Introduction à la science informatique pour l'étude des systèmes dynamiques

Loïc Paulevé

École Polytechnique / LIX (équipe AMIB)
pauleve@lix.polytechnique.fr

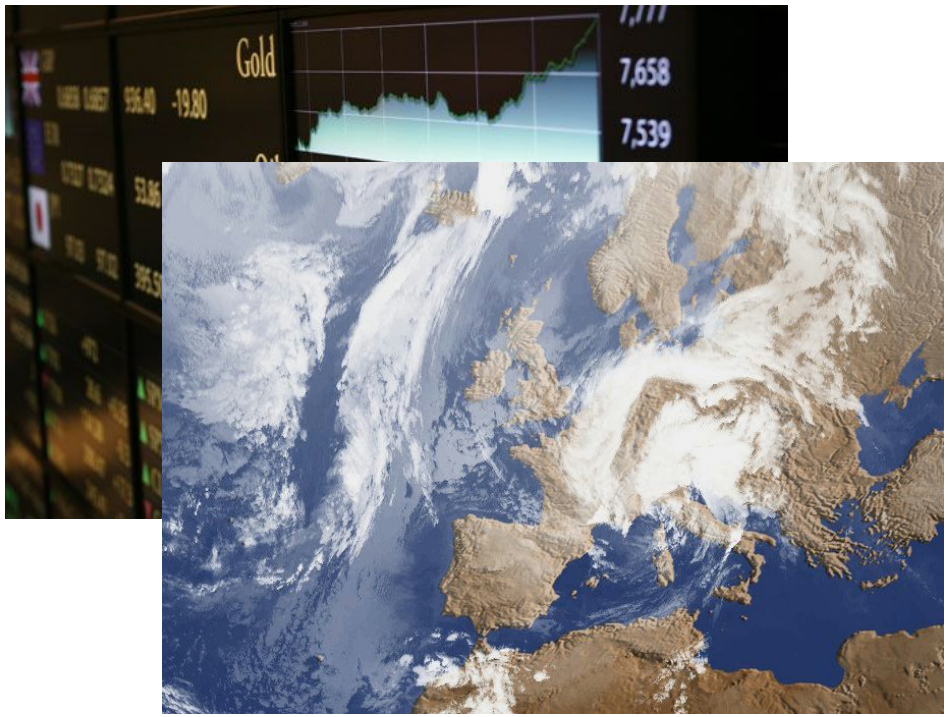
http://loicpauleve.name

Plan

Systèmes dynamiques complexes

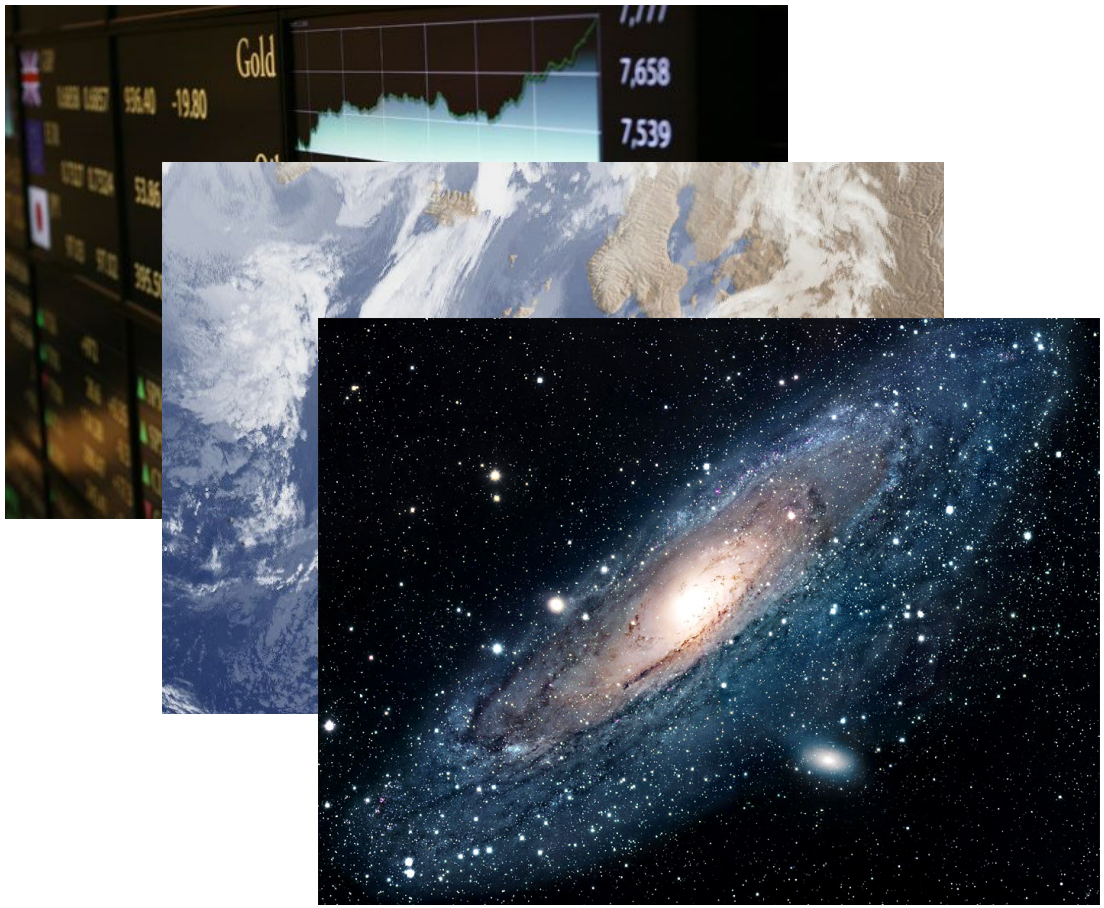
Interactions locales “simples”, comportement global complexe.

Systèmes dynamiques complexes



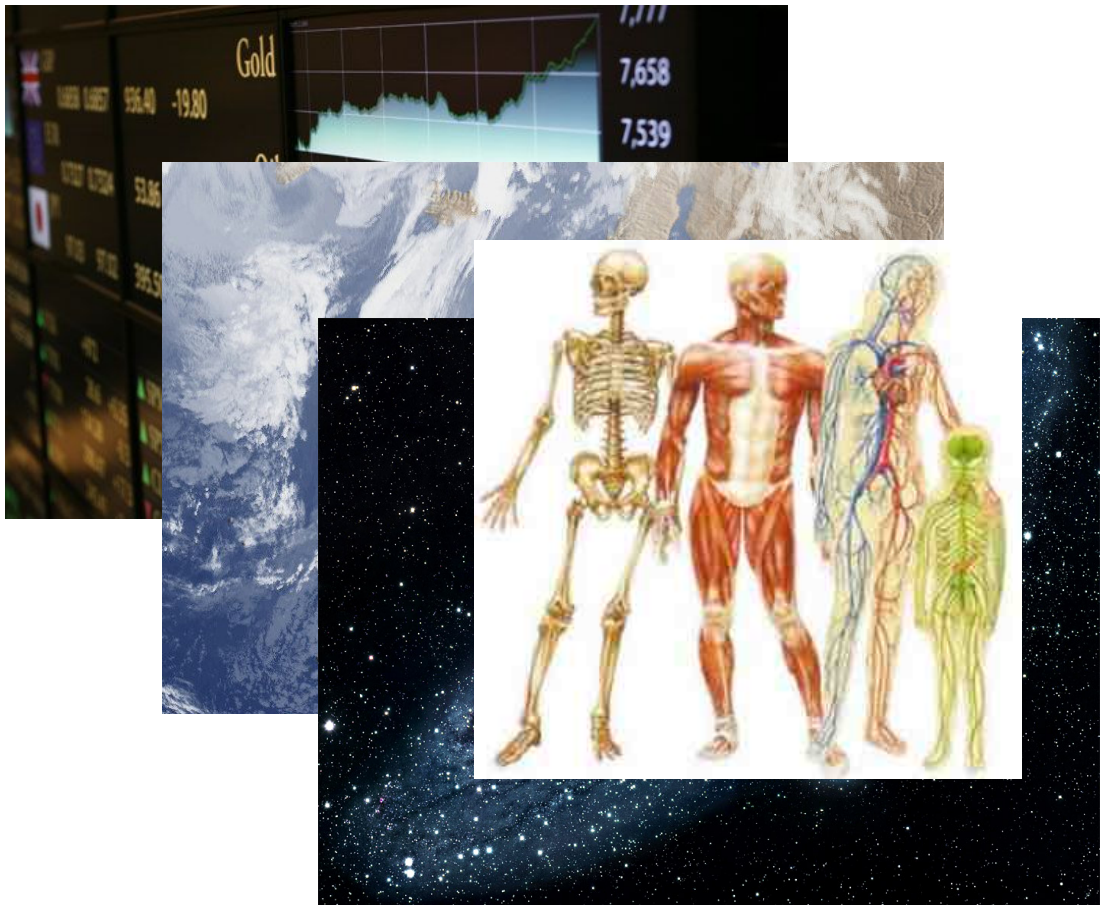
Interactions locales “simples”, comportement global complexe.

Systèmes dynamiques complexes



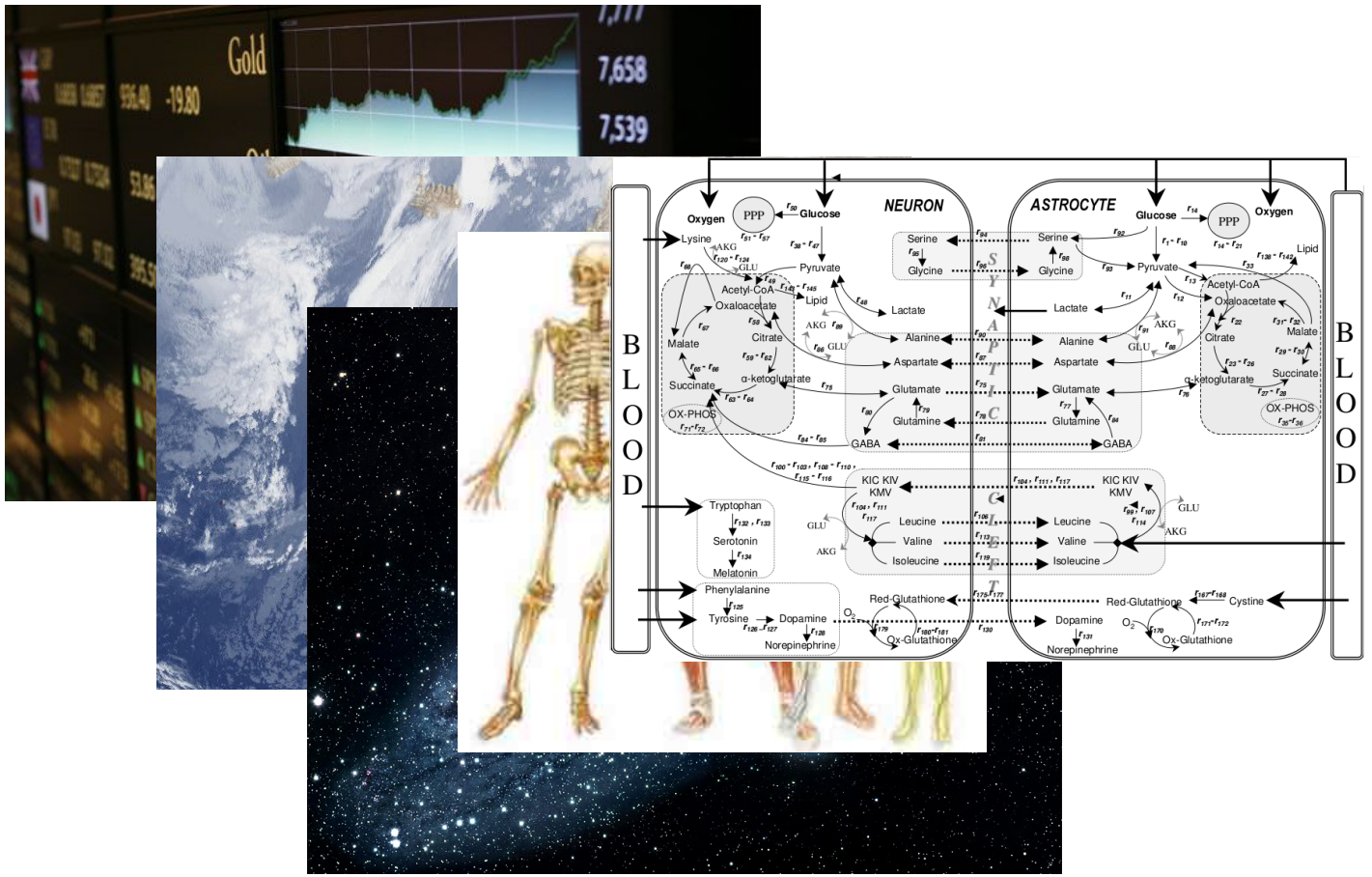
Interactions locales “simples”, comportement global complexe.

Systèmes dynamiques complexes



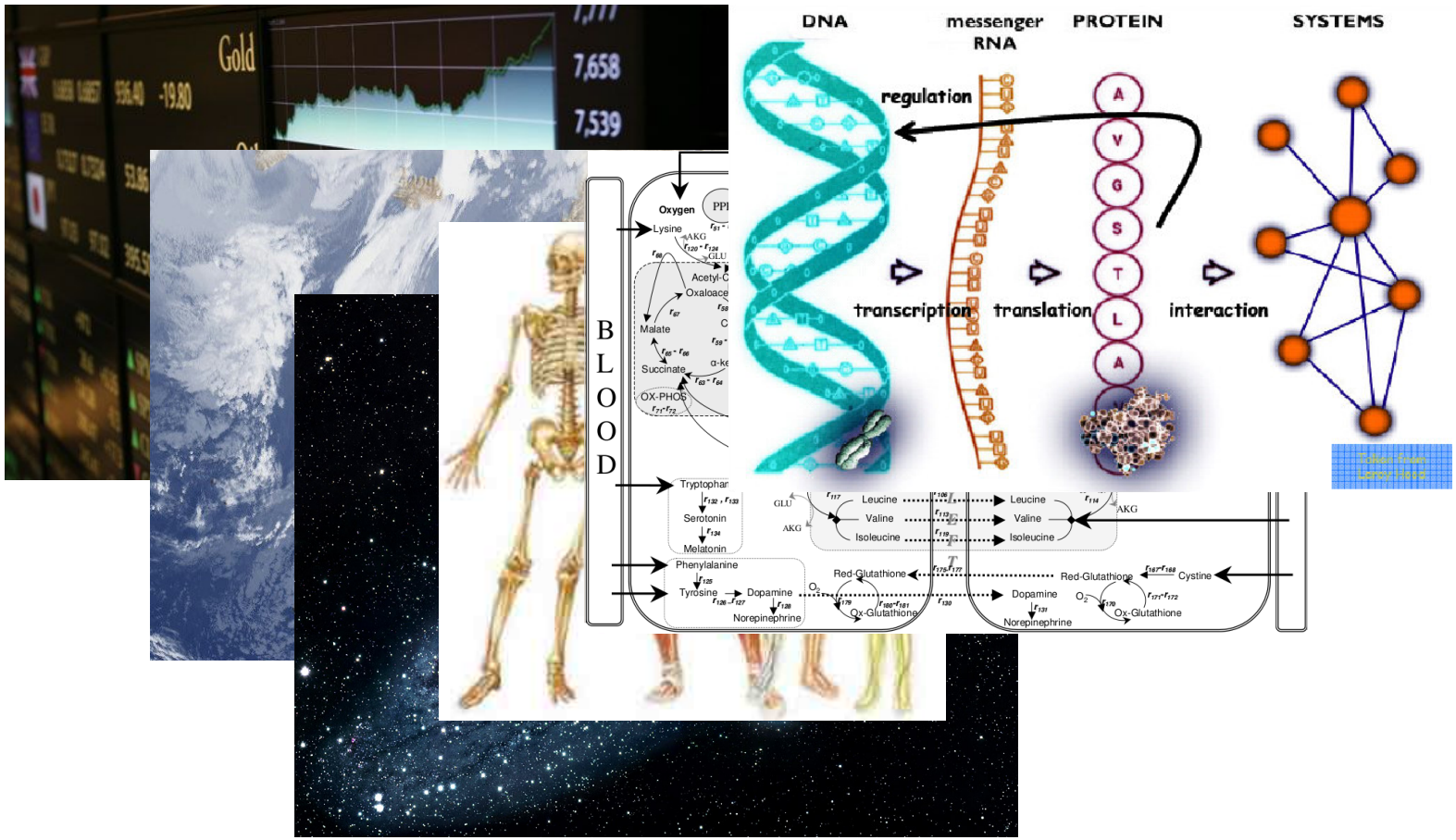
Interactions locales “simples”, comportement global complexe.

Systèmes dynamiques complexes



Interactions locales “simples”, comportement global complexe.

Systèmes dynamiques complexes



Interactions locales “simples”, comportement global complexe.

Le principe de la modélisation

Briques/modèles ← connaissances, expériences, intuitions.

Propriétés

Propriétés

Propriétés

Quelques buts de la modélisation

Reproduire un comportement

- Simulations, analyses exhaustives.
- Prédictions : comportements possibles/impossibles dans le modèle.
- Expériences in vitro : faciles, gratuites, rapides, sans erreur.

Comprendre un comportement

- Briques nécessaires ou suffisantes ;
- Robustesse aux modifications du modèle.
- Étude de la causalité, composants clés.
- Relations entre différents modèles du même système.

$\Rightarrow$ contrôler le système réel.

But de ce cours

Donner quelques bases de l'informatique

- Exemples autour des graphes.
- Brève introduction à l'algorithmique.
- Brève introduction aux langages et leur sémantique.

Méthodes informatiques pour l'étude des systèmes complexes

- Notion de système de transition.
- Simplification d'un modèle.
- Interprétation abstraite.

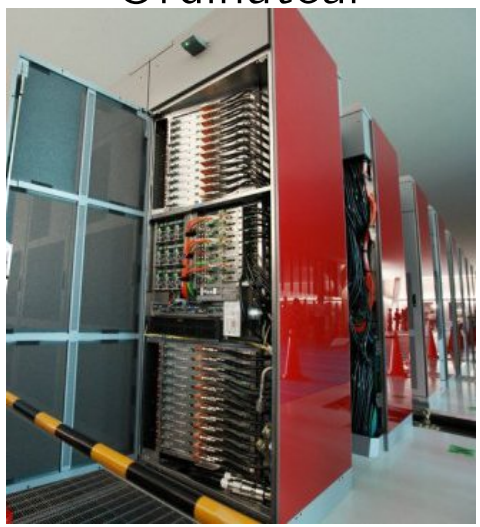
Plan

Humain contre ordinateur

Humain

- Intuitions
- Émotions
- Erreurs
- Lenteur (de calcul)

Ordinateur



- Stupidité
- Formalisation
- Exactitude
- Rapidité (de calcul)

Formalisation

Détermine les règles (briques) :

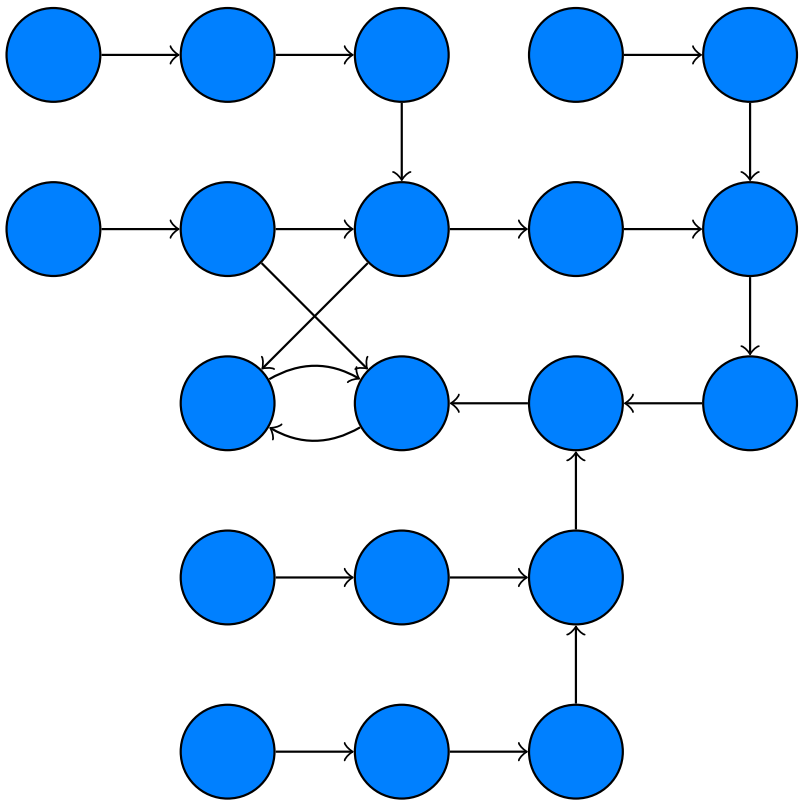
- Formulation (d'un problème) ;
- Raisonnement (manière de résoudre un problème).

Étape cruciale pour généraliser et prouver un raisonnement.

Avantages connexes :

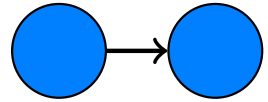
- Augmenter la clarté d'un raisonnement ;
- Oblige à réfléchir à certains détails ;
- Les détails ont leur importance.

Exemple : les graphes

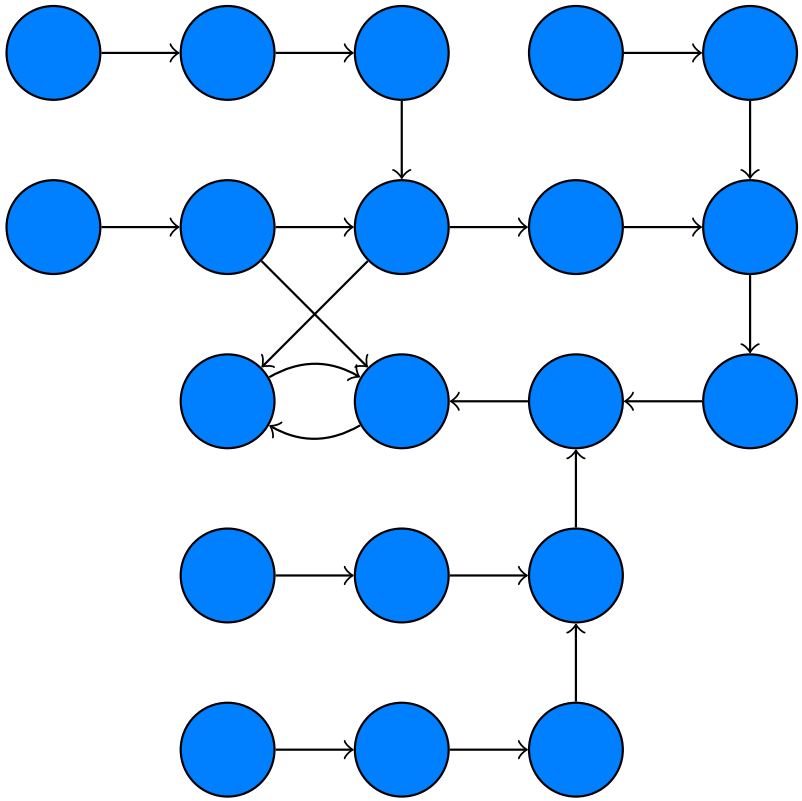


Nœuds :

Arcs :



Exemple : les graphes



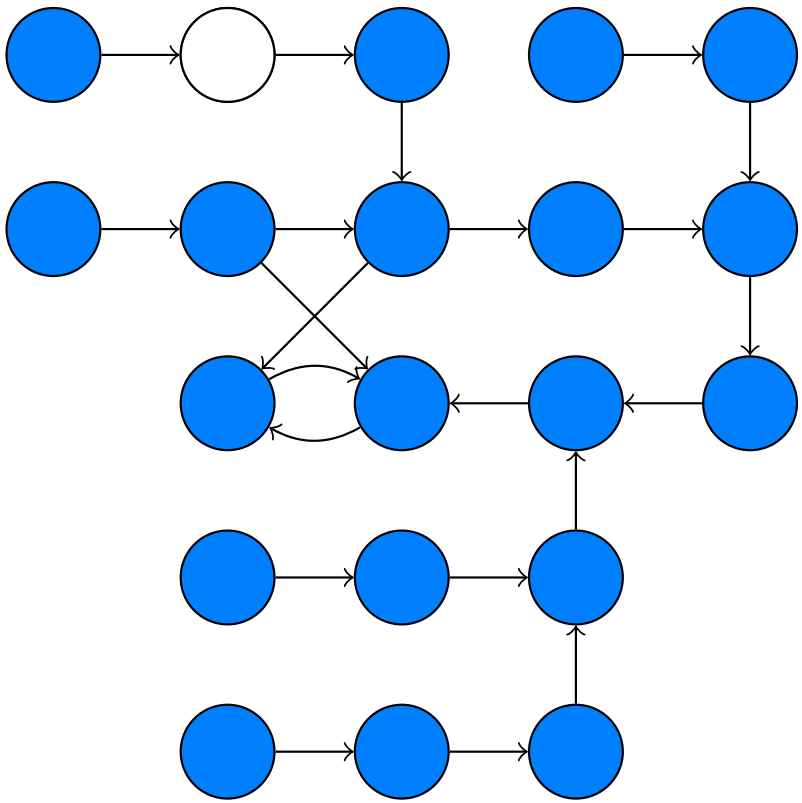
Nœuds :

Arcs :

Enfants( ) =

{ | }

Exemple : les graphes



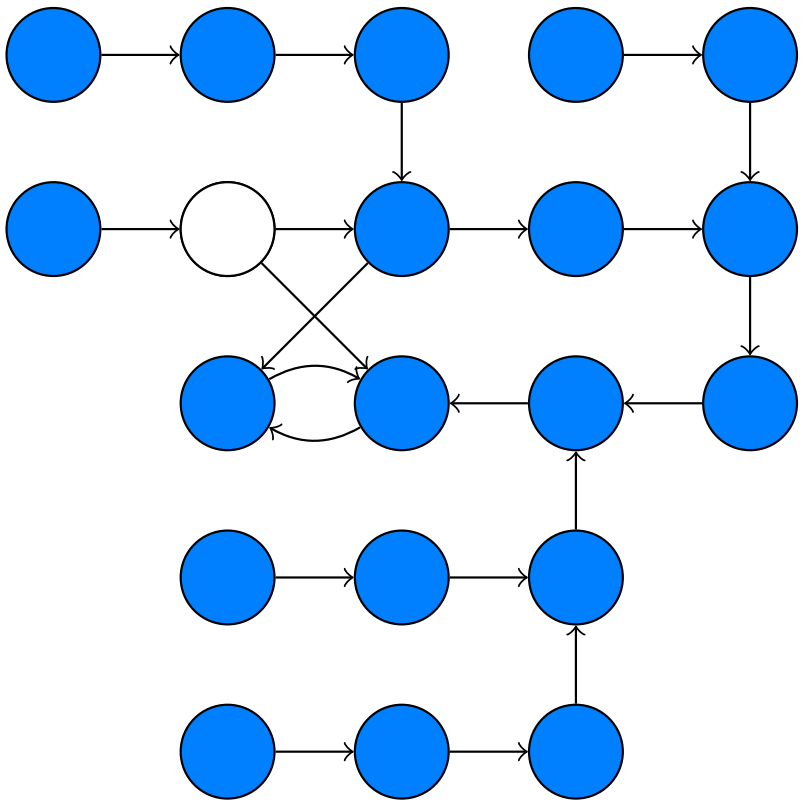
Nœuds :

Arcs :

Enfants( ) =

{ | }

Exemple : les graphes



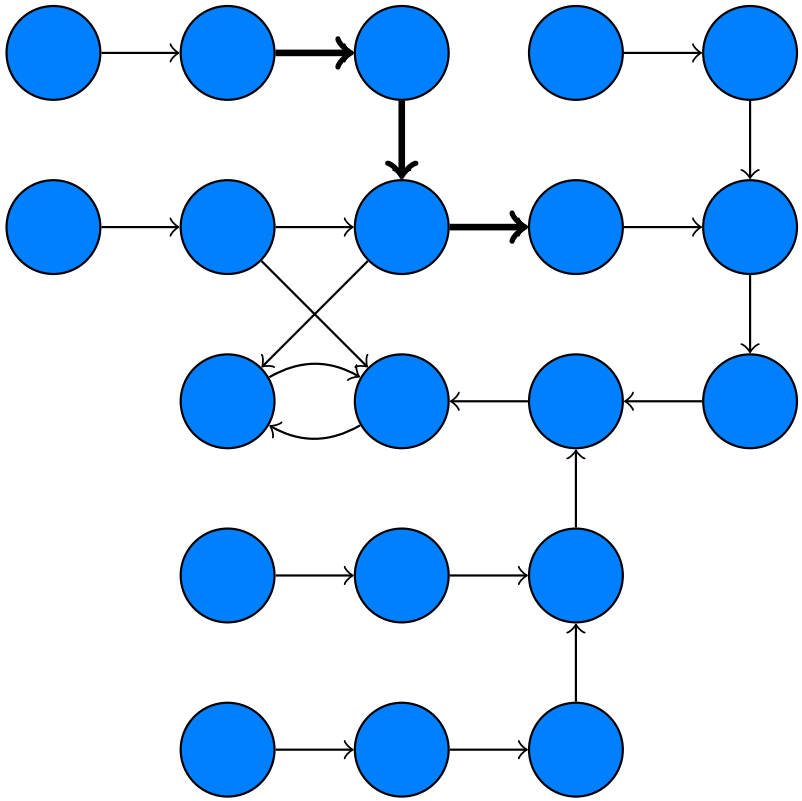
Nœuds :

Arcs :

Enfants( ) =

{ | }

Exemple : les graphes



Nœuds :

Arcs :

Enfants( ) =

{ | }

Chemin :

Algorithmes

Algorithme

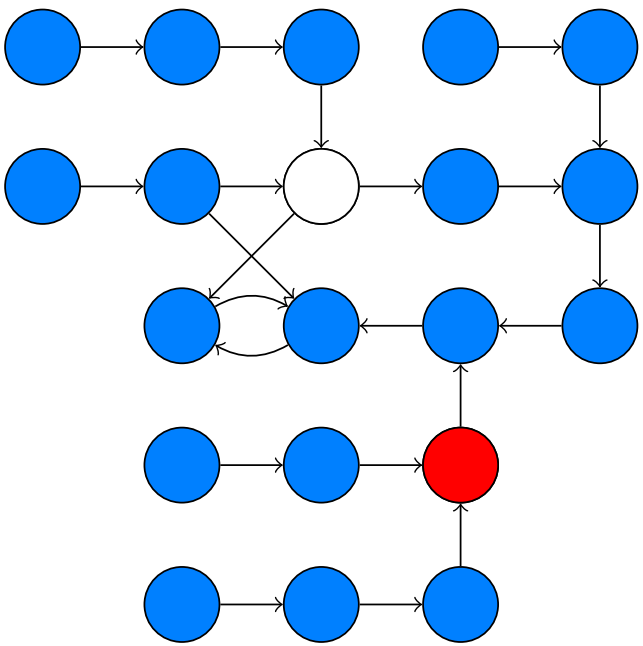
- Suite d'opérations élémentaires et non-ambigües.
- Répond à un problème.

Algorithmes

Algorithme

- Suite d'opérations élémentaires et non-ambigües.
- Répond à un problème.

Exemple : Existe-t-il un chemin entre ○ et ● ?

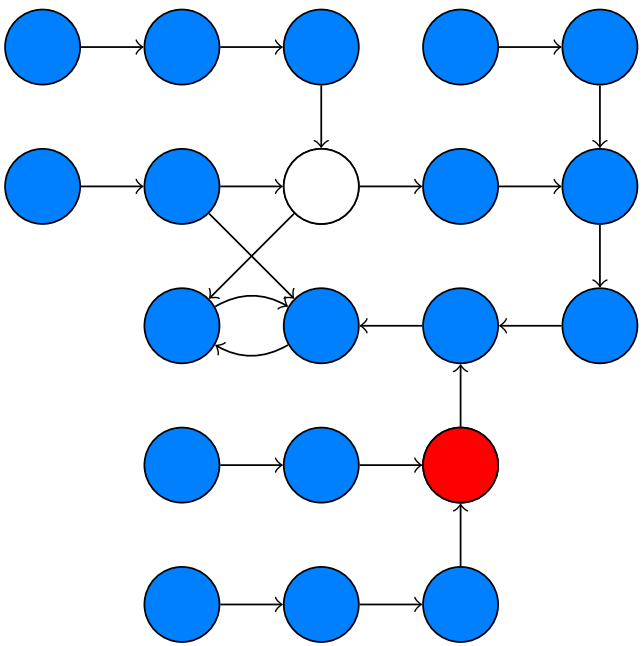


Algorithmes

Algorithme

- Suite d'opérations élémentaires et non-ambigües.
- Répond à un problème.

Exemple : Existe-t-il un chemin entre ○ et ● ?



Parcours en profondeur d'abord :

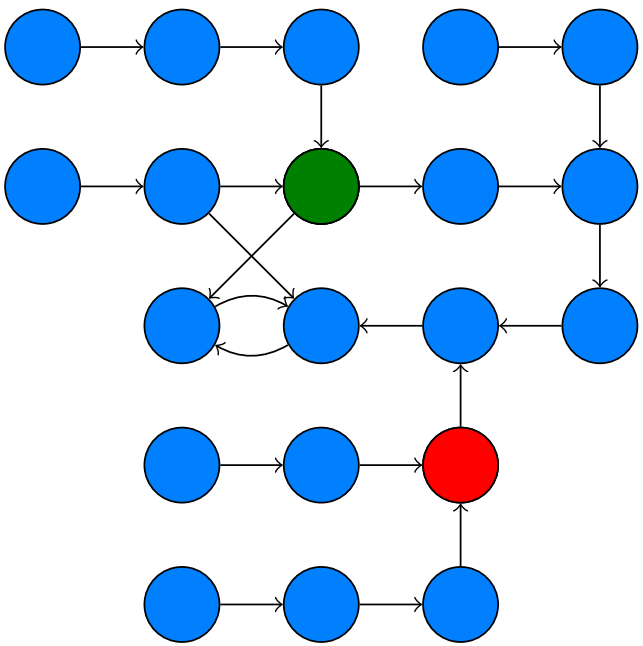
- *TODO* = ○ ; *FAIT* = ∅.
- tant que *TODO* n'est pas vide :
  - ● = *TODO*.*enlever dernier*
  - Si ● = ● : **répondre OUI**
  - *FAIT* = *FAIT* ∪ {●}.
  - pour tout ● ∈ Enfants(●) :
    - si ● ∉ *TODO* ∪ *FAIT* :
      TODO.*ajouter dernier* ●
- **répondre NON.**

Algorithmes

Algorithme

- Suite d'opérations élémentaires et non-ambigües.
- Répond à un problème.

Exemple : Existe-t-il un chemin entre ○ et ● ?



Parcours en profondeur d'abord :

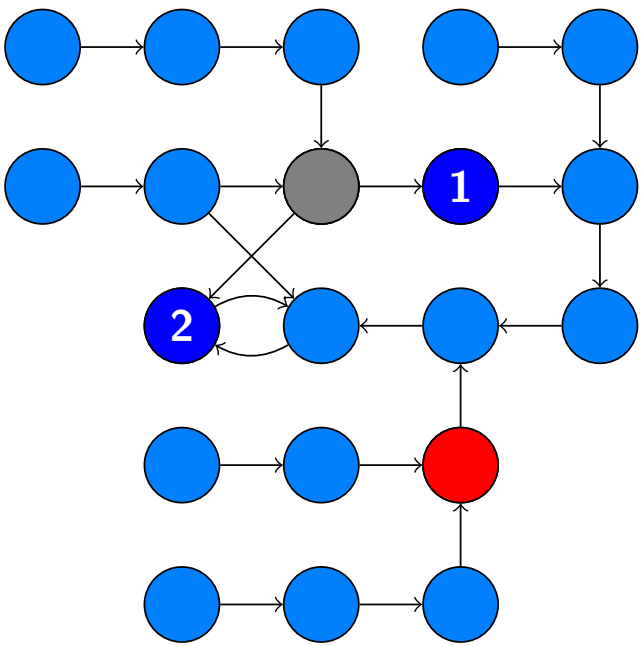
- *TODO* = ○ ; *FAIT* = ∅.
- tant que *TODO* n'est pas vide :
  - ● = *TODO*.*enlever dernier*
  - Si ● = ● : **répondre OUI**
  - *FAIT* = *FAIT* ∪ {●}.
  - pour tout ● ∈ Enfants(●) :
    - si ● ∉ *TODO* ∪ *FAIT* :
      TODO.*ajouter dernier* ●
- **répondre NON.**

Algorithmes

Algorithme

- Suite d'opérations élémentaires et non-ambigües.
- Répond à un problème.

Exemple : Existe-t-il un chemin entre ○ et ● ?



Parcours en profondeur d'abord :

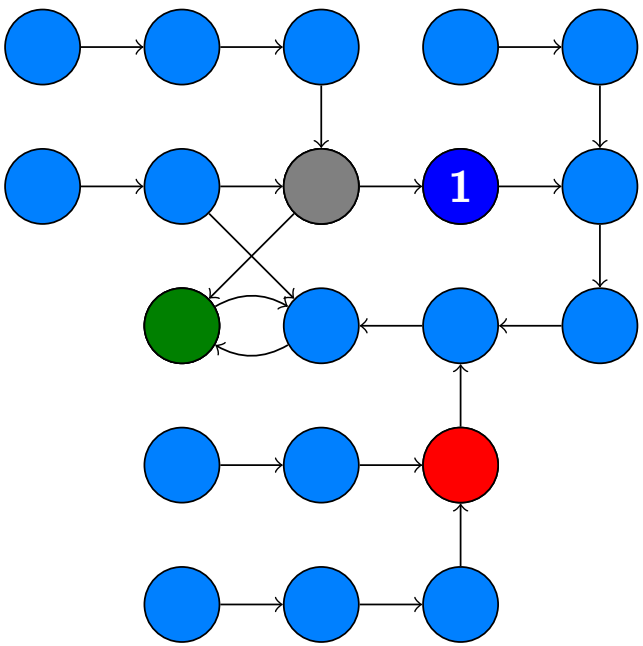
- *TODO* = ○ ; *FAIT* = ∅.
- tant que *TODO* n'est pas vide :
  - ● = *TODO*.*enlever dernier*
  - Si ● = ● : **répondre OUI**
  - *FAIT* = *FAIT* ∪ {●}.
  - pour tout ● ∈ Enfants(●) :
    - si ● ∉ *TODO* ∪ *FAIT* :
      TODO.*ajouter dernier* ●
- **répondre NON.**

Algorithmes

Algorithme

- Suite d'opérations élémentaires et non-ambigües.
- Répond à un problème.

Exemple : Existe-t-il un chemin entre ○ et ● ?



Parcours en profondeur d'abord :

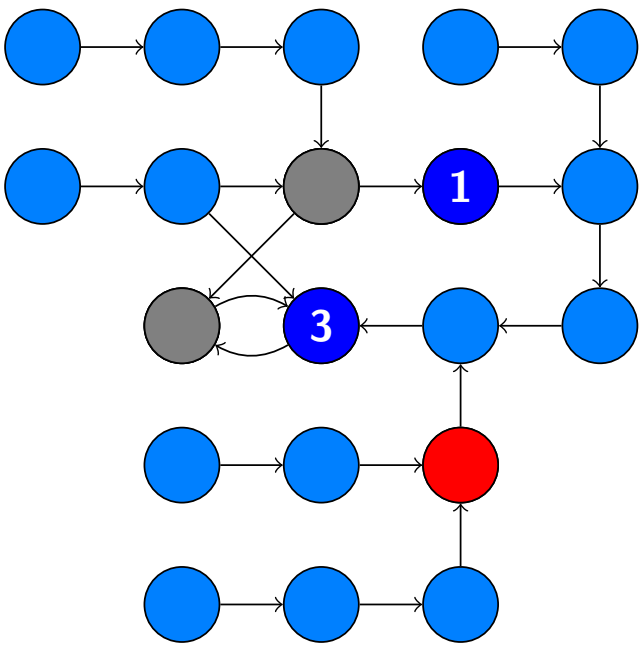
- *TODO* = ○ ; *FAIT* = ∅.
- tant que *TODO* n'est pas vide :
  - ● = *TODO*.*enlever dernier*
  - Si ● = ● : **répondre OUI**
  - *FAIT* = *FAIT* ∪ {●}.
  - pour tout ● ∈ Enfants(●) :
    - si ● ∉ *TODO* ∪ *FAIT* :
      TODO.*ajouter dernier* ●
- **répondre NON.**

Algorithmes

Algorithme

- Suite d'opérations élémentaires et non-ambigües.
- Répond à un problème.

Exemple : Existe-t-il un chemin entre ○ et ● ?



Parcours en profondeur d'abord :

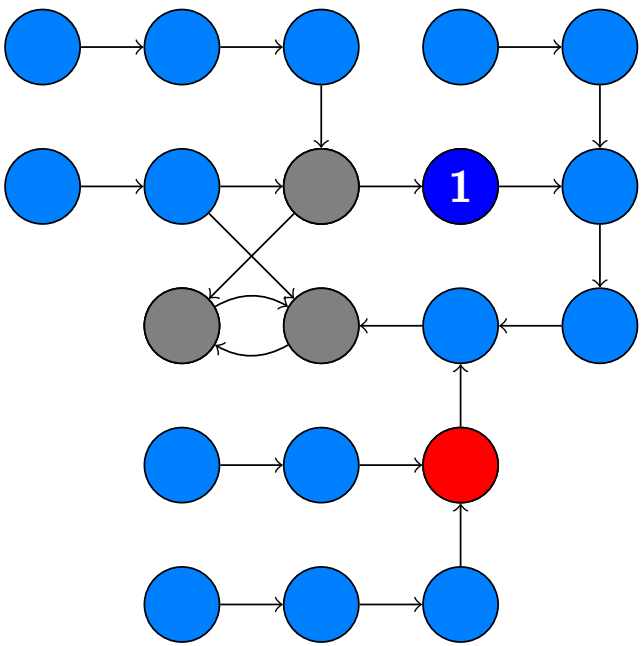
- *TODO* = ○ ; *FAIT* = ∅.
- tant que *TODO* n'est pas vide :
  - ● = *TODO*.*enlever dernier*
  - Si ● = ● : **répondre OUI**
  - *FAIT* = *FAIT* ∪ {●}.
  - pour tout ● ∈ Enfants(●) :
    - si ● ∉ *TODO* ∪ *FAIT* :
      TODO.*ajouter dernier* ●
- **répondre NON.**

Algorithmes

Algorithme

- Suite d'opérations élémentaires et non-ambigües.
- Répond à un problème.

Exemple : Existe-t-il un chemin entre ○ et ● ?



Parcours en profondeur d'abord :

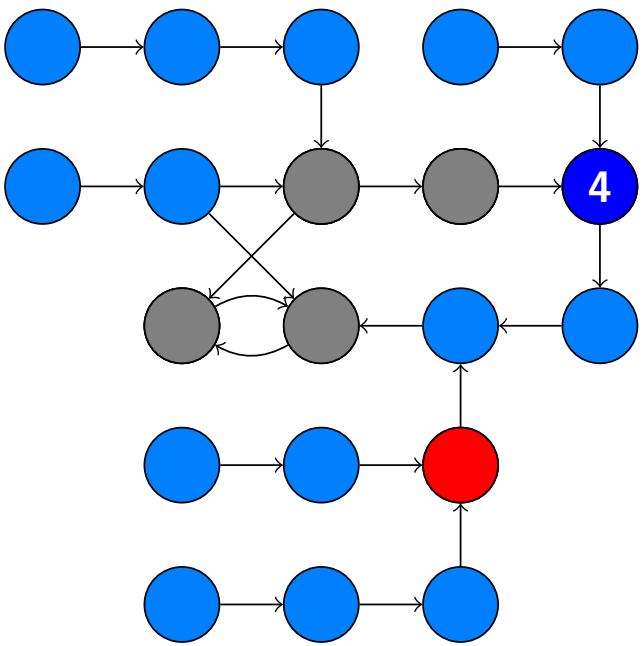
- *TODO* = ○ ; *FAIT* = ∅.
- tant que *TODO* n'est pas vide :
  - ● = *TODO*.*enlever dernier*
  - Si ● = ● : **répondre OUI**
  - *FAIT* = *FAIT* ∪ {●}.
  - pour tout ● ∈ Enfants(●) :
    - si ● ∉ *TODO* ∪ *FAIT* :
      TODO.*ajouter dernier* ●
- **répondre NON.**

Algorithmes

Algorithme

- Suite d'opérations élémentaires et non-ambigües.
- Répond à un problème.

Exemple : Existe-t-il un chemin entre ○ et ● ?



Parcours en profondeur d'abord :

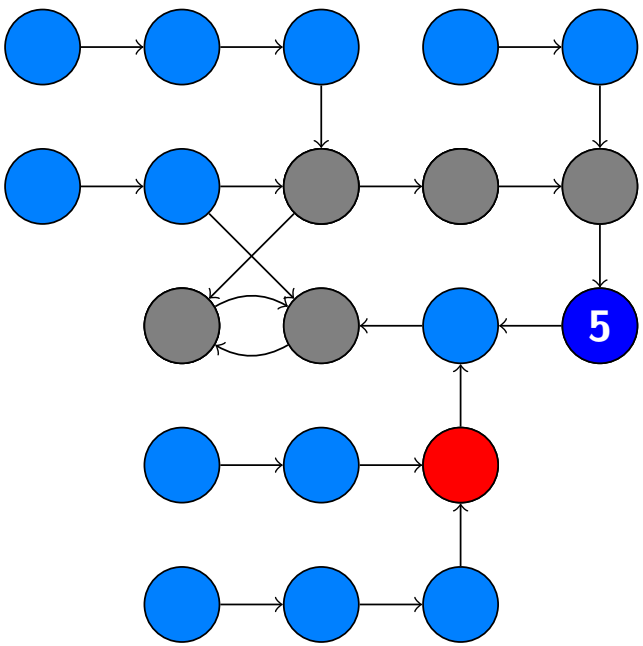
- *TODO* = ○ ; *FAIT* = ∅.
- tant que *TODO* n'est pas vide :
  - ● = *TODO*.*enlever dernier*
  - Si ● = ● : **répondre OUI**
  - *FAIT* = *FAIT* ∪ {●}.
  - pour tout ● ∈ Enfants(●) :
    - si ● ∉ *TODO* ∪ *FAIT* :
      TODO.*ajouter dernier* ●
- **répondre NON.**

Algorithmes

Algorithme

- Suite d'opérations élémentaires et non-ambigües.
- Répond à un problème.

Exemple : Existe-t-il un chemin entre ○ et ● ?



Parcours en profondeur d'abord :

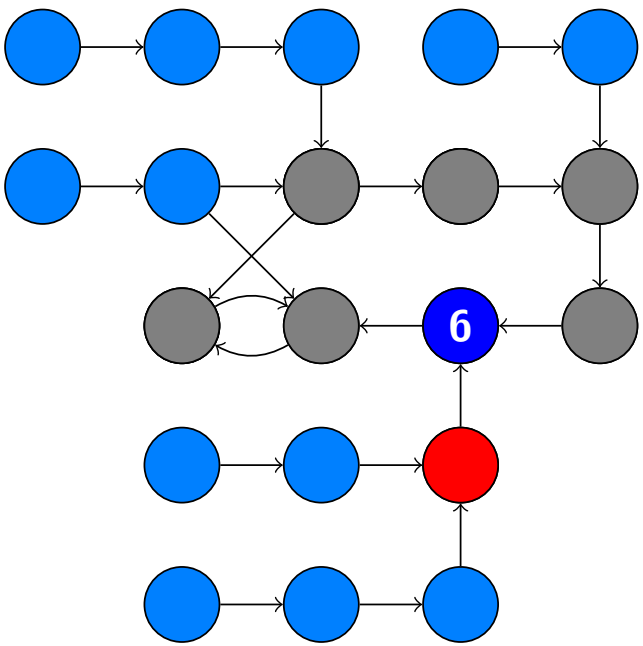
- *TODO* = ○ ; *FAIT* = ∅.
- tant que *TODO* n'est pas vide :
  - ● = *TODO*.*enlever dernier*
  - Si ● = ● : **répondre OUI**
  - *FAIT* = *FAIT* ∪ {●}.
  - pour tout ● ∈ Enfants(●) :
    - si ● ∉ *TODO* ∪ *FAIT* :
      TODO.*ajouter dernier* ●
- **répondre NON.**

Algorithmes

Algorithme

- Suite d'opérations élémentaires et non-ambigües.
- Répond à un problème.

Exemple : Existe-t-il un chemin entre ○ et ● ?



Parcours en profondeur d'abord :

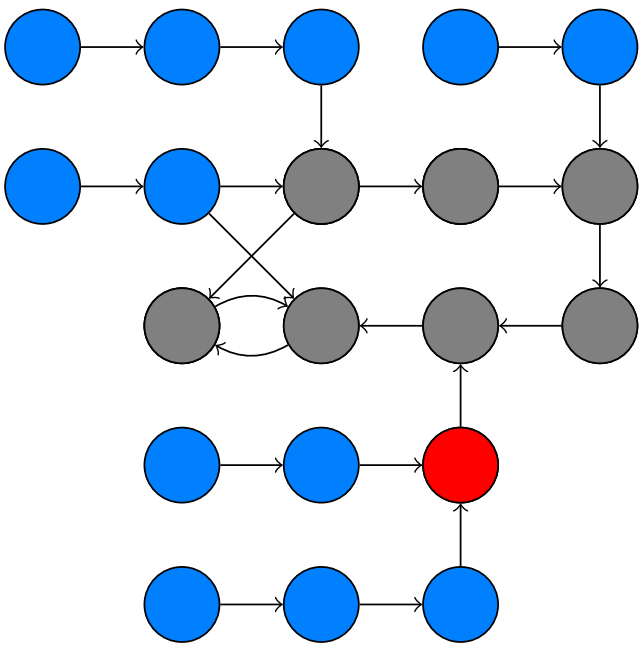
- *TODO* = ○ ; *FAIT* = ∅.
- tant que *TODO* n'est pas vide :
  - ● = *TODO*.*enlever dernier*
  - Si ● = ● : **répondre OUI**
  - *FAIT* = *FAIT* ∪ {●}.
  - pour tout ● ∈ Enfants(●) :
    - si ● ∉ *TODO* ∪ *FAIT* :
      TODO.*ajouter dernier* ●
- **répondre NON.**

Algorithmes

Algorithme

- Suite d'opérations élémentaires et non-ambigües.
- Répond à un problème.

Exemple : Existe-t-il un chemin entre ○ et ● ?



Parcours en profondeur d'abord :

- *TODO* = ○ ; *FAIT* = ∅.
- tant que *TODO* n'est pas vide :
  - ● = *TODO*.*enlever dernier*
  - Si ● = ● : **répondre OUI**
  - *FAIT* = *FAIT* ∪ {●}.
  - pour tout ● ∈ Enfants(●) :
    - si ● ∉ *TODO* ∪ *FAIT* :
      TODO.*ajouter dernier* ●
- **répondre NON.**

Complexité d'un algorithme

En fonction de la taille de l'entrée (p. ex. du graphe) :

- DANS LE PIRE DES CAS.
- Nombre d'opérations : complexité en temps.
- Espace requis : complexité en espace.

Algorithme précédent : temps $\approx$ taille du graphe (idem pour l'espace).

Pour schématiser :

- Algorithmes efficaces : taille entrée x 2 $\Rightarrow$ complexité x 2.
  Entrée de taille 100, temps : 100s ;
  Entrée de taille 200, temps : 200s.
- Algorithmes exponentiels : taille entrée x 2 $\Rightarrow$ complexité^2.
  Entrée de taille 100, temps : 100s ;
  Entrée de taille 200, temps : 10000s ($\approx$ 3h).

Problèmes et Complexités

Un problème ⇒ 0, 1, ou plusieurs algorithmes possibles.

Pour certains problèmes :

- Nous ne connaissons pas d'algorithme efficace.
- Nous savons qu'il n'existe pas d'algorithme efficace.
- Nous savons qu'ils sont indécidables (pas d'algorithme possible).

Autres critères :

- Complexité en moyenne (dépend de l'application).
- Compromis entre la complexité en espace et en temps.

Langage et sémantique

But : exprimer de manière compacte un objet (graphe, etc.)

Programme

1. $n : \{1, 2, \ldots, 20\}$
2. n dizaine $\rightarrow n/10$
3. n pair $\rightarrow n/2$
4. n impair $\rightarrow n + 1$

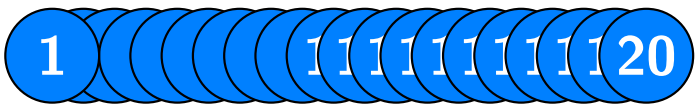
Langage et sémantique

But : exprimer de manière compacte un objet (graphe, etc.)

Programme

1. $n : \{1, 2, \ldots, 20\}$
2. n dizaine $\rightarrow n/10$
3. n pair $\rightarrow n/2$
4. n impair $\rightarrow n + 1$

Sémantique



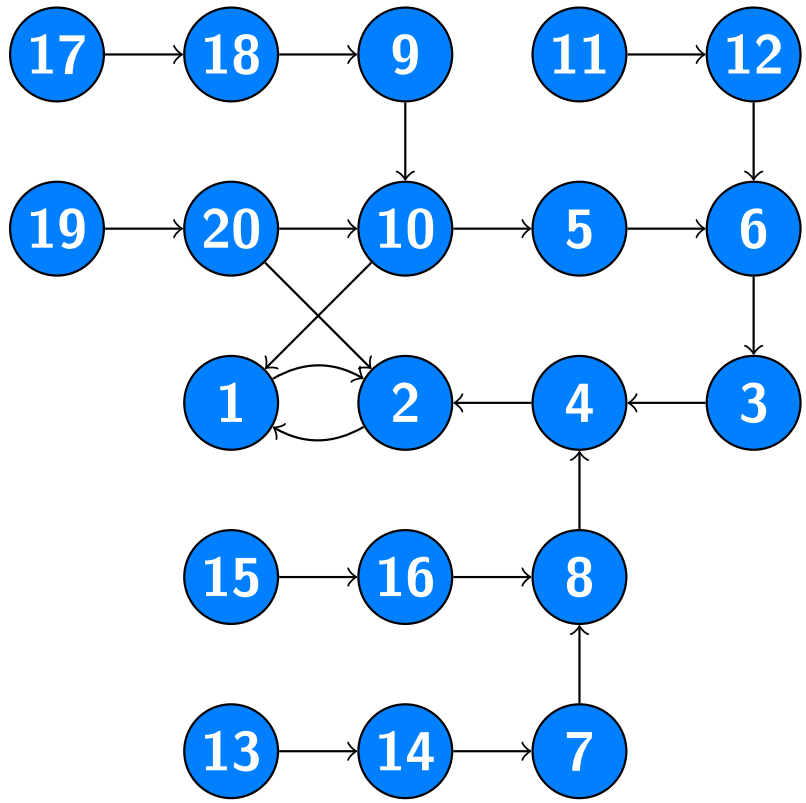
$$\frac{cond(n) \rightarrow f(n) \quad cond(n) \wedge m = f(n)}{n \rightarrow m}$$

Langage et sémantique

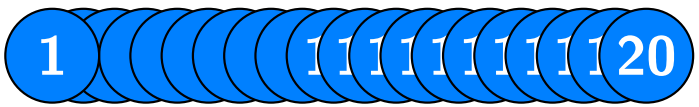
But : exprimer de manière compacte un objet (graphe, etc.)

Programme

1. $n : \{1, 2, \ldots, 20\}$
2. n dizaine $\rightarrow n/10$
3. n pair $\rightarrow n/2$
4. n impair $\rightarrow n + 1$



Sémantique



$$\frac{cond(n) \rightarrow f(n) \quad cond(n) \wedge m = f(n)}{n \rightarrow m}$$

Plan

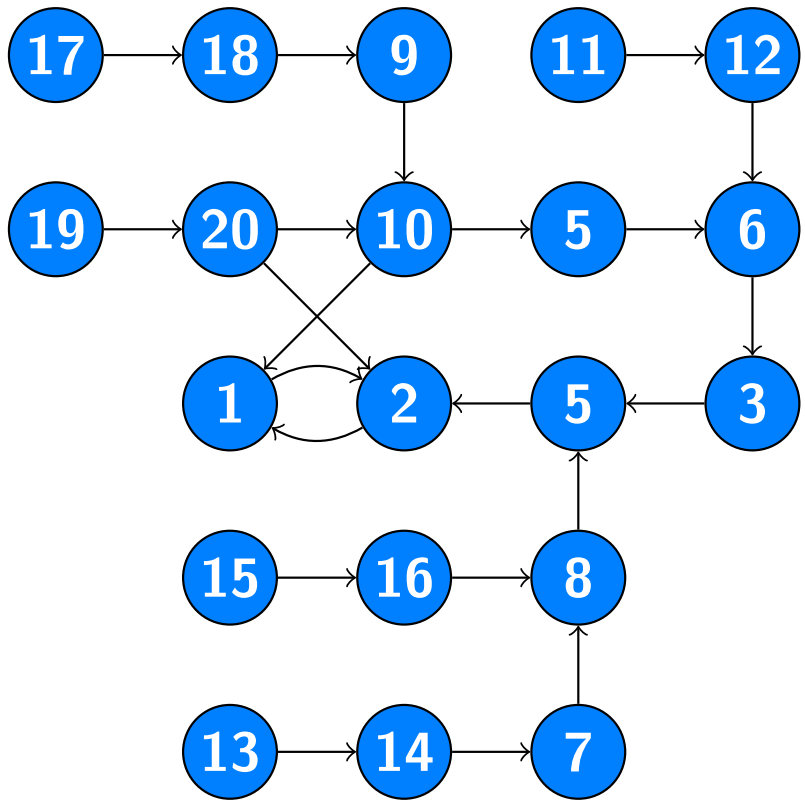
Systèmes de transition d'états discrets

État discret

- Nœud d'un graphe.
- P. ex. : valeur de tous les composants du système : nb. de molécules ; état qualitatif (absent, un peu, beaucoup) ; etc.

Transition

- Arc d'un graphe.
- Le système change d'état.
- Non-déterminisme.



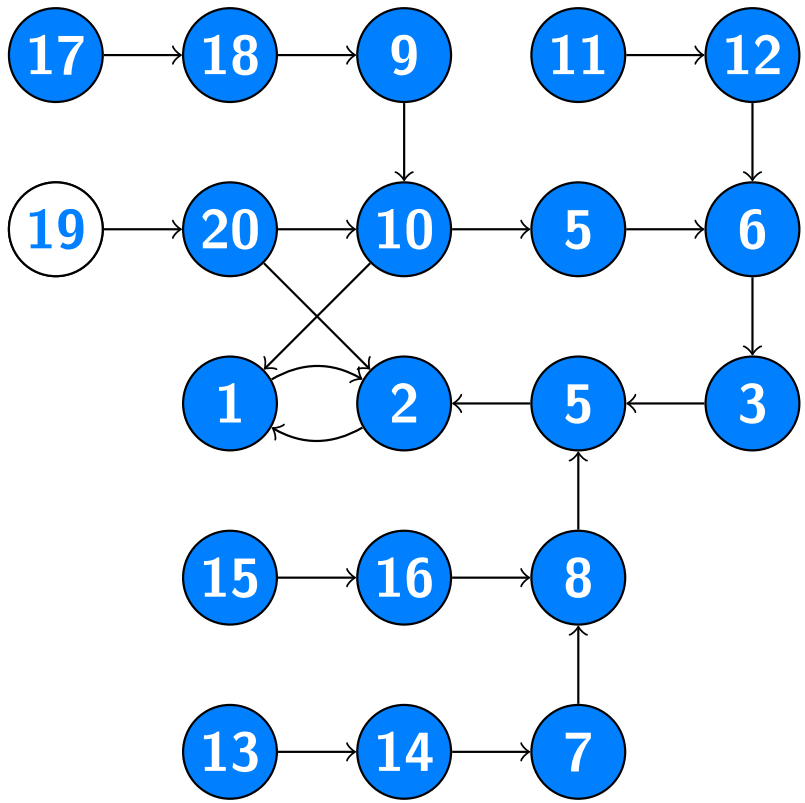
Systèmes de transition d'états discrets

État discret

- Nœud d'un graphe.
- P. ex. : valeur de tous les composants du système : nb. de molécules ; état qualitatif (absent, un peu, beaucoup) ; etc.

Transition

- Arc d'un graphe.
- Le système change d'état.
- Non-déterminisme.



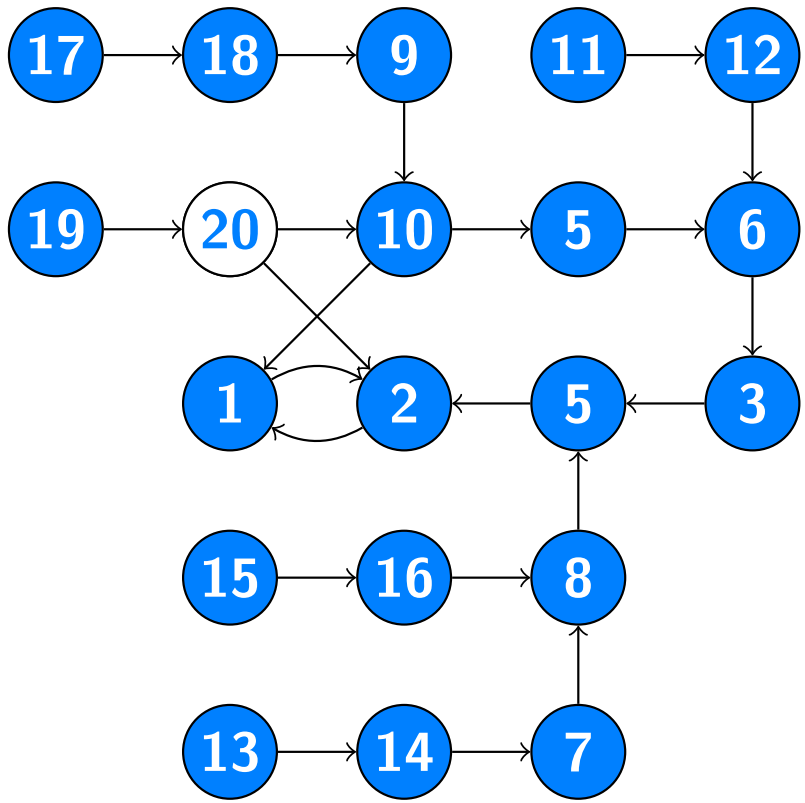
Systèmes de transition d'états discrets

État discret

- Nœud d'un graphe.
- P. ex. : valeur de tous les composants du système : nb. de molécules ; état qualitatif (absent, un peu, beaucoup) ; etc.

Transition

- Arc d'un graphe.
- Le système change d'état.
- Non-déterminisme.



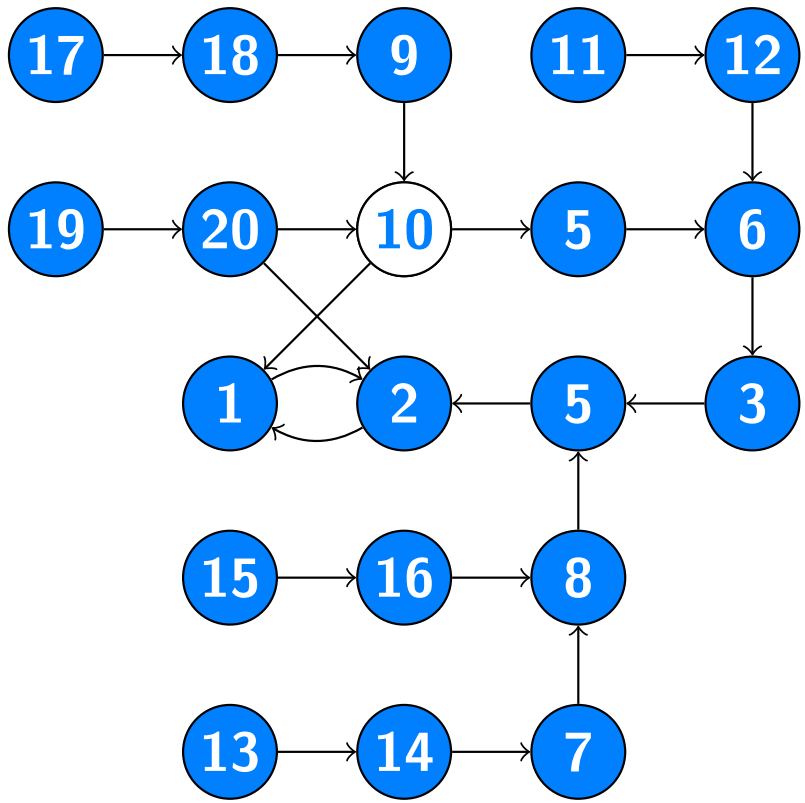
Systèmes de transition d'états discrets

État discret

- Nœud d'un graphe.
- P. ex. : valeur de tous les composants du système : nb. de molécules ; état qualitatif (absent, un peu, beaucoup) ; etc.

Transition

- Arc d'un graphe.
- Le système change d'état.
- Non-déterminisme.



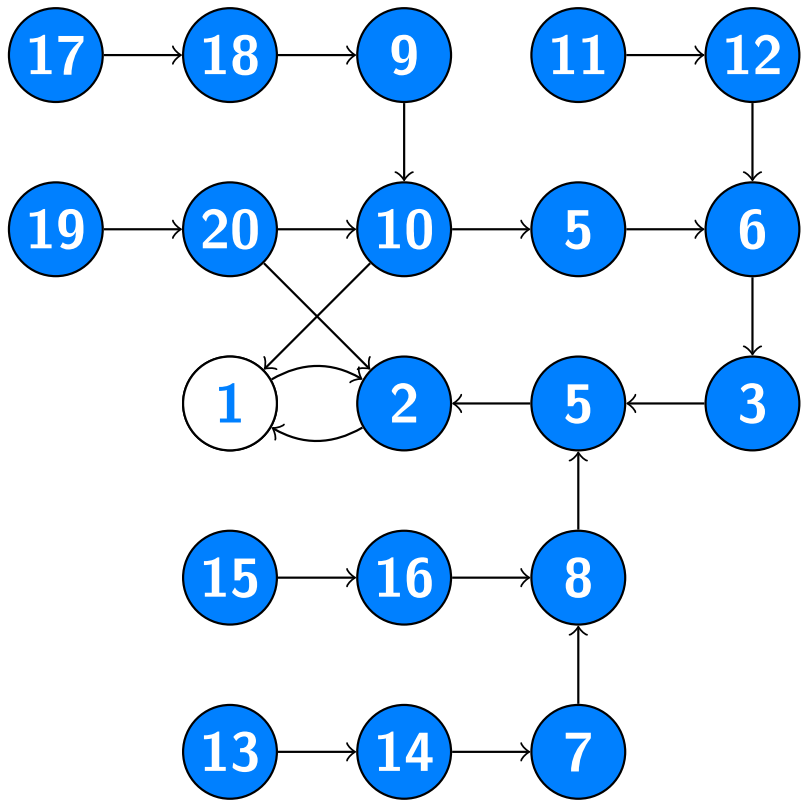
Systèmes de transition d'états discrets

État discret

- Nœud d'un graphe.
- P. ex. : valeur de tous les composants du système : nb. de molécules ; état qualitatif (absent, un peu, beaucoup) ; etc.

Transition

- Arc d'un graphe.
- Le système change d'état.
- Non-déterminisme.



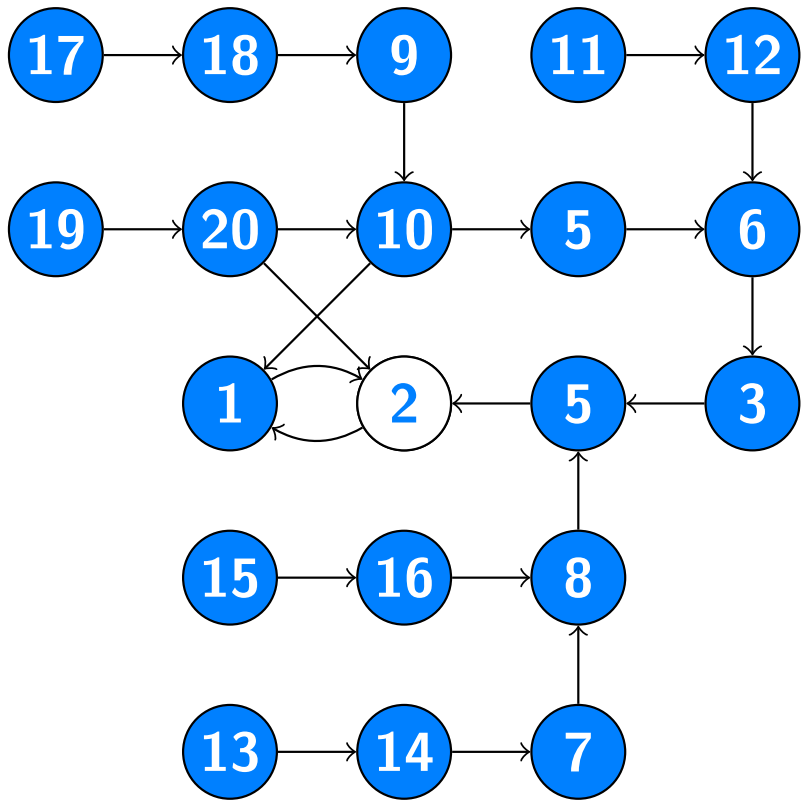
Systèmes de transition d'états discrets

État discret

- Nœud d'un graphe.
- P. ex. : valeur de tous les composants du système : nb. de molécules ; état qualitatif (absent, un peu, beaucoup) ; etc.

Transition

- Arc d'un graphe.
- Le système change d'état.
- Non-déterminisme.



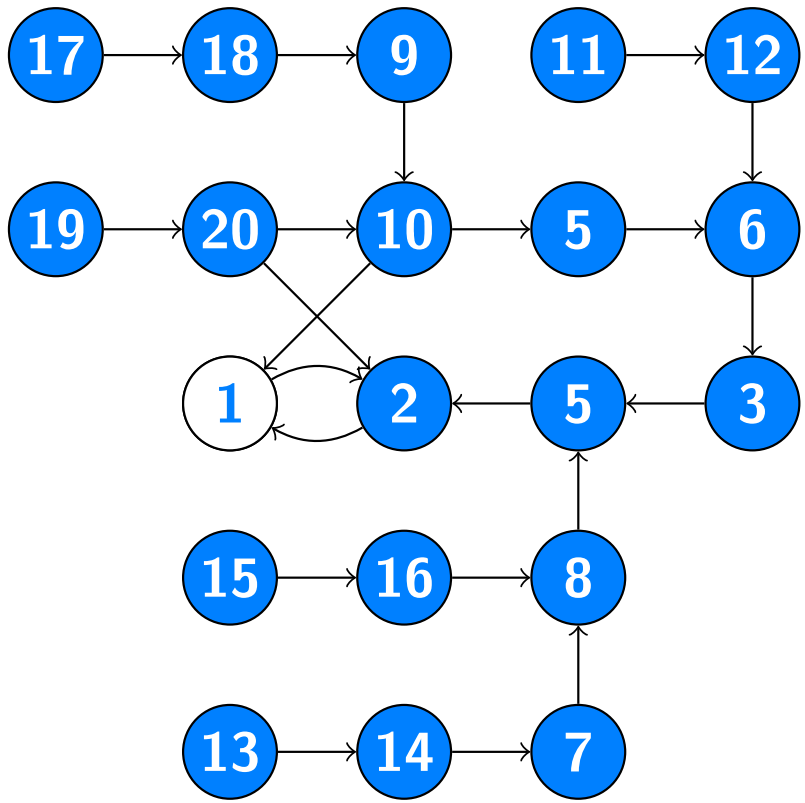
Systèmes de transition d'états discrets

État discret

- Nœud d'un graphe.
- P. ex. : valeur de tous les composants du système : nb. de molécules ; état qualitatif (absent, un peu, beaucoup) ; etc.

Transition

- Arc d'un graphe.
- Le système change d'état.
- Non-déterminisme.



Autre exemple de modèle

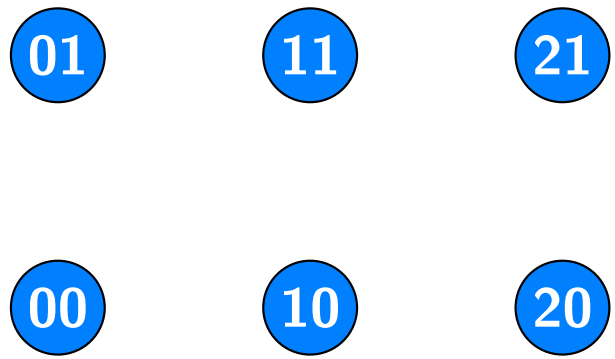
Programme

1. $a : \{0, 1, 2\}$
2. $b : \{0, 1\}$
3. si $b = 0 : a+$
4. si $b = 1$ et $a < 2 : a-$
5. si $a > 0 : b+$
6. si $a = 0 : b-$

Autre exemple de modèle

Programme

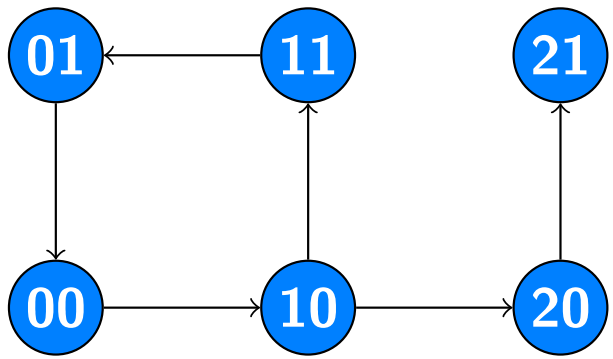
1. $a : \{0, 1, 2\}$
2. $b : \{0, 1\}$
3. si $b = 0 : a+$
4. si $b = 1$ et $a < 2 : a-$
5. si $a > 0 : b+$
6. si $a = 0 : b-$



Autre exemple de modèle

Programme

1. $a : \{0, 1, 2\}$
2. $b : \{0, 1\}$
3. si $b = 0 : a+$
4. si $b = 1$ et $a < 2 : a-$
5. si $a > 0 : b+$
6. si $a = 0 : b-$



Utilité pour l'étude du système *réel*

Généralement, modèle = approximation supérieure du système réel.

Prouver l'existence ou l'absence de comportements (suite d'états)

- Simulation : exemple (non-déterministe) de comportement.
- Vérification formelle (exhaustivité de l'analyse) : prouve notamment l'absence de comportement.

Comprendre l'émergence de certains comportements

- Analyse de causalité : trouver les responsables.
- Guide le raffinement du modèle, ou le contrôle du système réel.

Les défis

Étant donné le graphe des transitions, nous voulons :

- Vérifier l'existence/l'absence de comportements (atteignabilités) ;
- rechercher l'existence de cycles limites ;
- rechercher les causes d'un comportement ; etc.

En pratique, les propriétés recherchées se prouvent efficacement.

Les défis

Étant donné le graphe des transitions, nous voulons :

- Vérifier l'existence/l'absence de comportements (atteignabilités) ;
- rechercher l'existence de cycles limites ;
- rechercher les causes d'un comportement ; etc.

En pratique, les propriétés recherchées se prouvent efficacement.

Problème : taille du graphe exponentielle selon le nb de composants.

$\Rightarrow$ explosion combinatoire des comportements.
Même si les algorithmes sont efficaces, le graphe devient trop grand.

Les défis

Étant donné le graphe des transitions, nous voulons :

- Vérifier l'existence/l'absence de comportements (atteignabilités) ;
- rechercher l'existence de cycles limites ;
- rechercher les causes d'un comportement ; etc.

En pratique, les propriétés recherchées se prouvent efficacement.

Problème : taille du graphe exponentielle selon le nb de composants.

⇒ explosion combinatoire des comportements.
Même si les algorithmes sont efficaces, le graphe devient trop grand.

⇒ nécessité de simplifier un modèle.
⇒ trouver l'explication la plus simple.

Abstraction des systèmes de transition

Intuition

- Espace d'état trop grand (voire infini)
- $\Rightarrow$ grouper (partitionner) les états.

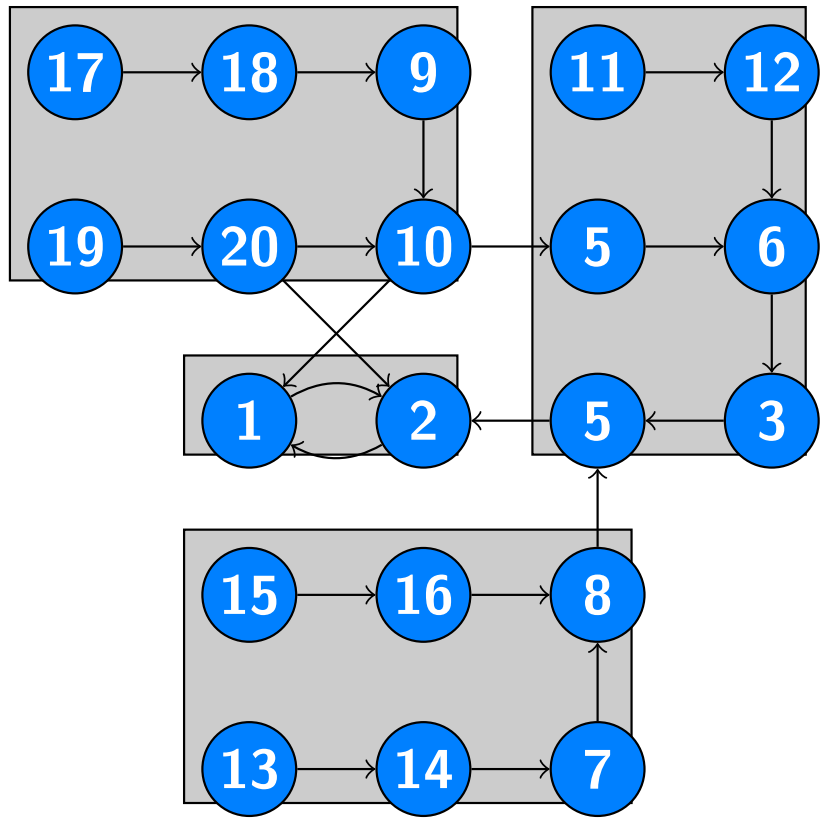
Abstraction des systèmes de transition

Intuition

- Espace d'état trop grand (voire infini)
- $\Rightarrow$ grouper (partitionner) les états.

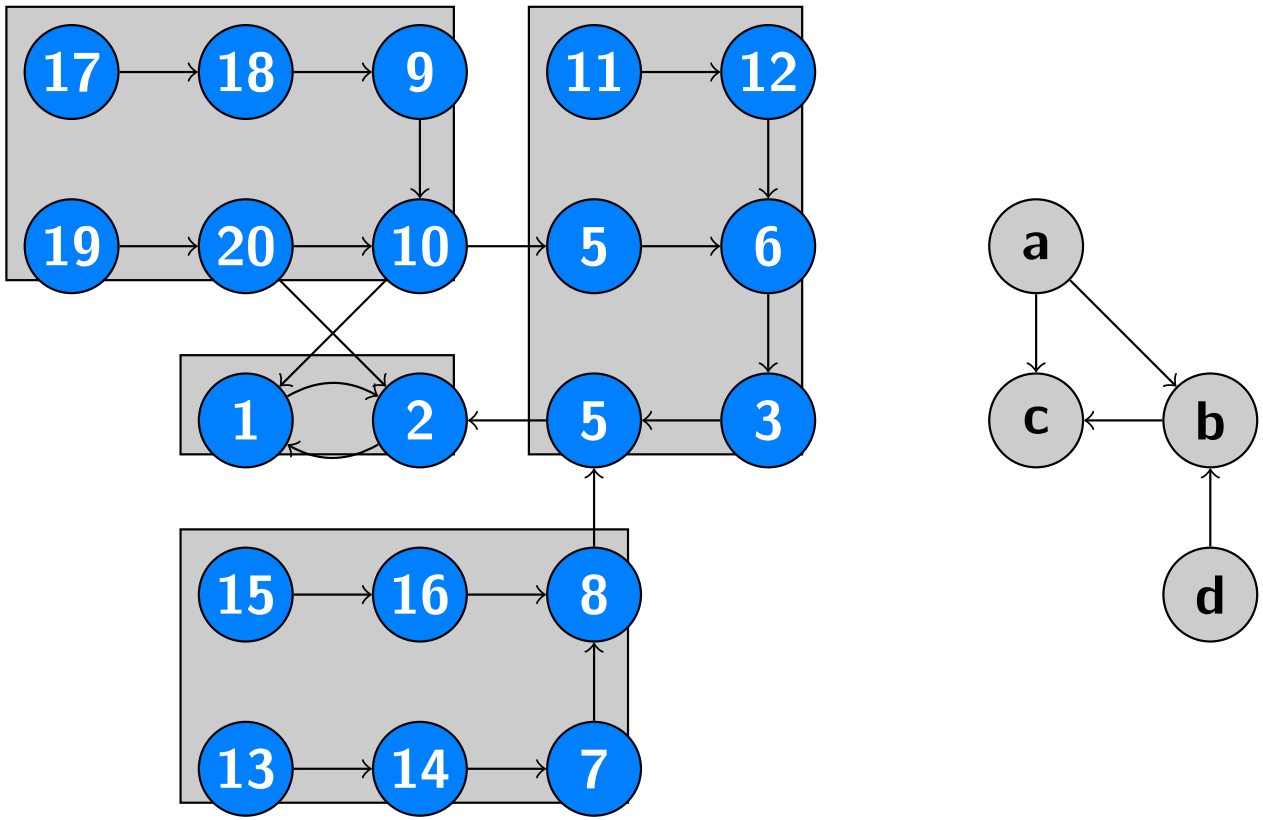
Processus d'abstraction

- Un groupe = état abstrait ;
- $\Rightarrow$ transitions ?
- Plusieurs stratégies : existentielle, universelle, etc.



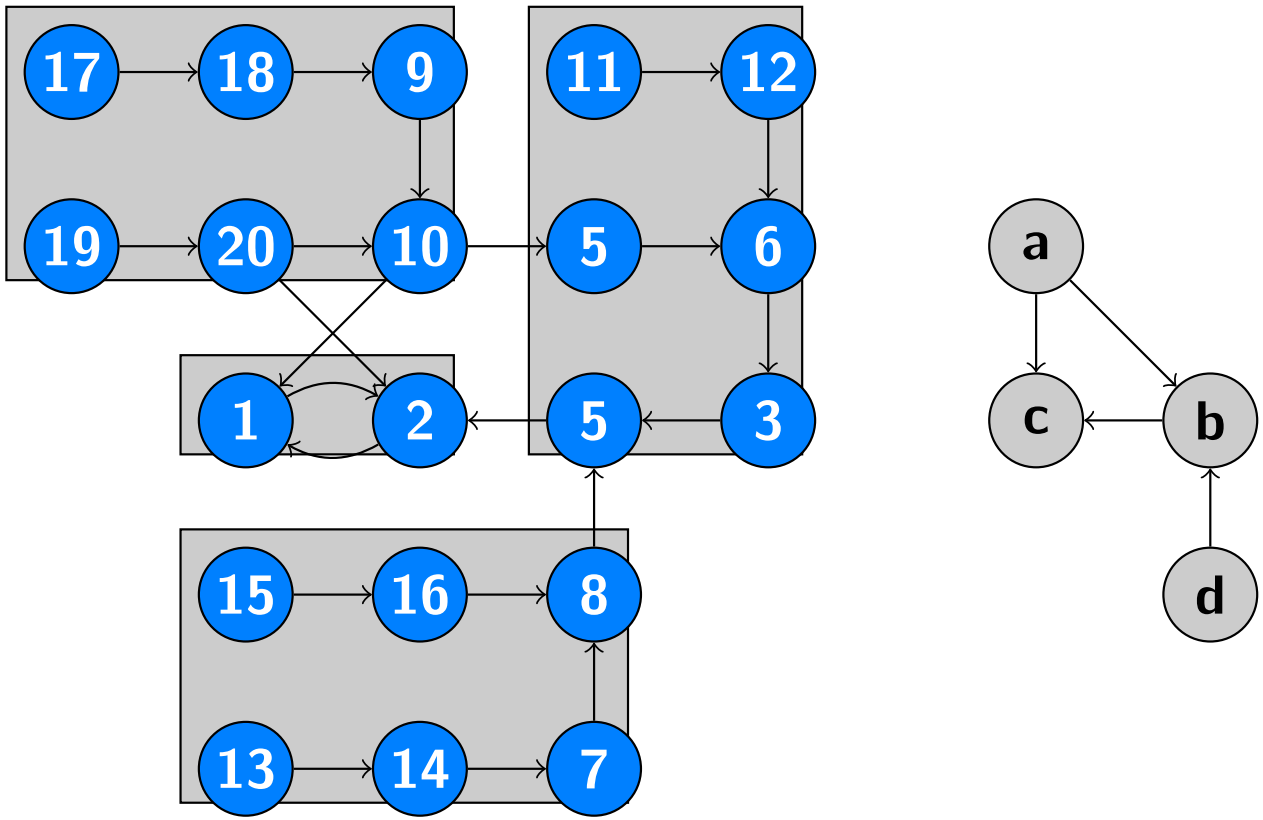
Abstraction des systèmes de transition

Intuition



Abstraction des systèmes de transition

Intuition

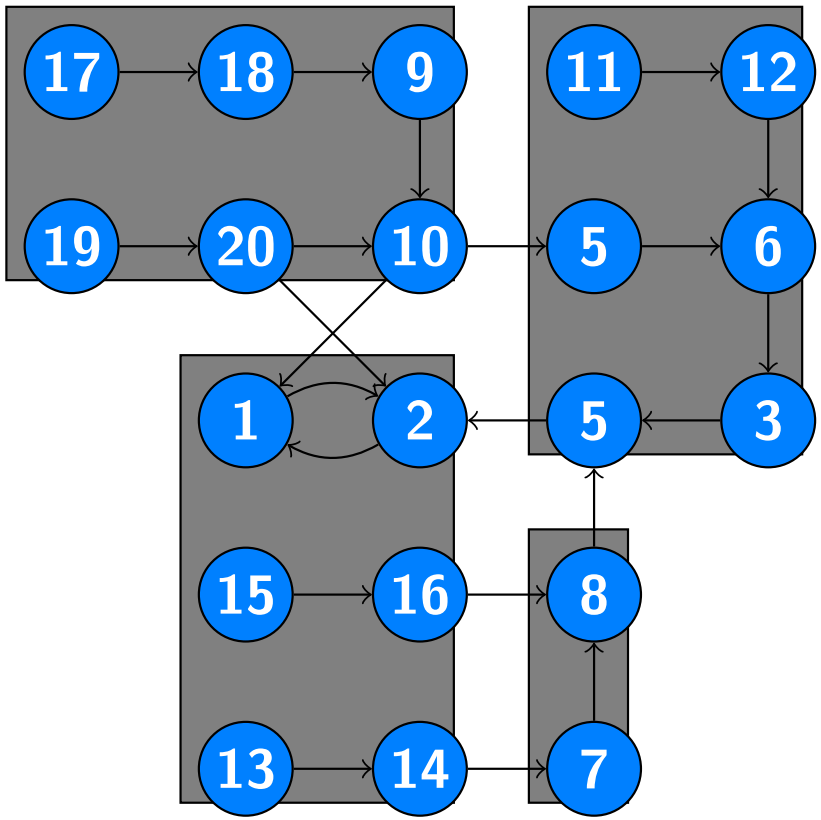


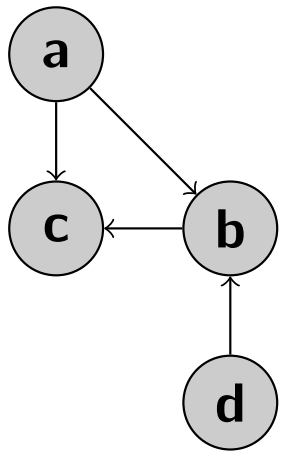
Impact sur les comportements (et donc les preuves)

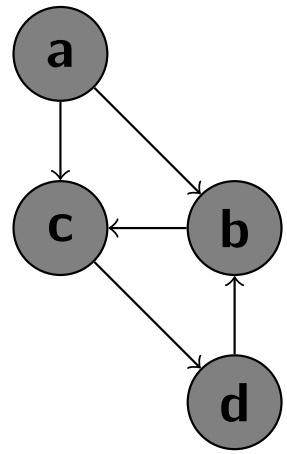
- Tout comportement concret se retrouve dans le modèle abstrait.
- La réciproque n'est généralement pas vraie.

Abstraction des systèmes de transition

Intuition







Impact sur les comportements (et donc les preuves)

- Tout comportement concret se retrouve dans le modèle abstrait.
- La réciproque n'est généralement pas vraie.

Abstraction des systèmes de transition

Problématiques associées

- Validité des abstractions (théorie).
- Efficacité d'une abstraction.
- Perte d'information.

Comment choisir une bonne abstraction ?

$\Rightarrow$ dépend de la question à répondre.

Abstraction des systèmes de transition

Illustration

But : étudier l'évolution du signe de l'état.

Programme

1. $n : \{-1000 \text{ milliard}, \ldots, 1000 \text{ milliard}\}$
2. n dizaine $\rightarrow n/10$
3. n pair $\rightarrow n/2$
4. n impair $\rightarrow n + 1$

Abstraction des systèmes de transition

Illustration

But : étudier l'évolution du signe de l'état.

Programme

1. $n : \{-1000 \text{ milliard}, \ldots, 1000 \text{ milliard}\}$
2. n dizaine $\rightarrow n/10$
3. n pair $\rightarrow n/2$
4. n impair $\rightarrow n+1$

Programme abstrait

1. $n^\# : \{-, 0, +\}$
2. $n^\# \rightarrow n^\#$
3. $n^\# \rightarrow n^\#$
4. $n^\#$ est $- \rightarrow 0$
   $n^\#$ est $- \rightarrow n^\#$
   $n^\#$ est $+ \rightarrow n^\#$

Abstraction des systèmes de transition

Illustration

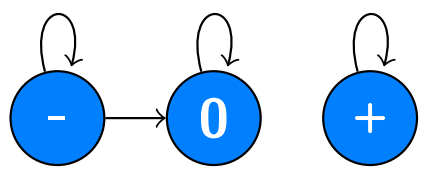
But : étudier l'évolution du signe de l'état.

Programme

1. $n : \{-1000 \text{ milliard}, \ldots, 1000 \text{ milliard}\}$
2. n dizaine $\rightarrow n/10$
3. n pair $\rightarrow n/2$
4. n impair $\rightarrow n+1$

Programme abstrait

1. $n^\# : \{-, 0, +\}$
2. $n^\# \rightarrow n^\#$
3. $n^\# \rightarrow n^\#$
4. $n^\#$ est $- \rightarrow 0$
   $n^\#$ est $- \rightarrow n^\#$
   $n^\#$ est $+ \rightarrow n^\#$



Abstraction des systèmes de transition

La théorie

L'interprétation abstraite (Cousot & Cousot)

- Cadre formel générique pour manipuler et prouver les abstractions.
- Abstractions automatiques.
- Propriétés remarquables et leurs conséquences.
- Liens entre le système concret et le système abstrait.

Interprétation abstraite

Idées générales

Nous nous fixons :

- un domaine concret D ;
  p. ex. $\{\text{arbre}, \text{banane}, \text{cheval}, \text{voiture}, \ldots\}$.
- un domaine abstrait $D^{\#}$;
  p. ex. $\{1, 2, 3, \ldots\}$.

Interprétation abstraite

Idées générales

Nous nous fixons :

- un domaine concret D ;
  p. ex. $\{\text{arbre}, \text{banane}, \text{cheval}, \text{voiture}, \ldots\}$.
- un domaine abstrait $D^{\#}$;
  p. ex. $\{1, 2, 3, \ldots\}$.

Nous définissons alors :

- une fonction d'abstraction α ; p. ex. : longueur des mots
  $\alpha(\{\text{arbre}\}) = \{5\}$ $\quad$ $\alpha(\{\text{banane}\}) = \{6\}$
  $\alpha(\{\text{un}, \text{cheval}, \text{mange}, \text{une}, \text{banane}\}) = \{2, 3, 5, 6\}$

Interprétation abstraite

Idées générales

Nous nous fixons :

- un domaine concret D ;
  p. ex. $\{\text{arbre}, \text{banane}, \text{cheval}, \text{voiture}, \ldots\}$.
- un domaine abstrait $D^{\#}$;
  p. ex. $\{1, 2, 3, \ldots\}$.

Nous définissons alors :

- une fonction d'abstraction α ; p. ex. : longueur des mots
  $\alpha(\{\text{arbre}\}) = \{5\}$ $\quad$ $\alpha(\{\text{banane}\}) = \{6\}$
  $\alpha(\{\text{un}, \text{cheval}, \text{mange}, \text{une}, \text{banane}\}) = \{2, 3, 5, 6\}$
- une fonction de concrétisation γ :
  $\gamma(\{6\}) = \{\text{banane}, \text{cheval}, \text{machin}, \ldots\}$

Interprétation abstraite

Idées générales (suite)

$\alpha(\{\text{un}, \text{cheval}, \text{mange}, \text{une}, \text{banane}\}) = \{2, 3, 5, 6\}$
$\gamma(\{6\}) = \{\text{banane}, \text{cheval}, \text{machin}, \ldots\}$

Propriétés de base (d est un sous-ensemble de D, $d^\#$ de $D^\#$)

- $\alpha(d) \subseteq \alpha(d \cup \ldots)$;
- $\gamma(d^\#) \subseteq \gamma(d^\# \cup \ldots)$.

Interprétation abstraite

Idées générales (suite)

$\alpha(\{\text{un}, \text{cheval}, \text{mange}, \text{une}, \text{banane}\}) = \{2, 3, 5, 6\}$
$\gamma(\{6\}) = \{\text{banane}, \text{cheval}, \text{machin}, \ldots\}$

Propriétés de base (d est un sous-ensemble de D, $d^\#$ de $D^\#$)

- $\alpha(d) \subseteq \alpha(d \cup \ldots)$;
- $\gamma(d^\#) \subseteq \gamma(d^\# \cup \ldots)$.

Propriétés remarquables

- $d \subseteq \gamma(\alpha(d))$;
- $\alpha(\gamma(d^\#)) \subseteq d^\#$.

$\Rightarrow$ connexion de Galois entre D et $D^\#$.

Plan

Interprétation abstraite

Applications en bio-informatique

L'interprétation abstraite permet (formellement) de :

- Simplifier, réduire un modèle.
- Réduire la complexité de certains calculs.
- Conserver un lien avec le modèle original (interprétation des preuves, etc).

Interprétation abstraite

Applications en bio-informatique

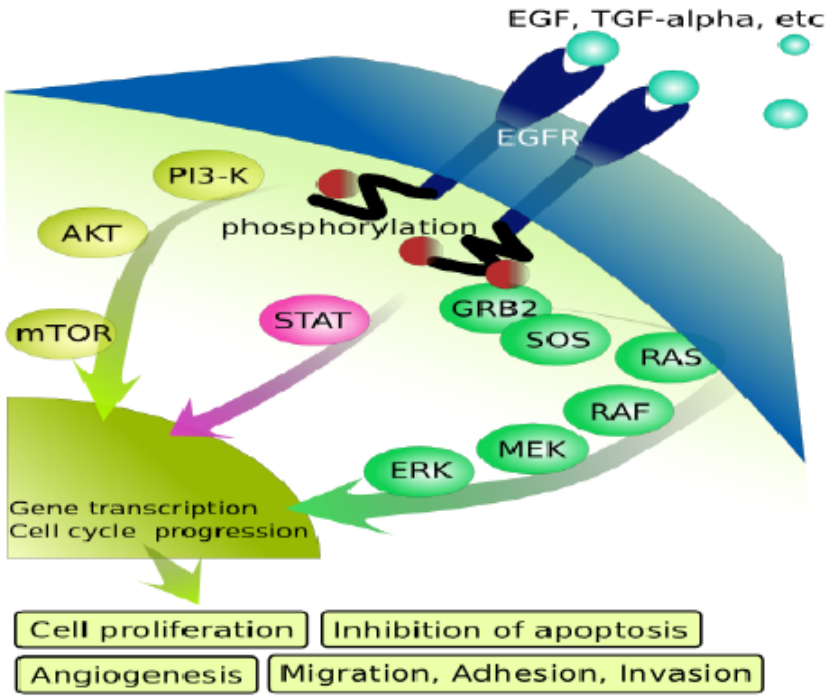
L'interprétation abstraite permet (formellement) de :

- Simplifier, réduire un modèle.
- Réduire la complexité de certains calculs.
- Conserver un lien avec le modèle original (interprétation des preuves, etc).

Exemples de travaux récents (et en cours)

- Réduction automatique de modèles à base de règles (Feret et al) :
  - Modélisation des systèmes en Kappa ;
  - Sémantiques continues (ODE) et stochastiques.

Signalling Pathways



tion abstraite

en bio-informatique

t) de :

interprétation des

Eikuch, 2007 : de règles (Feret et al) :

- Modélisation des systèmes en Kappa ;
- Sémantiques continues (ODE) et stochastiques.

Interprétation abstraite

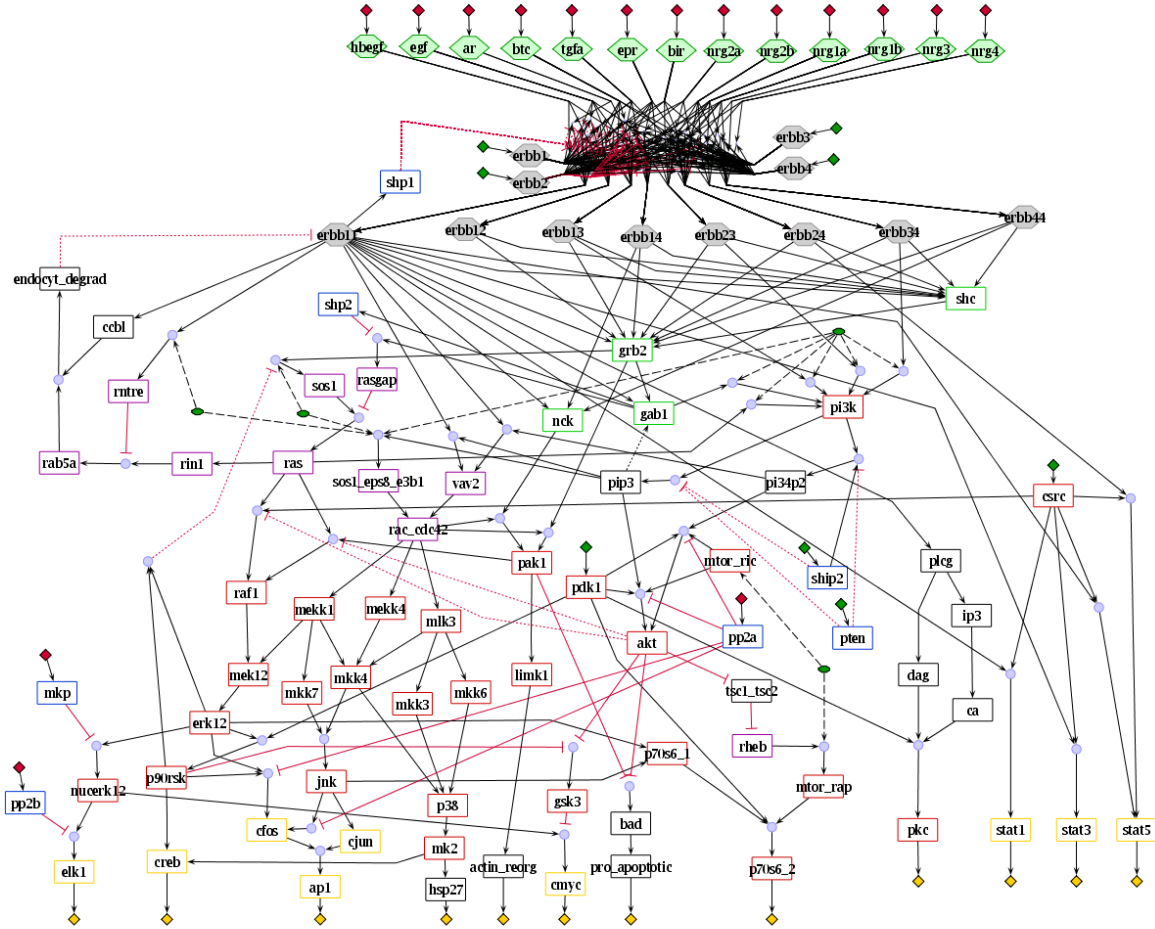
Applications en bio-informatique

L'interprétation abstraite permet (formellement) de :

- Simplifier, réduire un modèle.
- Réduire la complexité de certains calculs.
- Conserver un lien avec le modèle original (interprétation des preuves, etc).

Exemples de travaux récents (et en cours)

- Réduction automatique de modèles à base de règles (Feret et al) :
  - Modélisation des systèmes en Kappa ;
  - Sémantiques continues (ODE) et stochastiques.
- Vérification formelle de grands réseaux de régulation (Paulevé et al) :
  - Modélisation des systèmes en Process Hitting.
  - Extraction de composants clés.



tion abstraite

en bio-informatique

t) de :

interprétation des

de règles (Feret et al) :

- Sémantiques continues (ODE) et stochastiques.

- Vérification formelle de grands réseaux de régulation (Paulevé et al) :
  - Modélisation des systèmes en Process Hitting.
  - Extraction de composants clés.

Conclusion

Étude des systèmes dynamiques :

- Comportement global émergent complexe.
- Systèmes de grande taille.
- Nombre gigantesque de comportements possibles.

But : trouver les explications les plus simples.

L'informatique apporte des méthodes automatiques réutilisables.

Abstraction des modèles

- Réduction des modèles.
- Simplification des modèles.
- Connexions théoriques entre les différents modèles.