# Introduction :

Aie aie aie.

# Définitions :

Dans cette section, nous nous intéressons donc à la définition d’un écosystème, ainsi que de ces composantes.

## Ecosystème

Un écosystème est donc un ensemble d’agents A qui peuvent être présents ou non.

Nous partons du principe qu’aucun agent ne peut être présent et absent à la fois.

La présence d’un agent, appelée polarité, est dénoté par a+ si l’agent a est présent, et a- sinon, celle-ci dépend des règles de réécritures R.

Plus formellement, un écosystème Ɛ est un tuple (A, R) tel que :

* A est un ensemble d’agents avec polarité tel que :

Ap = {a+, a- | a

## Agents

## Règles de réécritures

## 

## Exemple 1 et 2

Nos environnements sont représentés par un ensemble d’agents et de règles, comme par exemple :

**Agents :**

B : Oiseaux

I : Insectes

Pe : Pesticides

P : pluie

**Règles :**

B+ => I- Les oiseaux mangent les insectes

Pe-, R+ => I+ La pluie et l’absence de pesticides laissent les insectes se développer

Mais comment représenter les similarités entre deux environnements sur un solveur tel que Cplex ?

Nous devons donc faire le lien entre les agents des deux environnements, puis entre les règles. Nous utilisons pour ça deux matrices booléennes, la première ; X représentant les correspondances entre agents, et la seconde Y représentant les correspondances entre règles.

Par exemple, si l’on avait deux environnements :

**Environnement 1 :**

Agents : A, B

Règles : A+ => B+ A+ => A-

**Environnement 2 :**

Agents : C, D

Règles : C- => D+ D- => D+

La matrice X sera sous la forme :

A+ A- B+ B-

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

C+

C-

D+

D-

Ici, si la case [A+][C+] vaut 1, c’est que notre programme a trouvé que la comparaison entre nos environnement la plus optimale nous donne une correspondance entre les agents A+ et C+.

La matrice Y sera :

A+ => B+ A+ => A-

|  |  |
| --- | --- |
|  |  |
|  |  |

C- => D+

D- => D+

Si la première case vaut 1, c’est qu’il y a correspondance entre les règles A+ => B+ et C- => D+.

Les contraintes à mettre en place pour rendre les correspondances injectives sont alors assez simples ;

-Dans les deux cas on ne doit avoir qu’un « 1 » par ligne et par colonne, un agent ou une règle

## Similarités entre 2 écosystèmes

# Mon travail :

Le but a donc été pour moi

## Contraintes non linéaires dans la fonction d’évaluation :

Une des difficultés rencontrées a été la //multiplication de variables nécessaire au calcul de notre fonction d’évaluation. En effet cplex ne calcule pas les contraintes non linéaires. Pour pallier à ce problème, nous avons utilisé une astuce permettant indirectement d’avoir une contrainte contenant une multiplication de deux cases de tableaux dvar : sachant que nos valeurs sont des booléens, si Z==X\*Y n’est pas une contrainte calculable par Cplex, Z<=X, Z<=Y, Z>=X+Y-1 nous renverra le même résultat pour Z et sera accepté par cplex. Ainsi, en développant notre calcul de base, nous obtenons un résultat identique et juste.

## Exemples travaillés sur cplex :

Pour comprendre comment automatiser la comparaison de deux environnements, nous avons commencé par traduire deux exemples en code cplex, le premier, assez simple :

Environnement 1 :

Agents : A, B

Règles : A+ ==> B+ A+ ==> A-

Environnement 2 :

Agents : C, D

Règles : C- ==> D+ D- ==> D+

Lors de la comparaison de ces deux environnements sur cplex, on obtient une correspondance entre des couples d’agents (A+, C-), (A-, C+), (B+, D+) et (B-, D-), et des règles (A+ ==> B+, C- ==> D+) et (A+ ==> A-, D- ==> D+).

Cette correspondance est bien celle qui nous donne le meilleur résultat pour notre fonction d’évaluation S, on se demande alors comment étendre notre modèle à des cas aux règles plus complexes, nous essayons donc un deuxième exemple :

Environnement 1 :

Agents : A, B, C

Règles : A+, B- ==> B+ C- ==> A+

Environnement 2 :

Agents : D, E

Règles : D- ==> E+ E+, E- ==> E+

La taille des contraintes nous pose alors le problème de stockage de nos calculs de contraintes non linéaire. La solution a alors été la création d’un tableau pour chaque case de notre matrice de contrainte, rendant ainsi notre exemple plus proche de ce que l’on voudra obtenir avec notre traducteur.

## Notre programme :

Dans un premier temps, notre programme devra donc récupérer nos environnements et les stocker en mémoire

L’une des premières difficultés rencontrées lors de l’écriture de notre programme a été le stockage de nos agents et règles. En effet, il nous fallait un moyen pour faire le lien entre les différents agents d’une même règle de transition pour le calcul de notre fonction d’évaluation. Nous avons donc créé des structures adaptées d’environnement, d’agents et de règles. Les règles contiennent l’adresse des agents

# Analyse des résultats :

# Conclusion :