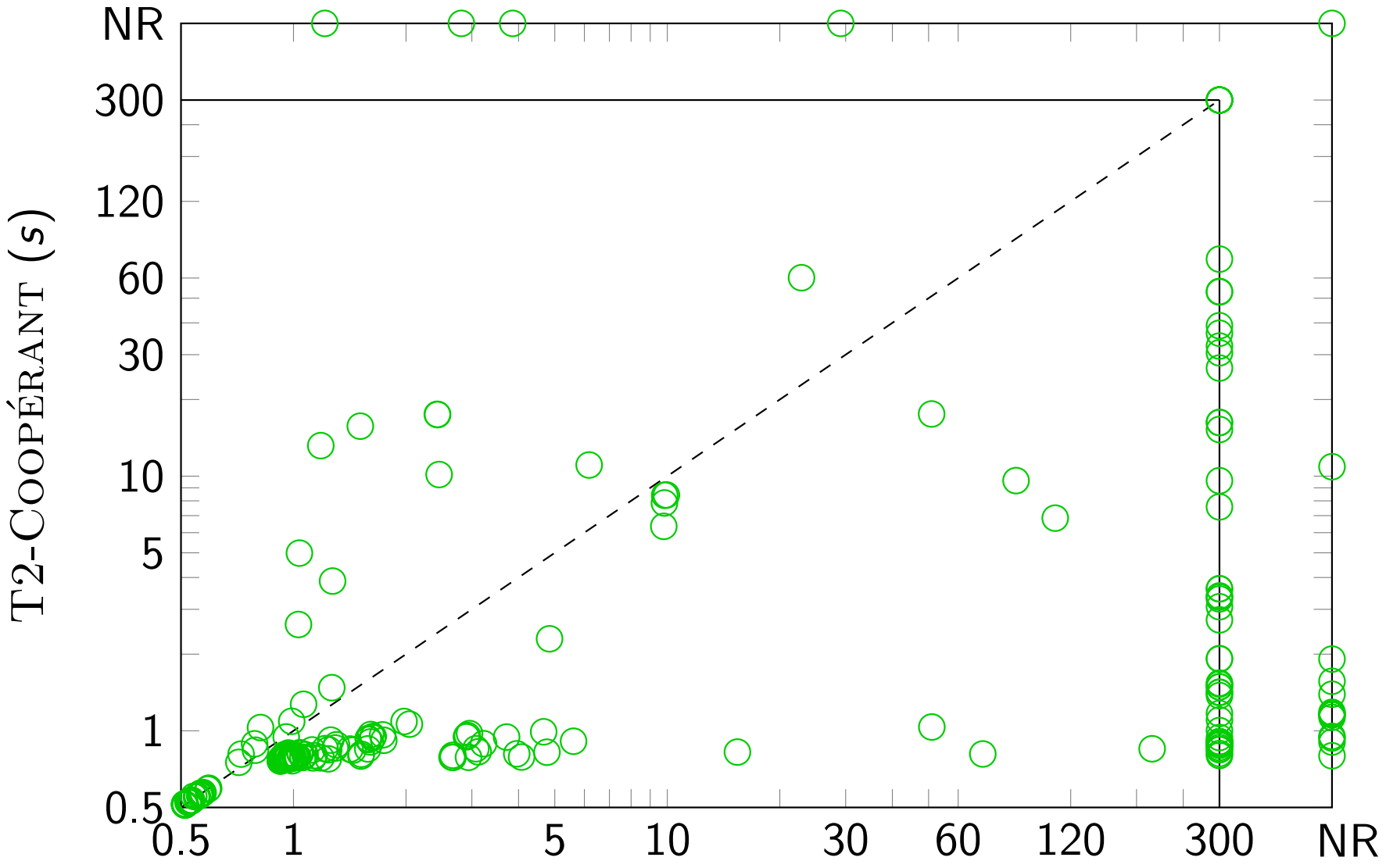
Évalué sur 449 benchmarks pour les preuves de terminaison (300 *s* timeout)
260 connus terminants, 181 connus non-terminants, 8 inconnus
Sources : pilotes de Windows, Apache, PostgreSQL, ...

	Term (#)	Term (temps moyen, *s*)
T2-Coopérant	245	3.42
AProVE	197	2.21
KITTeL	196	4.65
T2	189	5.15
AProVE+Interproc	185	1.53
Terminator	177	4.99
Size-Change/MCNP	156	17.50
ARMC	138	16.16

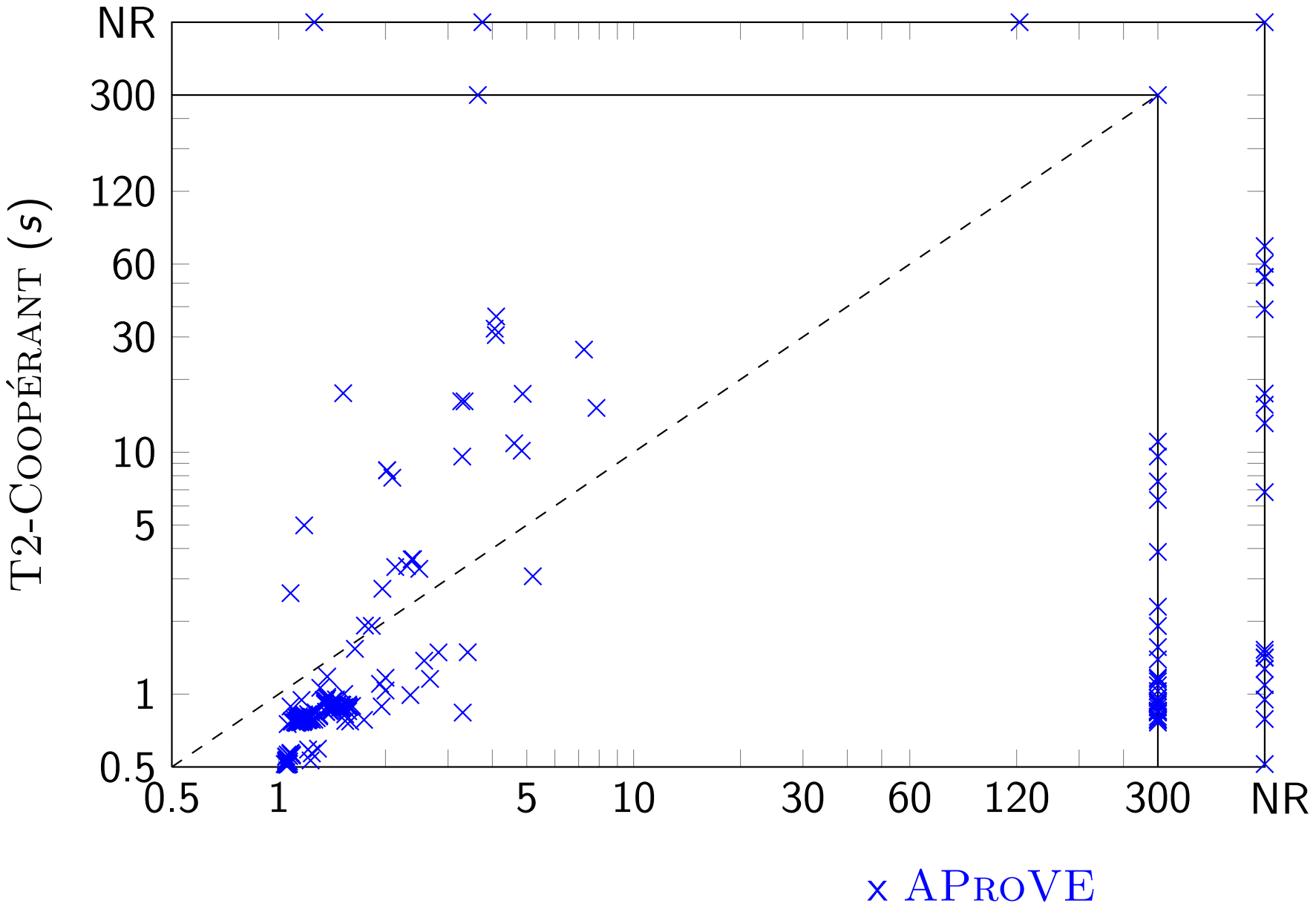
Coopération : Évaluation



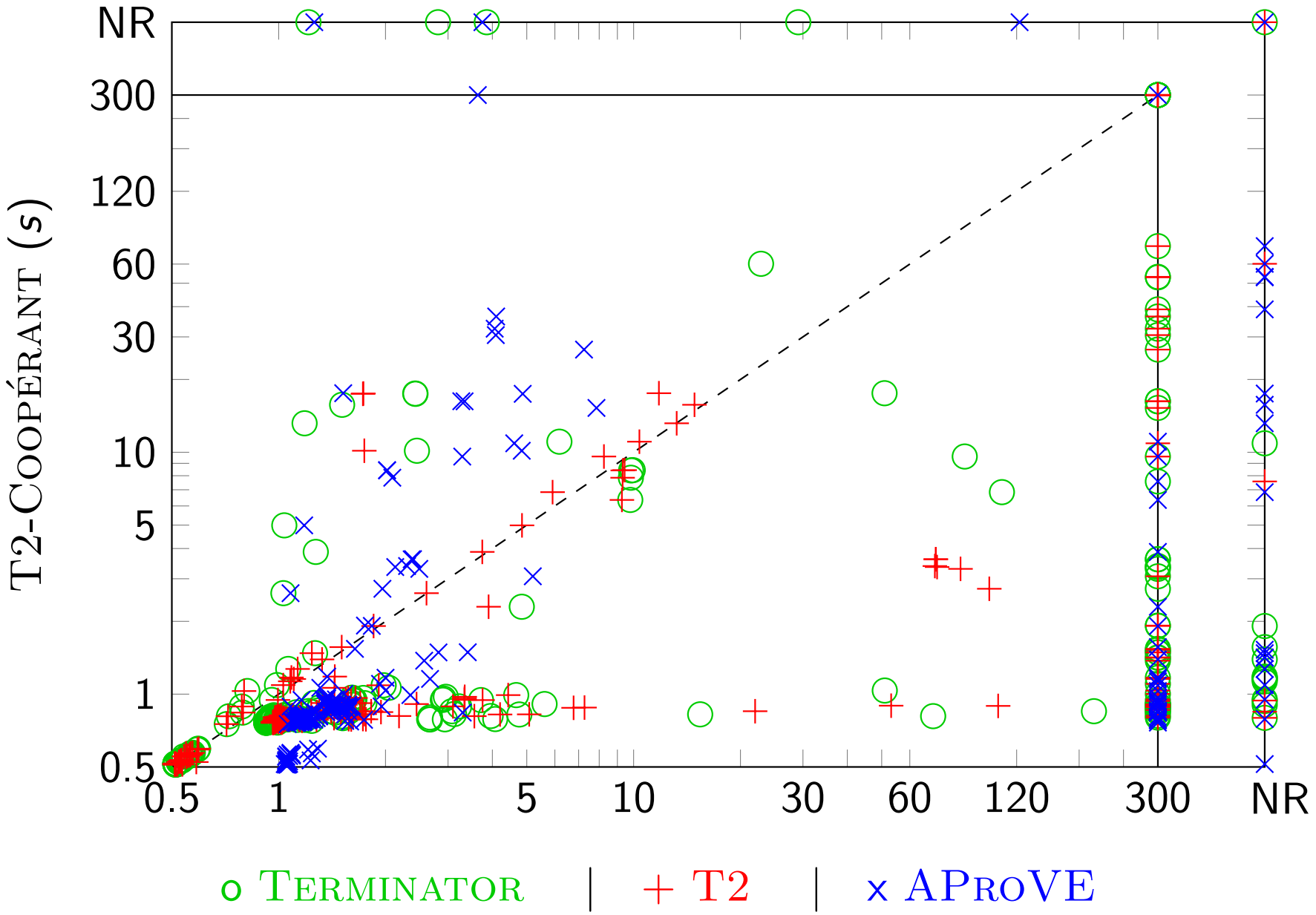
Coopération : Évaluation



Coopération : Évaluation



Coopération : Évaluation



Conclusion

Évalué sur 449 benchmarks pour les preuves de terminaison (300 *s* timeout)
260 connus terminants, 181 connus non-terminants, 8 inconnus
Sources : pilotes de Windows, Apache, PostgreSQL, ...

	Term (#)	Term (temps moyen, *s*)
T2-Coopérant	245	3.42
AProVE	197	2.21
KITTeL	196	4.65
T2	189	5.15
AProVE+Interproc	185	1.53
Terminator	177	4.99
Size-Change/MCNP	156	17.50
ARMC	138	16.16

Détails : http://verify.rwth-aachen.de/brockschmidt/Cooperating-T2/

Sources disponibles :
http://research.microsoft.com/en-us/projects/t2/

Article de conférence : [Brockschmidt, Cook, Fuhs, *Actes de CAV 2013*]