# Calcul par membranes en MGS

## Analyse du protocole Needham-Schroeder

Antoine Spicher

Séminaire MeFoSyLoMa

d'après les travaux d'Olivier Michel et Florent Jacquemard

# Plan

- Présentation du langage MGS
  - ✓ Motivations
  - ✓ Collections topologiques
  - ✓ Transformations
  - ✓ Illustrations
- Analyse du protocole NSPK
  - ✓ Le protocole NSPK et son attaque
  - ✓ Recherche de la faille
  - ✓ Implantation en MGS
- Conclusion

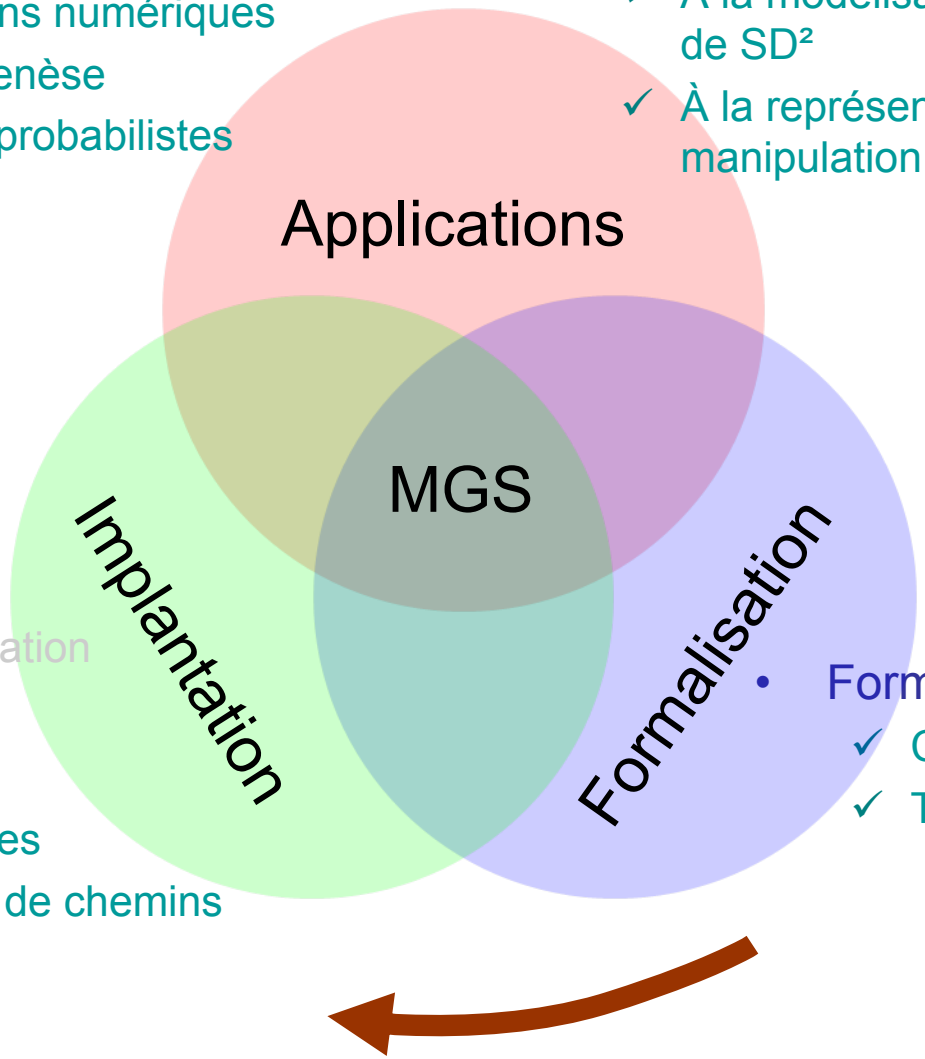
# Plan

- Présentation du langage MGS
  - ✓ Motivations
  - ✓ Collections topologiques
  - ✓ Transformations
  - ✓ Illustrations
- Analyse du protocole NSPK
  - ✓ Le protocole NSPK et son attaque
  - ✓ Recherche de la faille
  - ✓ Implantation en MGS
- Conclusion

# Projet MGS

- Exemples
  - ✓ Modifications topologiques
  - ✓ Simulations numériques
  - ✓ Morphogenèse
  - ✓ Modèles probabilistes

- Motivations : langage dédié
  - ✓ À la modélisation et la simulation de SD²
  - ✓ À la représentation et la manipulation de l'espace




- Compilateur
  - ✓ Typage
  - ✓ SK-combinatorisation
- Interprète
  - ✓ G-cartes
  - ✓ Chaînes abstraites
  - ✓ Transformations de chemins
  - ✓ Patches
  - ✓ …

- Formalisation
  - ✓ Collections topologiques
  - ✓ Transformations

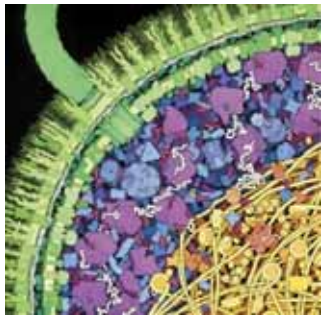
# Motivations

- Système dynamique à structure dynamique
  - ✓ Système complexe
  - ✓ Dont la structure évolue au cours du temps
    - La structure contraint le devenir du système qui contraint la structure …
    - L'espace ne peut être défini ***a priori***
    - La fonction d'évolution ne peut être spécifiée **globalement**

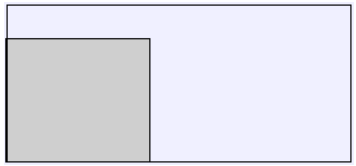
- Exemples
  - ✓ Biologie
    - Biologie moléculaire
    - Biologie du développement
  - ✓ Physique
    - Mécanique des corps mous, systèmes multi-échelles
    - Relativité générale
  - ✓ Urbanisme

    Croissance des villes, contrôle du trafic routier, …
  - ✓ Informatique

    Internet, réseau de senseurs, robotique modulaire, réseau de connaissances, …



© David S. Goodsell 1999



© Tecplot

© L. Sanders - EUROSIM

© C. Harrison - Clusterball project

# « Réécriture » et simulation

## Systèmes dynamiques

### Modélisation

#### État (espace)

Organisations hiérarchiques

#### Évolution

Interaction locale

Trajectoires

### Simulation

#### Gestion du temps

Discrète, orientée évènements : synchrone/asynchrone/…

## Techniques de réécriture

### Définition

#### Structure de données

Arbres formels (termes)

#### Système de réécriture

α => β, α : motif, β : expression

Dérivations

### Application

#### Stratégies d'application des règles

Maximale parallèle/séquentielle/ stochastique/…

# « Réécriture » et simulation

## Systèmes dynamiques

### Modélisation

#### État (espace)

Organisations hiérarchiques

**Organisations arbitraires**

#### Évolution

Interaction locale

Trajectoires

### Simulation

#### Gestion du temps

Discrète, orientée évènements : synchrone/asynchrone/…

## Techniques de réécriture

### Définition

#### Structure de données

Arbres formels (termes)

**Quelles structures de données ?**

#### Système de réécriture

α => β, α : motif, β : expression

Dérivations

### Application

#### Stratégies d'application des règles

Maximale parallèle/séquentielle/ stochastique/…

# « Réécriture » et simulation

## Systèmes dynamiques

### Modélisation

#### État (espace)

Organisations hiérarchiques

**Organisations arbitraires**

#### Évolution

Interaction locale

Trajectoires

### Simulation

#### Gestion du temps

Discrète, orientée évènements : synchrone/asynchrone/…

## Techniques de réécriture

### Définition

#### Structure de données

Arbres formels (termes)

**Collections topologiques**

#### **Transformation**

α => β, α : motif, β : expression

Dérivations

### Application

#### Stratégies d'application des règles

Maximale parallèle/séquentielle/ stochastique/…

# Collection topologique

## Les collections topologiques

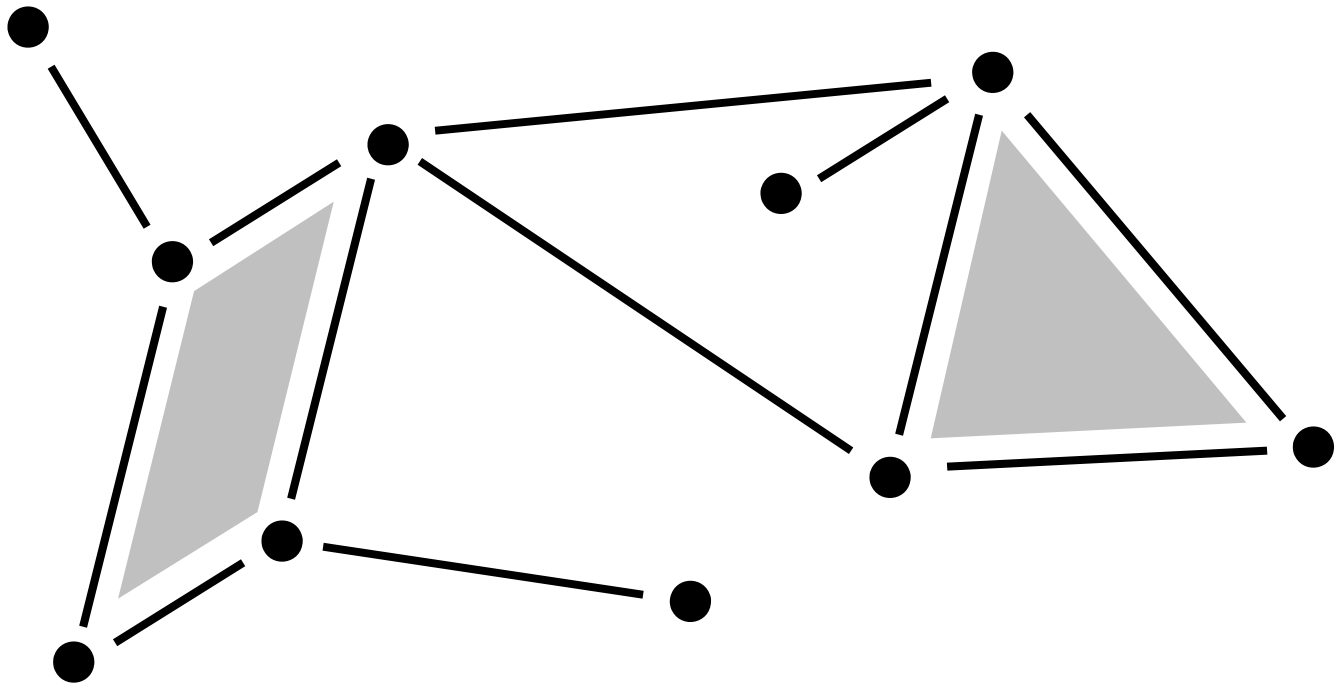
- ✓ **Structure :**
  - Une collection de ***cellules topologiques***

0-cellule 1-cellule 2-cellule 3-cellule

# Collection topologique

## Les collections topologiques

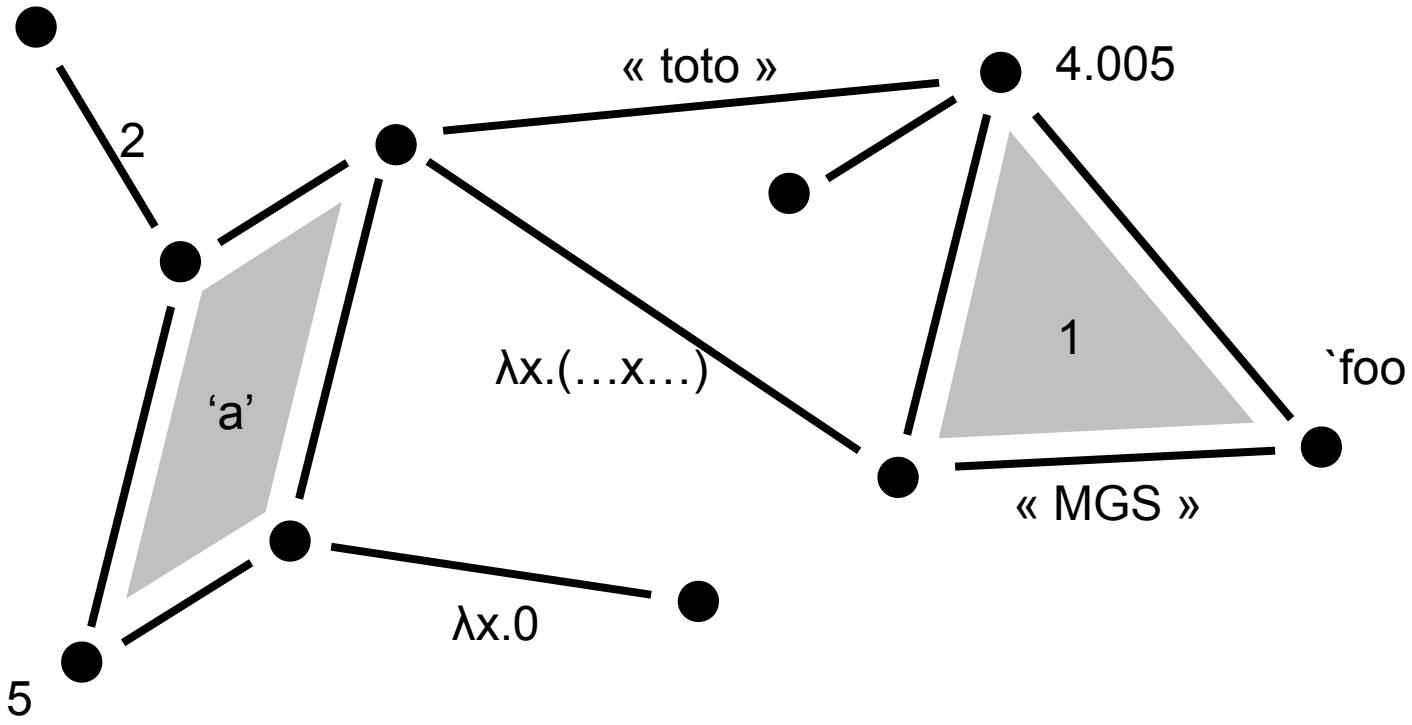
- ✓ **Structure :**
  - Une collection de cellules topologiques
  - Une ***relation d'incidence***



# Collection topologique

## Les collections topologiques

- ✓ **Structure** :
  - Une collection de cellules topologiques
  - Une relation d'incidence
- ✓ **Données** : ***associer une valeur à chaque cellule***

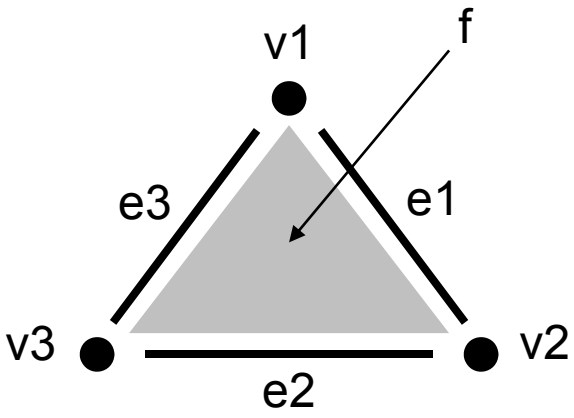


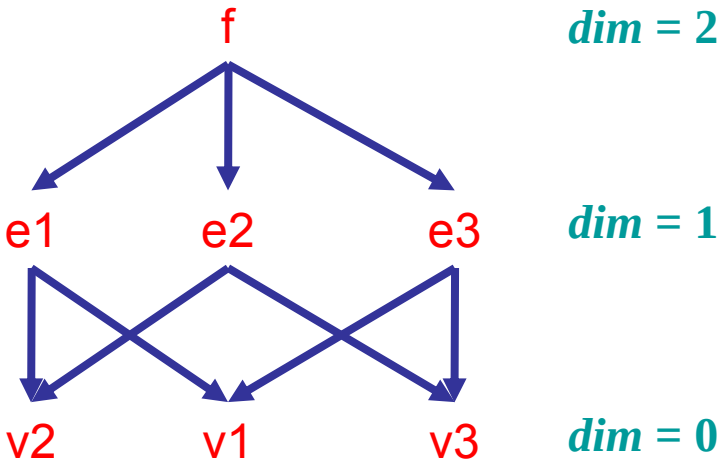
# Collection topologique

- Structure : complexe cellulaire abstrait
  - Définition

*Un* **complexe cellulaire abstrait** $\mathcal{K}$ *est un triplet* $(S, \prec, dim)$ *tel que*

- $S$ *est un ensemble de symboles, les* **cellules topologiques**
- $\prec \subset S \times S$ *est un ordre partiel sur* $S$*, la* **relation d'incidence**
- $dim : S \longrightarrow \mathbb{N}$*, la* **dimension**, *telle que* $\sigma_1 \prec \sigma_2 \Rightarrow dim(\sigma_1) < dim(\sigma_2)$



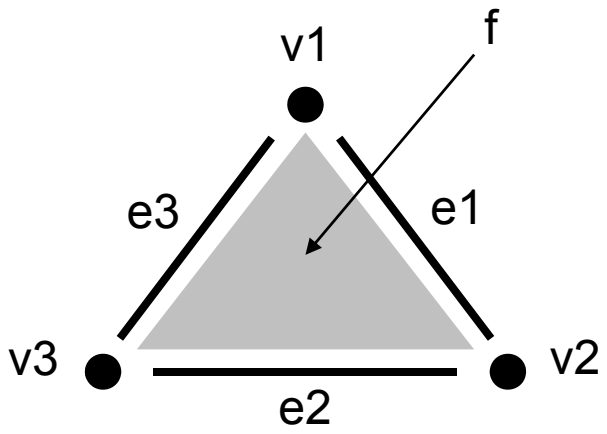
# Collection topologique

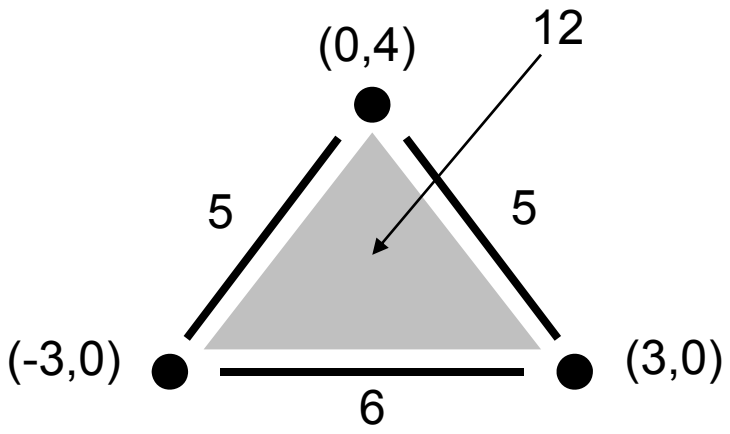
- Données : chaîne topologique
  - ✓ Définition

> *Une* **chaîne topologique** *c d'un complexe $\mathcal{K}$ à valeur dans un groupe abélien G est une* ***fonction totale nulle presque partout*** *de S dans G.*

  - ✓ Représentation par des sommes formelles finies

$$c \in C(\mathcal{K}, G) \quad \Rightarrow \quad c = \sum_{\sigma \in \mathcal{K}} c(\sigma).\sigma$$



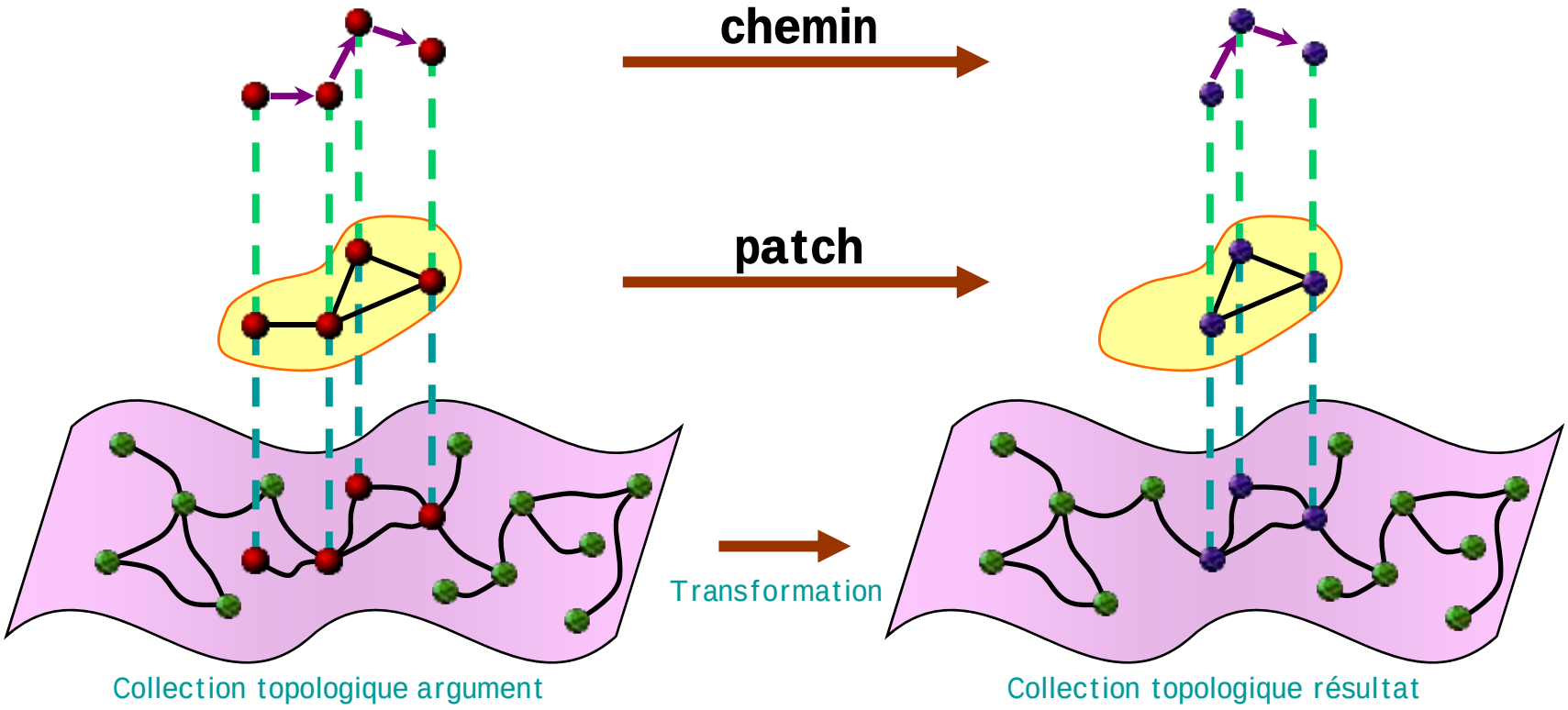

C = (0,4).v1 + (3,0).v2 + (-3,0).v3 + 5.e1 + 6.e2 + 5.e3 + 12.f

# Transformation

## Les transformations

- ✓ Réécriture locale de collection
- ✓ Fonction définie par cas
  - Chaque cas filtre une sous-collection
  - Un cas = un filtre

$$T = \begin{cases} rule_1 : pattern_1 \Rightarrow \text{expression}_1 \\ rule_2 : pattern_2 \Rightarrow \text{expression}_2 \\ \ldots \\ rule_n : pattern_n \Rightarrow \text{expression}_n \end{cases}$$



# Illustrations

- Réécriture de multi-ensemble (calcul chimique) : calcul des nombres premiers

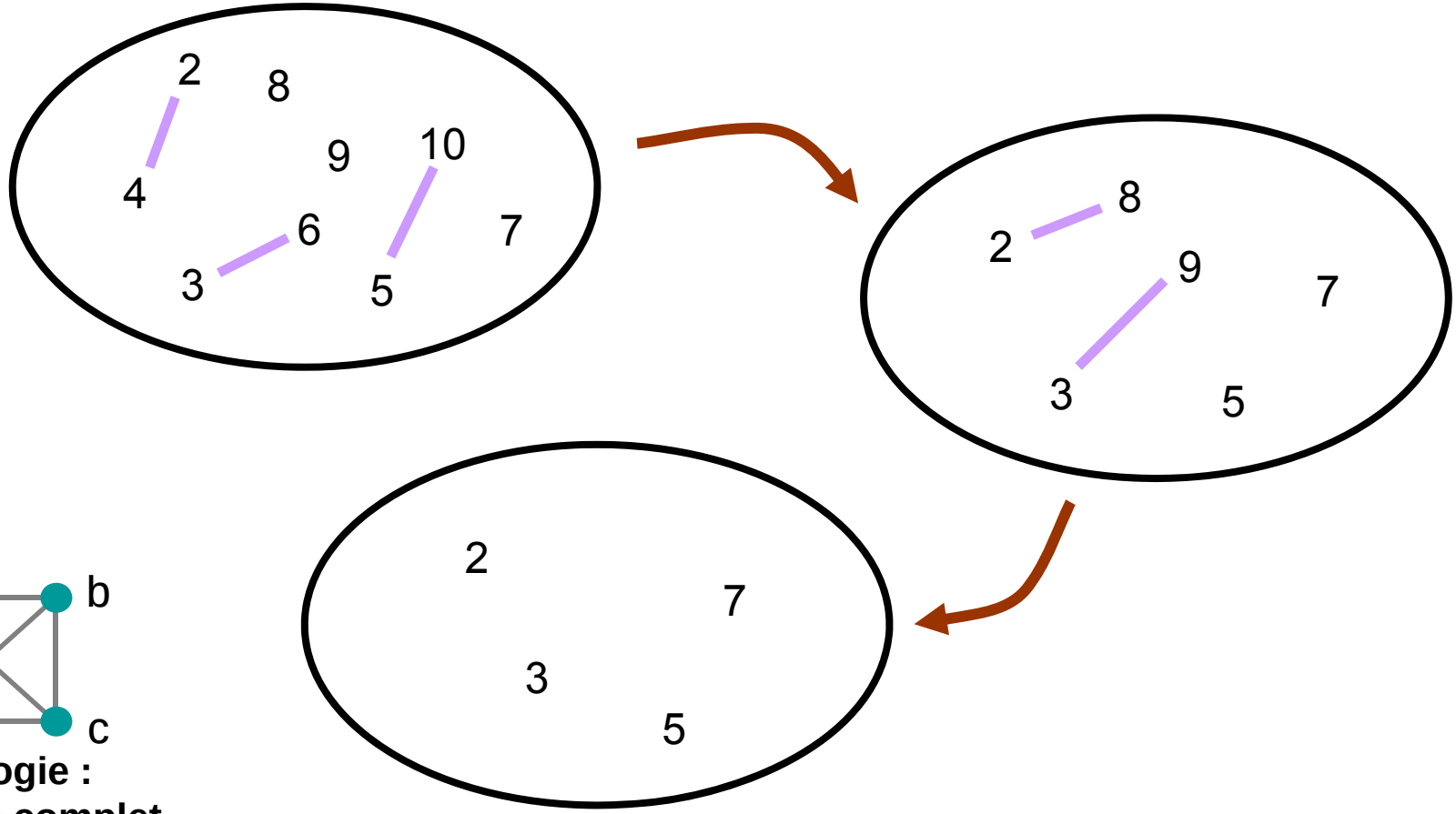
```
x, y / x div y => x
```

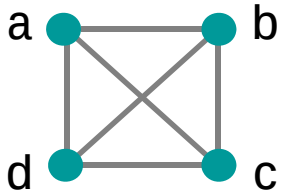
**Trouver x voisin de y tel que x divise y**
**Remplacer (x, y) par x**

# Illustrations

- Réécriture de multi-ensemble (calcul chimique) : calcul des nombres premiers

```
trans { x, y / x div y => x }
```




**Topologie :**
**un graphe complet**

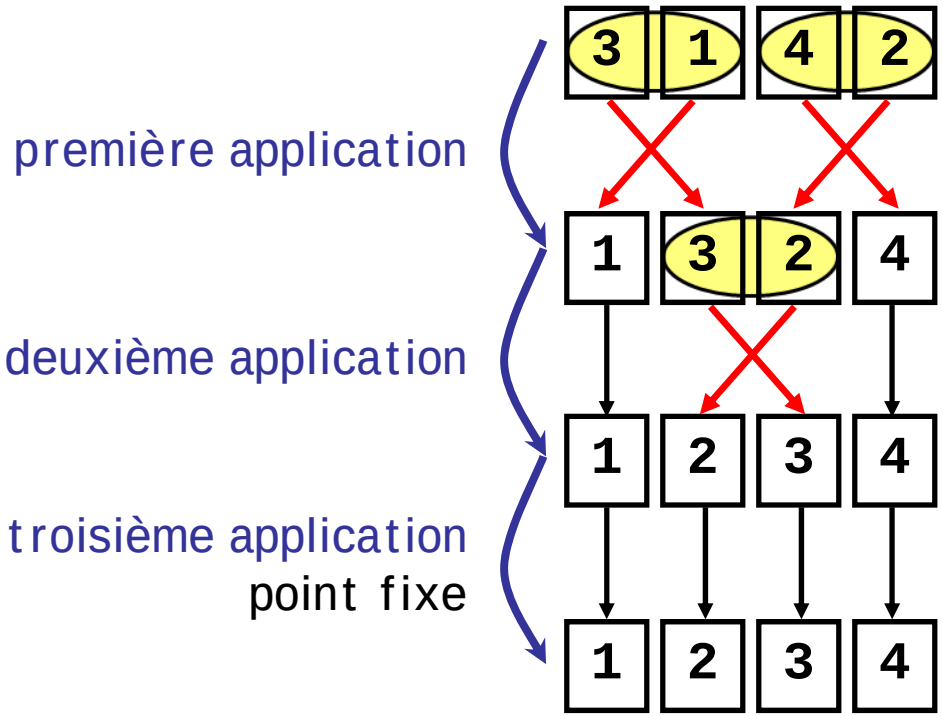
# Illustrations

- Réécriture de multi-ensemble (calcul chimique) : calcul des nombres premiers

```
trans { x, y / x div y => x }
```

- Réécriture de séquence (L système) : tri à bulles

```
trans { x, y / x > y => y, x }
```





**Topologie :**
**un graphe linéaire**

# Illustrations

- Réécriture de multi-ensemble (calcul chimique) : calcul des nombres premiers

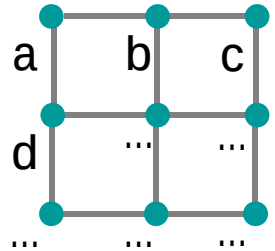
```
trans { x, y / x div y => x }
```

- Réécriture de séquence (L système) : tri à bulles

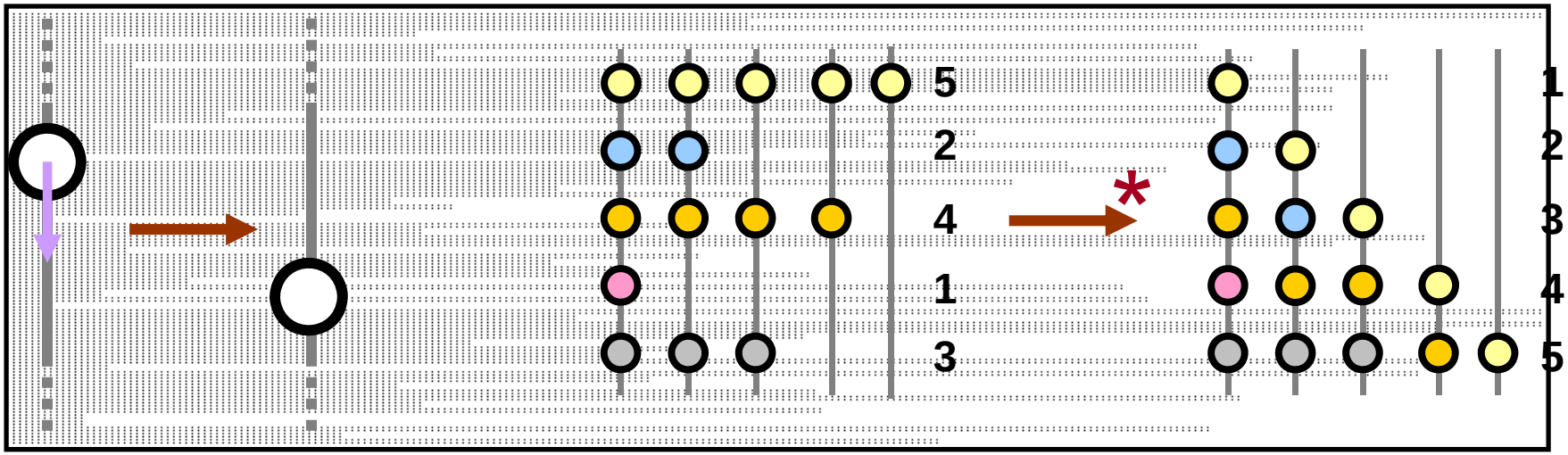
```
trans { x, y / x > y => y, x }
```

- Réécriture de tableau (automate cellulaire) : tri par boulier

```
trans { x |sud> <undef> => <undef>, x }
```




**Topologie :**
**un graphe régulier**



# Plan

- Présentation du langage MGS
  - ✓ Motivations
  - ✓ Collections topologiques
  - ✓ Transformations
  - ✓ Illustrations
- Analyse du protocole NSPK
  - ✓ Le protocole NSPK et son attaque
  - ✓ Recherche de la faille
  - ✓ Implantation en MGS
- Conclusion

# Description du protocole NSPK

- Objectif du protocole
  - ✓ *Authentification* mutuelle entre 2 utilisateurs *A(lice)* et *B(ob)*
  - ✓ Cryptographie fondée sur des *clefs publiques*

- Restriction
  - ✓ Version originale : serveur de clefs publique (PKI) + 3 messages
  - ✓ Supposition : A et B connaissent chacun la clef publique de l'autre

- Vocabulaire
  - ✓ Agents :
    constantes *identifiants* les utilisateurs : A et B sont des agents
  - ✓ Nonce :
    entier aléatoire généré par chaque agent pour ***une*** exécution du protocole : $N_a$
  - ✓ Message :
    donnée structurée contenant de l'information (possiblement chiffrée) : $\{ A, N_a \}$
  - ✓ Clef publique :
    valeur utilisée pour chiffrer les messages : $\{ A, N_a \}_{K(B)}$

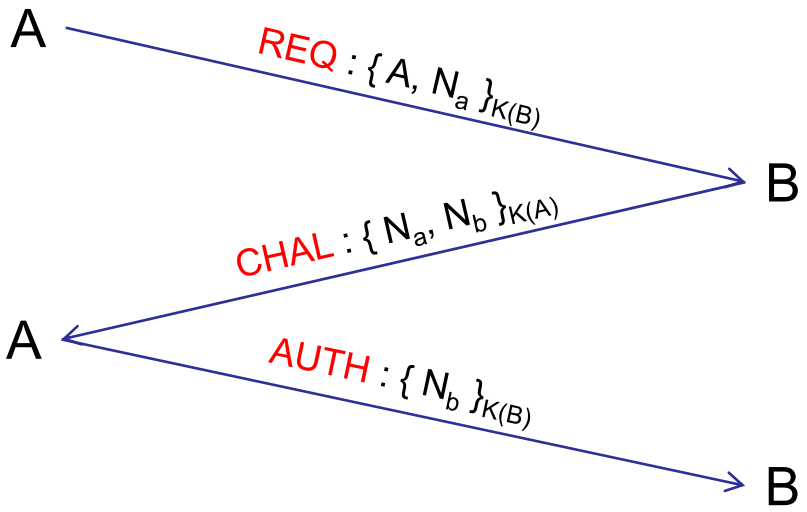
# Description du protocole NSPK

- Exécution du protocole : communication *asynchrone*
  - ✓ REQ

    *Alice* génère un nonce $N_a$, le concatène à son identifiant A, chiffre le tout avec la clef de *Bob* K(B) et envoie le message sur le réseau
  - ✓ CHAL

    *Bob* reçoit le message, le déchiffre, conserve le nonce $N_a$ d'*Alice*. Il génère un nonce $N_b$, le concatène à celui d'*Alice*, chiffe le tout avec la clef d'*Alice* K(A) et envoie le message
  - ✓ AUTH

    Alice reçoit le message, le déchiffre, conserve le nonce de $N_b$ Bob. Elle chiffre le nonce de Bob avec la clef K(B) et envoie le message

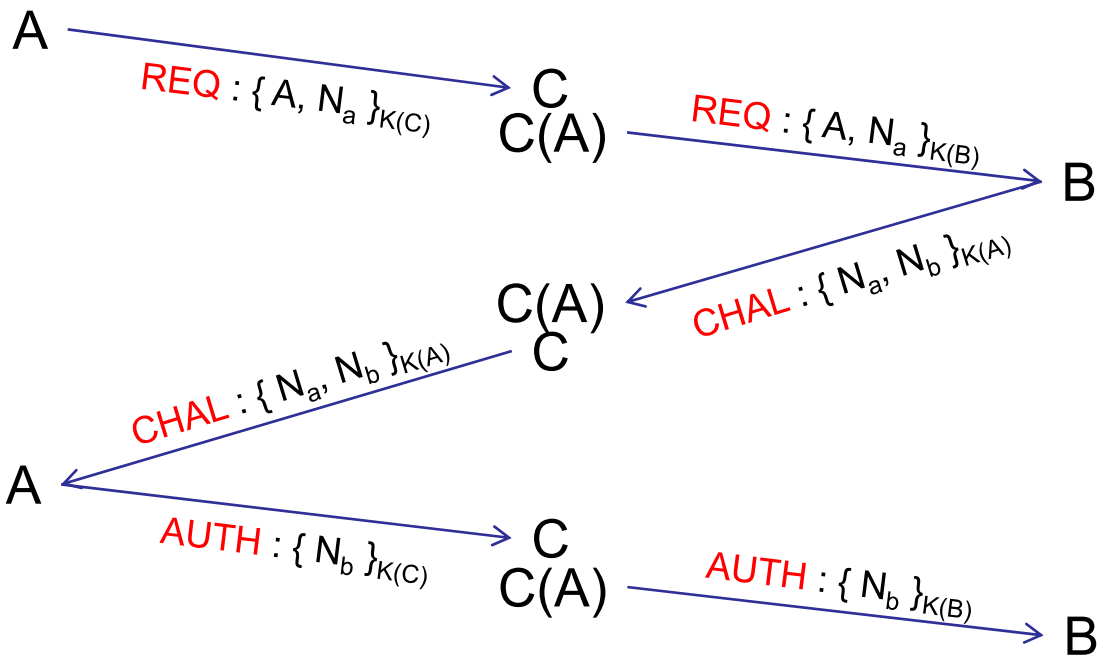


# Description du protocole NSPK

- Faille de sécurité
  - ✓ Un nouvel agent intrus : *C(harly)*
  - ✓ Charly se fait passer pour Bob auprès d'Alice

    Alice *récupère la clef publique* de Charly croyant que c'est celle de Bob
  - ✓ Charly peut se faire passer pour Alice auprès de Bob

    Charly *récupère le nonce* de Bob

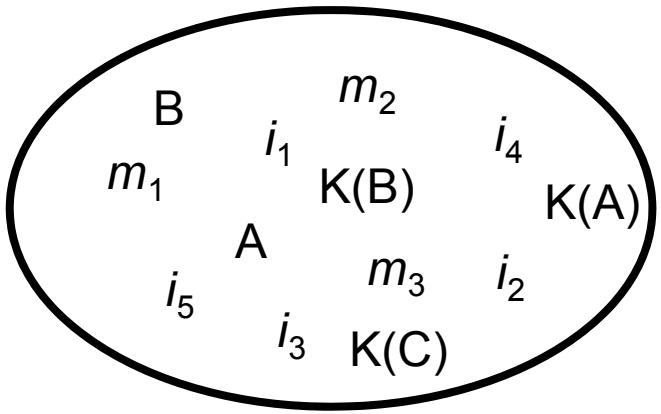


# Recherche de l'attaque

- Méthode
  - ✓ Force brute et exploration de l'espace d'états
  - ✓ Complétude
    - Indécidable dans le cas général (espace d'états généralement infini)
    - Cas d'un espace d'états *fini*

      *décidable* avec un nombre *fini* d'agents [Rusinowitch01]
  - ✓ Utilisation du calcul par membranes

    Réécriture de (multi-)ensembles imbriqués



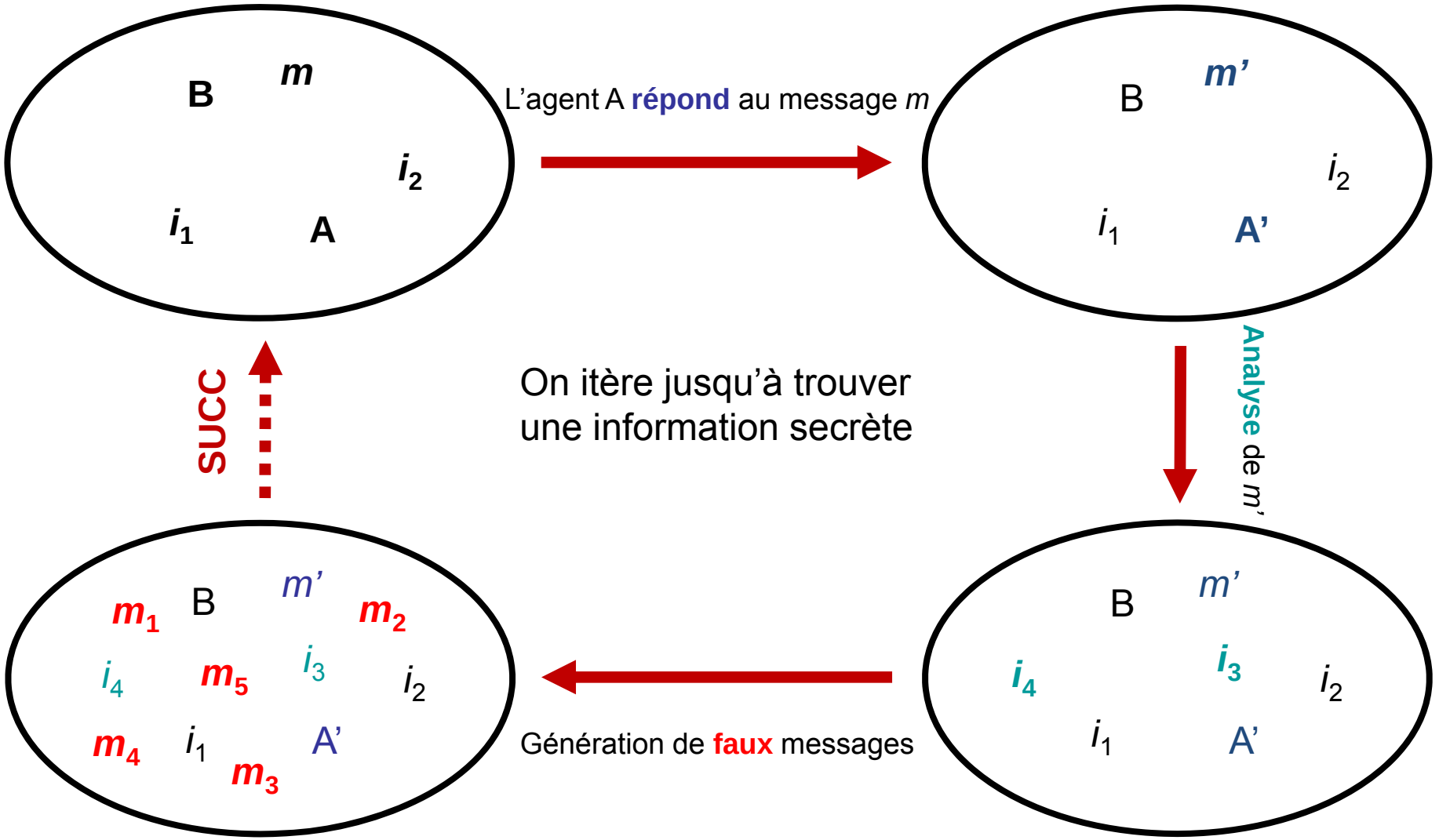

État = solution chimique

- ✓ Utilisateurs

  Alice et Bob
- ✓ Clefs publiques
- ✓ Messages
- ✓ Informations (centré intrus)
  - Clefs privées
  - Nonces
  - …

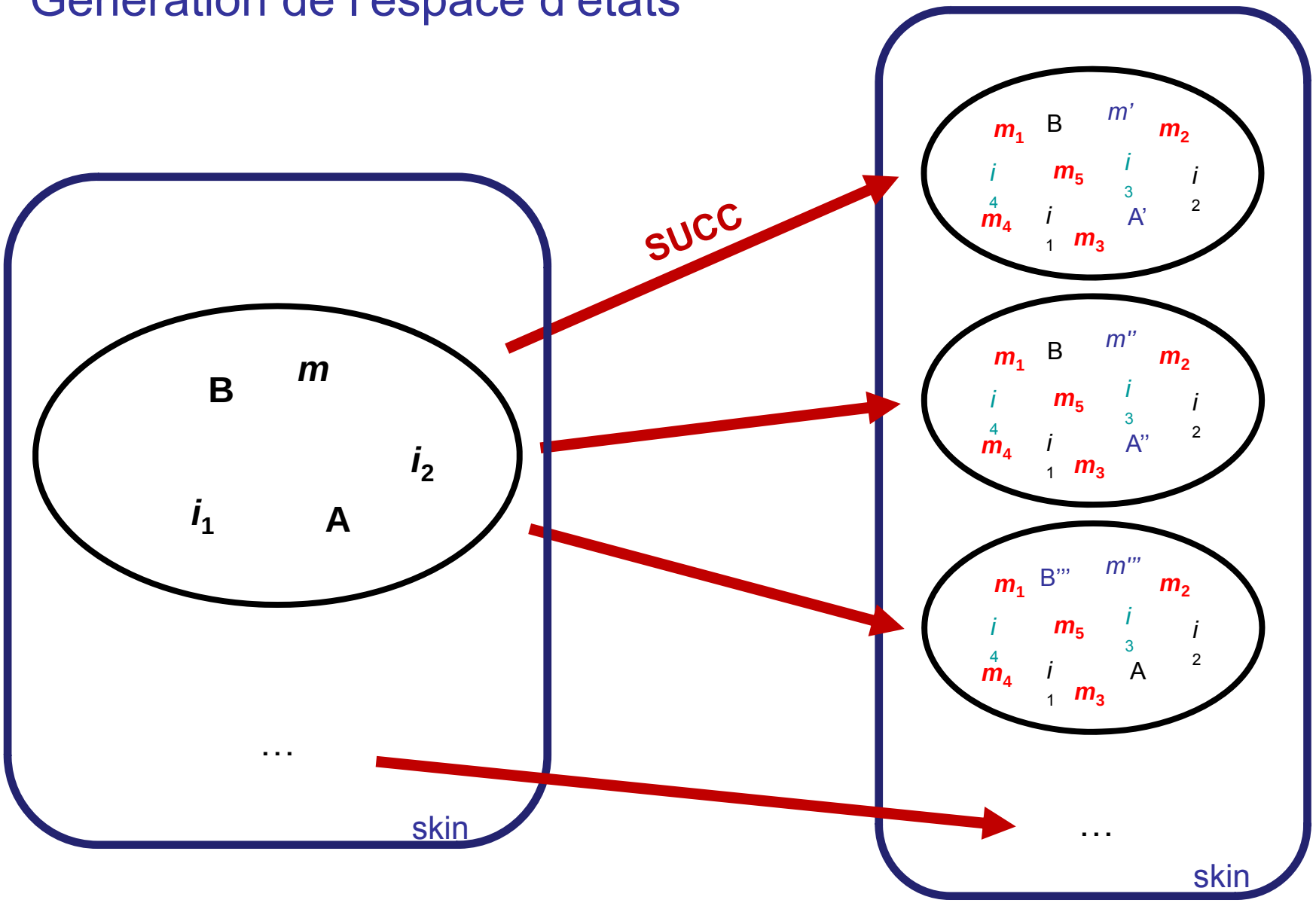
# Recherche de l'attaque

- Générations de l'espace d'états



# Recherche de l'attaque

- Génération de l'espace d'états



# Implantation en MGS

## Représentation des composants élémentaires

- ✓ Les agents : A, B, …

```
     record agent = { id:string, ni:int, nr:int, pc }
and  record alice = agent + { pc:instA, dest:string }
and  record bob   = agent + { pc:instB }
and  type   instA = `Req  | `Auth | `Finished
and  type   instA = `Chal | `Wait | `Finished ;;
```

- ✓ Les clefs publiques : K(A), K(B), …

```
     record pubKey = { pub:string } ;;
```

- ✓ Les messages : $\{ N_a, N_b \}_{K(C)}$, …

```
     record messageReq  = {  a:string, na:int, kb:pubKey }
and  record messageChal = { na:int,    nb:int, ka:pubKey }
and  record messageAuth = { nb:int,            kb:pubKey } ;;
```

- ✓ Les informations : $i_1$, $i_2$, $i_3$, …

```
     record info_name  = { name:string }
and  record info_nonce = { nonce:int   }
and  record info_priv  = { priv:string } and ... ;;
```

# Implantation en MGS

## Réaction des agents aux messages

- ✓ Une transformation par type d'agent et de message

```
trans alice_req = {        // type : state -> state set
  a:alice / (x.pc == `Req), k:pubKey / (k.pub == a.dest) =>
    return ( (a + { pc = `Auth }, k,                          // m-à-j d'Alice
              { a = a.id, na = a.ni, kb = k },                // envoi du message
              neighbors(a)) :: set:() )                       // le reste
} ;;
trans bob_chal = {         // type : state -> state set
  b:bob / (x.pc == `Chal) =>
    let known_keys   = filter(pubKey, neighbors(b)) in
    let all_messages = filter(messageReqCond(b,known_keys), neighbors(b))
    in
      return (map ((fun m -> b+{pc=`Wait, nr=m.na},           // m-à-j de Bob
                            {ka=m.a, na=m.na, nb=y.ni},       // envoi du message
                            neighbors(b)),                    // le reste
                   all_messages) )
} ;;
trans alice_auth = { … } ;;
trans bob_finish = { … } ;;
```

- ✓ Combinaisons des réactions

```
fun reactions(ens) =       // type state set -> state set
  flatten(trans (s => alice_req(s), bob_chal(s), …)(ens)) ;;
```

# Implantation en MGS

## Attaque de l'intrus

### ✓ Lecture des informations

```
trans intruder = {          // type : state -> state
    m:messageReq,  k:info_priv / decypher(k,m)
                               =>   m, { nonce = m.na }, { name = m.a }, k ;
    m:messageChal, k:info_priv / decypher(k,m)
                               =>   m, { nonce = m.na }, { nonce = m.nb }, k ;
    m:messageAuth, k:info_priv / decypher(k,m)
                               =>   m, { nonce = m.nb }, k
} ;;
```

### ✓ Génération des faux messages

```
trans forge'[acc] = {     // type : state -> state
   (k:info_pub), (n:info_nom), (m:info_nonce)    /
      (acc := { na = m.nonce,  a = n.nom,   kb = k.pub }, acc ; false) => !!(0);
   (k:info_pub), (n:info_nonce), (m:info_nonce)  /
      (acc := { na = m.nonce, nb = n.nonce, ka = k.pub },
              { nb = m.nonce, na = n.nonce, ka = k.pub }, acc ; false) => !!(0);
   (k:info_pub), (m:info_nonce)                  /
      (acc := { nb = m.nonce, kb = k.pub              }, acc ; false) => !!(0);
   _ => return(acc)
} ;;
fun forge(state) = forge'[acc=state](state) ;;  // type : state -> state
```

### ✓ Attaque

```
fun attack'(state) = forge(intruder(state)) ;;  // type : state -> state
fun attack(ens)    = map (attack'[*]) ens   ;;  // type : state set -> state set
```

# Implantation en MGS

- Trouver l'attaque

```
fun found(state) = member({ nonce = 1}, state) ;;
fun find(ens) = (
  let succ = attack(reactions(ens)) in
    if filter(found,succ) != set:()
    then raise Found succ
    else find(succ)
) ;;
```

- État initial

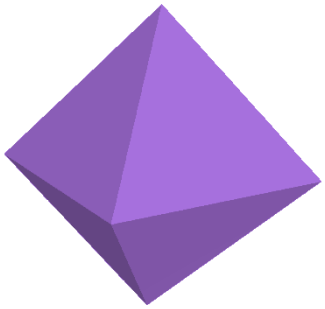
```
initial := (
  { id = "alice", dest = "charly", ni = 0, nr, pc = 'REQ },
  { id = "bob", ni = 1, nr, pc = 'CHAL },
  { priv = "charly" },
  { pub  = "charly" }, { pub = "alice" }, { pub = "bob" },
  set:())::set:() ;;
```

- Résultats
  - ✓ 150 lignes de code
  - ✓ Moins d'1s de calcul sur un PC 1,4GHz/Linux
  - ✓ Trouve la faille de sécurité connue

# Plan

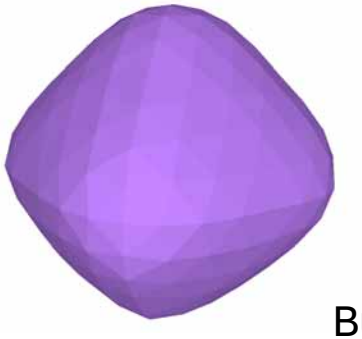
- Présentation du langage MGS
  - ✓ Motivations
  - ✓ Collections topologiques
  - ✓ Transformations
  - ✓ Illustrations
- Analyse du protocole NSPK
  - ✓ Le protocole NSPK et son attaque
  - ✓ Recherche de la faille
  - ✓ Implantation en MGS
- Conclusion

# Applications en modélisation géométrique

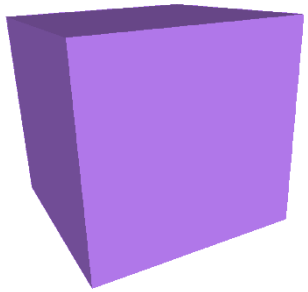


Maillage triangulaire

Loop



Butterfly



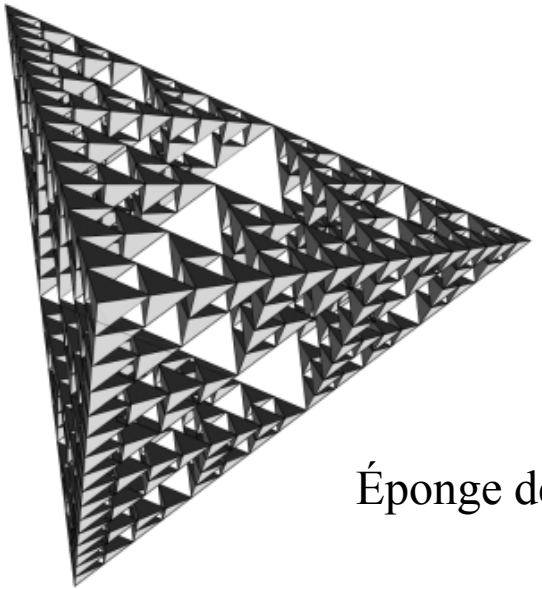
Maillage quadrangulaire

Catmull-Clark

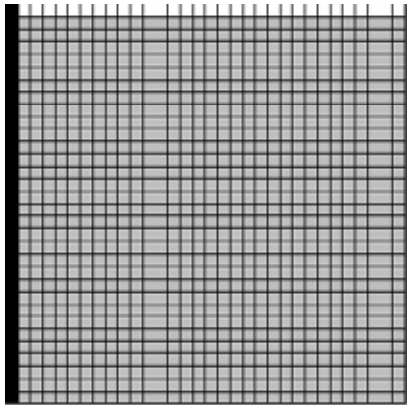
Kobbelt

Doo-Sabin,
chanfreinage

# Applications en auto-assemblage

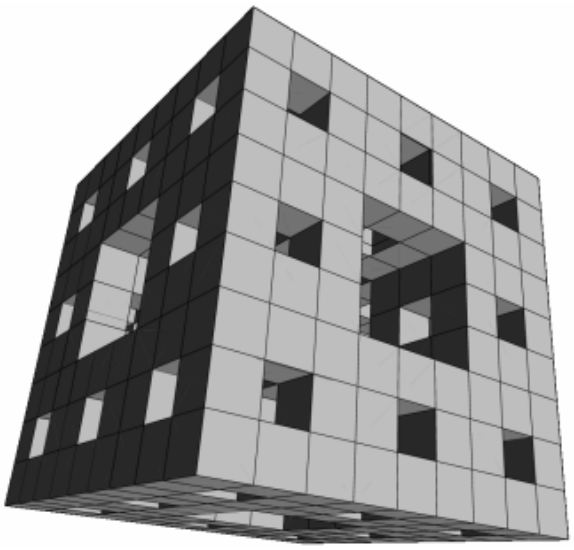


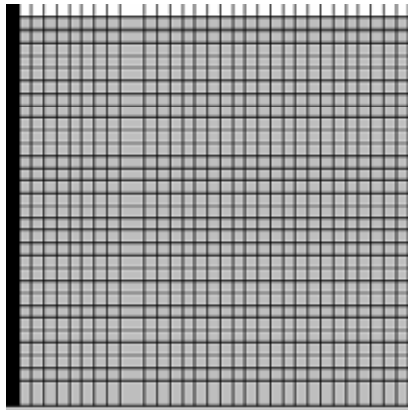
Éponge de Sierpinski



Stratégie maximale parallèle

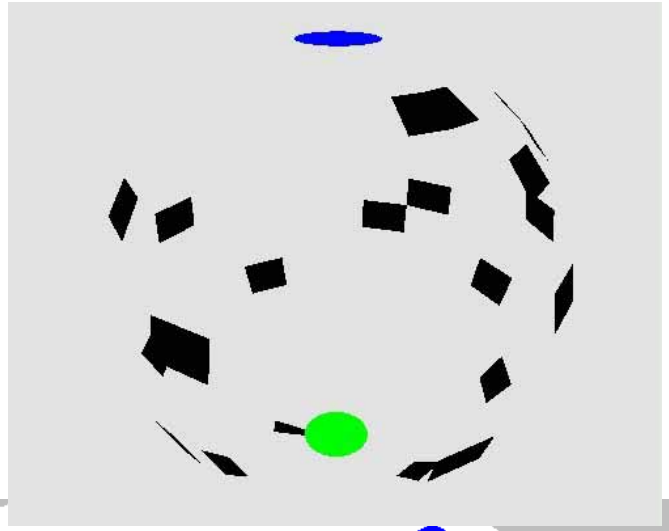
Éponge de Menger

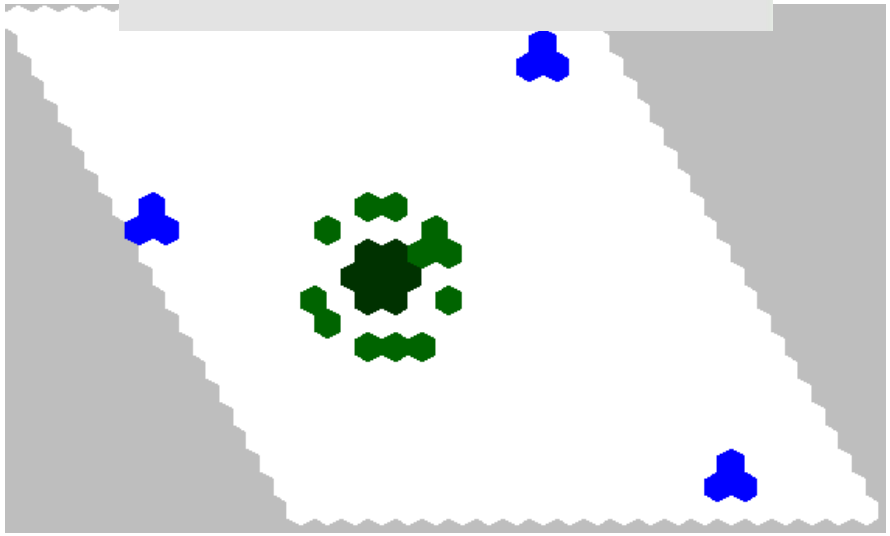




Stratégie stochastique

# Programmer une population





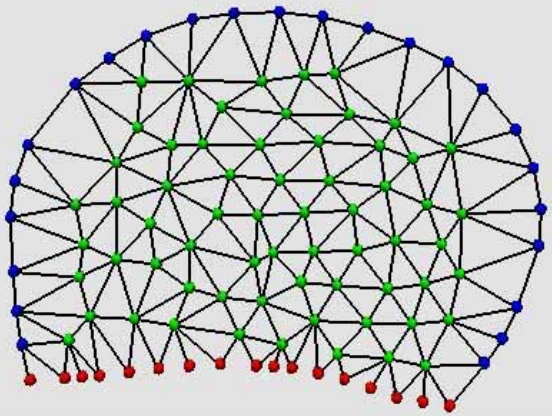
Ants foraging

Une transformation pour tout type de topologies : *polytypisme*

Rassemblement d'oiseaux

- Aucun leader
- 3 règles d'évolution
- Un comportement global cohérent

# Applications à la modélisation de SD²

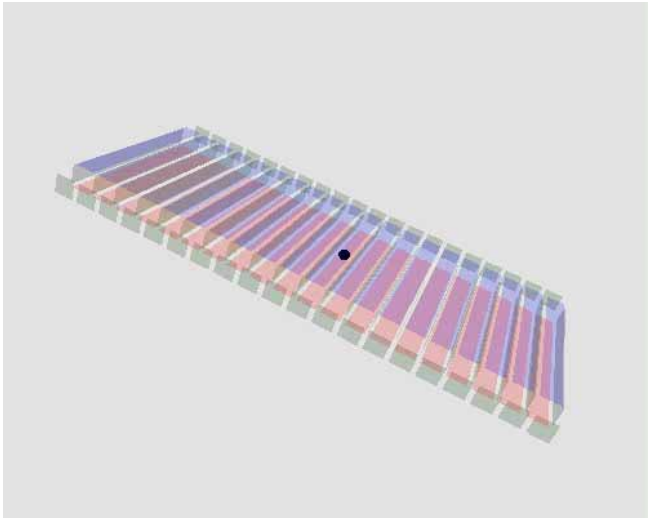


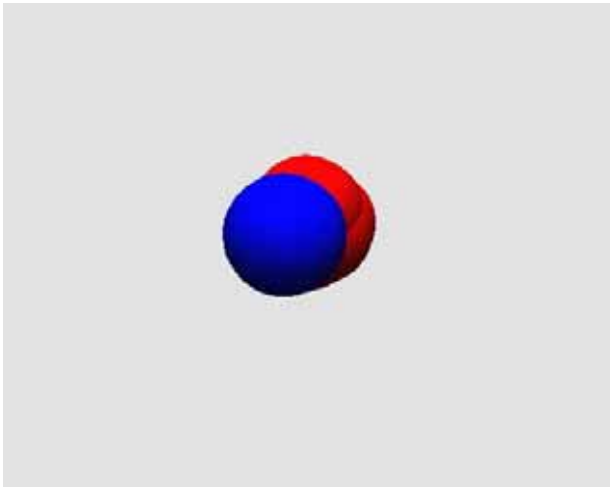
Déplacement cellulaire

Maillage adaptatif

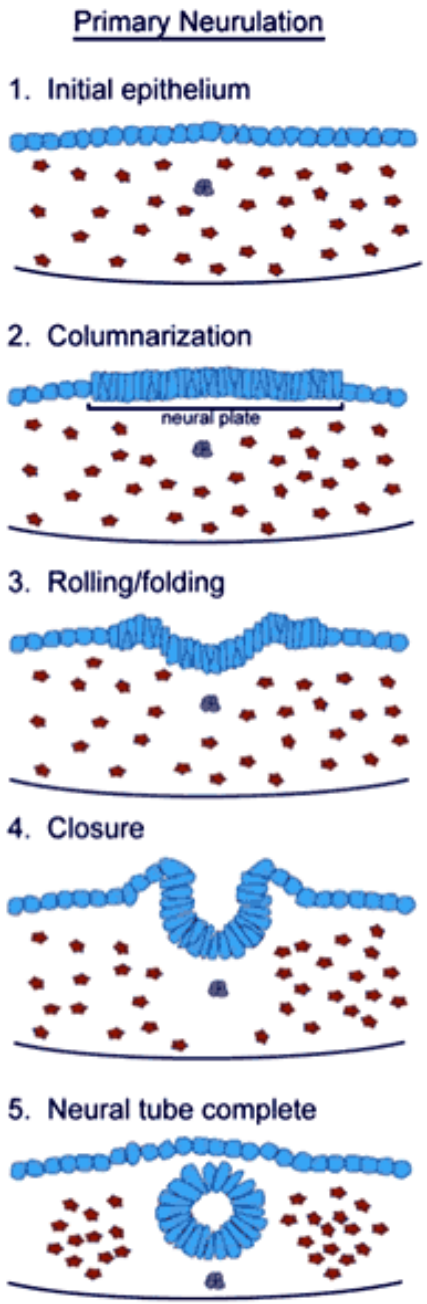
Neurulation

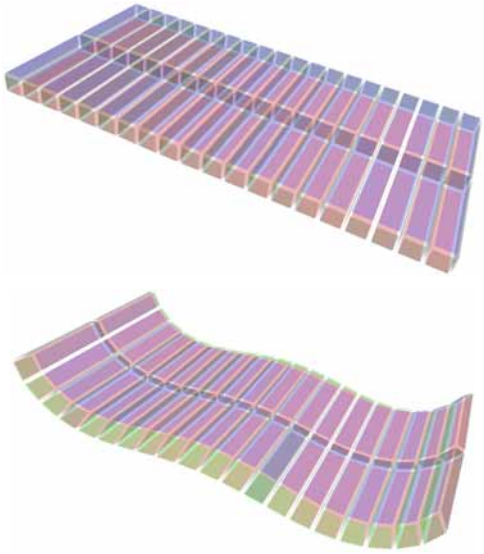
Changement de topologie





Croissance d'une tumeur



# Questions ?

Visitez notre page web ☺

`http://mgs.ibisc.uni-evry.fr`

http://pharyngula.org/index/weblog/comments/neurulation_in_zebrafish/