**Représentation fonctionnelle de diagrammes de Venn de dimension arbitraire.**

Outils pour le calcul des prédicats

**HANNIZ Anis, BEN HAMMOU Eddine**

# Rapport du Projet : Représentation fonctionnelle de diagrammes de Venn de dimension arbitraire.

31 Décembre 2023

## Contenu

# 1 Introduction

Nous avons étudié les représentations graphiques des diagrammes de Venn à deux et trois dimensions, définis respectivement sur deux ou trois prédicats. Traditionnellement, ces prédicats sont représentés sous forme de cercles. Cependant, représenter ces diagrammes pour des dimensions plus élevées peut devenir complexe. Une alternative consiste à les interpréter comme des fonctions partielles reliant les zones du diagramme à un ensemble de contraintes de remplissage.

Ce rapport offre une analyse détaillée des fonctions spécifiques implémentées dans le code Ocaml, explorant les concepts fondamentaux de la logique propositionnelle et des syllogismes. Il répond également aux questions énoncées, offrant une compréhension claire et détaillée des mécanismes sous-jacents et des méthodologies appliquées dans le cadre de la manipulation des diagrammes de Venn.

La logique propositionnelle et les syllogismes font appel à des outils visuels tels que les diagrammes de Venn pour représenter et comprendre les relations entre ensembles et les implications logiques. Ce rapport s'attache à détailler les différentes étapes et procédures utilisées pour calculer les diagrammes de Venn associés à des formules syllogistiques, ainsi que pour déterminer la compatibilité entre ces diagrammes.

En explorant les questions posées, ce rapport aborde en détail les concepts d'analyse des quantificateurs, d'utilisation des tables de vérité pour évaluer les formules logiques, et de construction des diagrammes de Venn correspondants. De plus, il détaille les étapes pour calculer la conjonction de deux diagrammes de Venn et explique comment déterminer la compatibilité de ces diagrammes en utilisant OCaml.

En somme, ce rapport constitue une ressource complète et détaillée, offrant une compréhension approfondie des mécanismes fondamentaux des diagrammes de Venn dans le contexte de la logique propositionnelle et des syllogismes.

# 2 Définitions

## 2.1 Diagramme de Venn

Un diagramme de Venn est une représentation graphique utilisée en théorie des ensembles et en logique pour illustrer les relations entre ensembles. Il est composé de formes géométriques, généralement des cercles, superposés ou juxtaposés, chacun représentant un ensemble. Les intersections et les unions des cercles permettent de visualiser les relations d'inclusion, d'intersection et d'union entre ces ensembles.

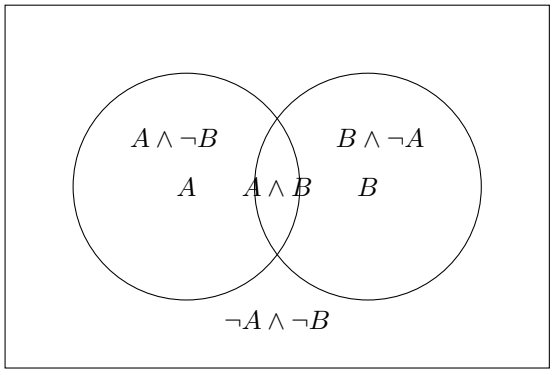
## 2.2 Opérations sur des diagrammes de Venn

Les opérations sur des diagrammes de Venn incluent des actions telles que la conjonction (intersection), la disjonction (union), la négation, la différence, et d'autres opérations logiques entre les ensembles représentés par les diagrammes de Venn. Ces opérations permettent de représenter graphiquement les résultats de différentes relations entre ensembles.

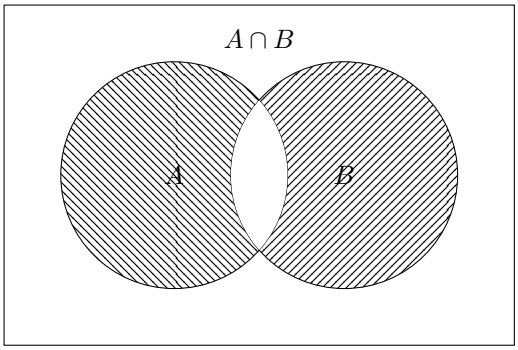
## 2.3 Logique propositionnelle

La logique propositionnelle est une branche de la logique formelle qui traite des propositions et de leurs relations en utilisant des connecteurs logiques tels que "et" ($\wedge$), "ou" ($\vee$), "non" ($\neg$), "implication" ($\rightarrow$), etc. Elle étudie la structure des propositions indépendamment de leur signification spécifique, se concentrant sur la validité des arguments.
Cas de l'Implication L'implication logique est une relation entre deux propositions, où une est la prémisse et l'autre est la conclusion. Si la prémisse est vraie, la conclusion doit également l'être.

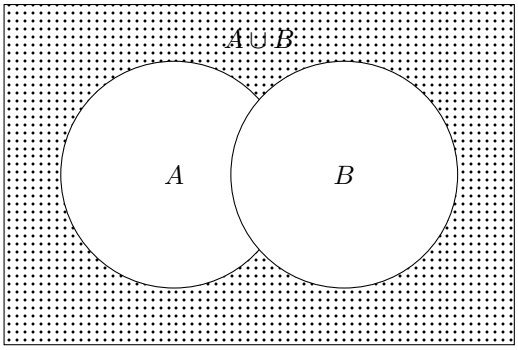
Exemple :

1. 

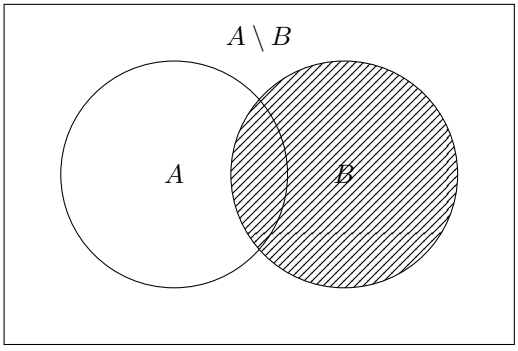
2. **Intersection ($\mathbf{A \cap B}$)**



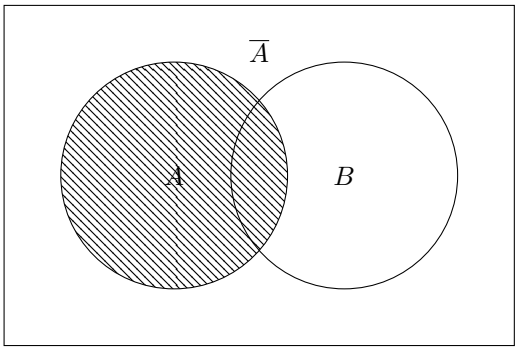

3. **Union ($\mathbf{A \cup B}$)**



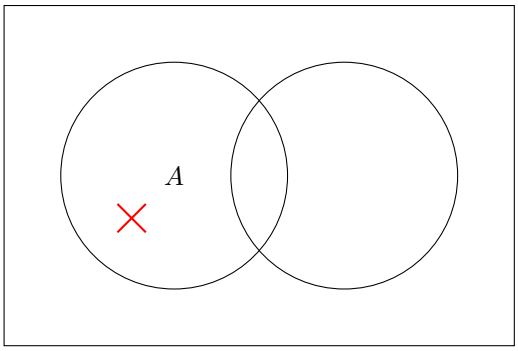

4. **Différence ($\mathbf{A \setminus B}$)**



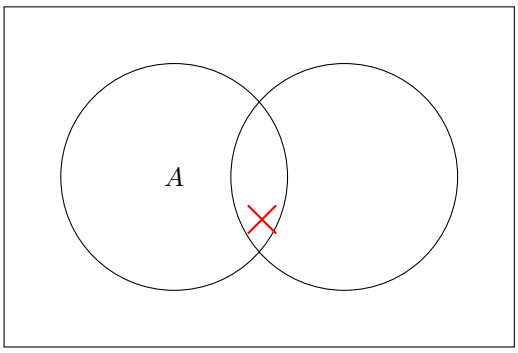
5. **Complément (non A)**




6. **Il existe A sans B**




7. **Il existe A qu'est B**

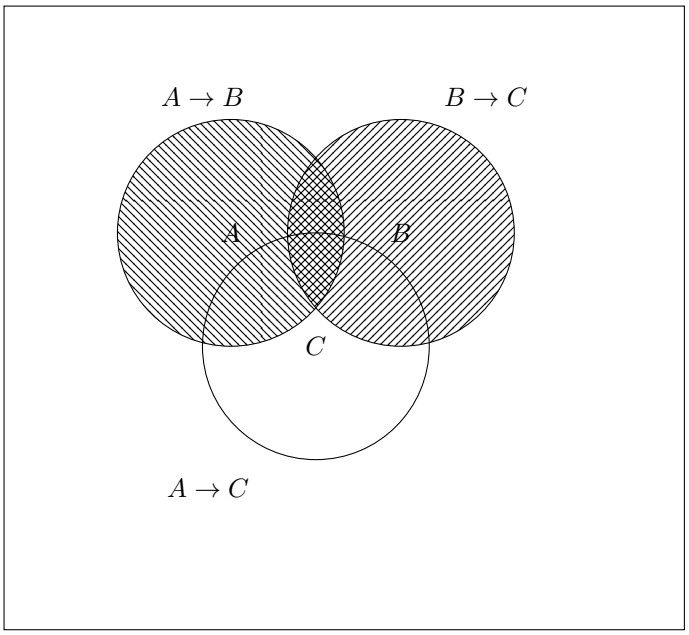


## 2.4 Syllogisme

Le syllogisme est un raisonnement déductif composé de deux propositions appelées prémisses et d'une conclusion. Il suit une structure logique particulière où une conclusion découle nécessairement des prémisses, basée sur des règles formelles. Il est souvent représenté sous forme de "Si A est vrai, et si A implique B, alors B est vrai".

Exemple :

1. **Premisse 1 : A $\rightarrow$ B**
2. **Premisse 2 : B $\rightarrow$ C**



# 3 Code Ocaml

## 3.1 Définition des types

- **formule_log_prop** définit les différentes formes qu'une formule logique propositionnelle peut prendre en OCaml. Il inclut des variantes pour les atomes, les connecteurs logiques tels que "Et", "Ou", "Implication", "Non", etc., ainsi que des variantes pour des quantifications et des opérations sur des listes de formules.

- **formule_syllogisme** représente les formules utilisées spécifiquement pour les syllogismes en OCaml. Il a deux variantes principales : `PourTout` et `IlExiste`, correspondant aux quantificateurs universel et existentiel respectivement, chacun prenant une formule logique propositionnelle.

## 3.2 Formule_Log_Prop.ml

Définit le type des formules de la logique propositionnelle. Il comporte des variantes pour les formules de base comme les atomes, les connecteurs logiques tels que Et, Ou, Non, etc., ainsi que des variantes pour des quantifications et des opérations sur des listes de formules.

- `atome_dans_f` : `formule_log_prop -> string list`
  - Renvoie une liste de chaînes de caractères correspondant à tous les atomes présents dans la formule.

- `est_dans_liste` : `string list -> string -> bool`
  - Vérifie si une chaîne de caractères est présente dans une liste de chaînes.

- `fusion_liste` : `string list -> string list -> string list`
  - Fusionne deux listes en ajoutant à la deuxième liste tous les éléments de la première liste qui ne sont pas déjà présents dans la deuxième liste.

- `eval_Formule_Log_Prop` : `string list -> formule_log_prop -> bool`
  - Évalue une formule logique propositionnelle en fonction d'une interprétation donnée par une liste de chaînes de caractères.

- `table_verite` : `string list -> formule_log_prop -> (string list * bool) list`
  - Renvoie la table de vérité d'une formule sur les atomes issus de la formule elle-même et de l'ensemble alpha (liste de chaînes de caractères).

- `string_of_formule_log_prop_var` : `string -> formule_log_prop -> string`
  - Convertit une formule en chaîne de caractères en représentant les formules comme des prédicats unaires appliqués sur des occurrences de la variable donnée.

## 3.3 Formule_Syllogisme.ml

Définit le type des formules utilisées pour les syllogismes. Il comporte deux variantes : `PourTout` qui prend une `formule_log_prop` et `IlExiste` qui prend également une `formule_log_prop`.

- `string_of_formule` : `formule_syllogisme -> string`
  - Conversion d'une formule en chaîne de caractères en considérant des prédicats unaires appliqués sur des occurrences de la variable 's'.
  - Utilise la fonction `string_of_formule_log_prop_var` pour effectuer la conversion en fonction du type de formule (`PourTout` ou `IlExiste`).
  - Renvoie une chaîne de caractères représentant la formule avec le quantificateur universel pour `PourTout` et le quantificateur existentiel pour `IlExiste`.

## 3.4 Test.ml

- `string_of_diagrams_list` :
  Cette fonction prend une liste de chaînes de caractères représentant des diagrammes et renvoie une chaîne contenant la représentation de chaque diagramme avec son numéro correspondant.

- `string_of_premises` :
  Cette fonction prend une liste de listes de chaînes de caractères représentant des diagrammes de prémisse et renvoie une chaîne contenant la représentation de chaque prémisse avec les représentations de ses diagrammes.

- `test` :
  Cette fonction prend une liste de formules de syllogisme `ps` et une formule de syllogisme `c`. Elle effectue plusieurs étapes :
  - Obtient les atomes (`alpha`) présents dans les formules de `ps`.
  - Génère les diagrammes (`dps`) pour chaque formule de `ps`.
  - Convertit les représentations des diagrammes en chaînes de caractères (`str_dps`).
  - Génère la représentation en chaîne de caractères de la conclusion (`str_dc`).
  - Génère la représentation en chaîne de caractères des diagrammes de la combinaison (`comb`).
  - Concatène toutes ces informations dans une chaîne `s`, puis l'affiche.

### 3.5 DiagVenn.ml

- `string_of_diag` : `diagramme -> string`
  Conversion d'un diagramme en une chaîne de caractères en représentant les ensembles de prédicats avec leur remplissage (`Vide` ou `NonVide`).
- `diag_from_formule` : `string list -> formule_syllogisme -> diagramme list`
  Utilise une table de vérité associée au diagramme et construit de ce dernier une formule sur les prédicats issus de la formule elle-même ou de l'ensemble alpha (liste de chaînes de caractères).
- `conj_diag` : `diagramme -> diagramme -> diagramme option`
  Calcul de la combinaison ou conjonction de deux diagrammes, renvoyant `None` en cas d'incompatibilité.
- `est_compatible_diag_diag` : `diagramme -> diagramme -> bool` Test de compatibilité entre deux diagrammes issus d'une prémisse et d'une conclusion.
- `est_compatible_diag_list` : `diagramme -> diagramme list -> bool`
  Test de compatibilité entre un diagramme issu d'une prémisse et une liste de diagrammes issus d'une conclusion.
- `est_compatible_list_list` : `diagramme list -> diagramme list -> bool`
  Test de compatibilité entre plusieurs diagrammes issus de prémisses et une liste de diagrammes issus d'une conclusion.
- `atome_dans_formules` : `formule_syllogisme list -> string list`
  Liste des atomes présents dans une liste de formules de syllogisme.
- `fusion_pourtout_diag_list` : `diagramme list -> diagramme`
  Fusion des diagrammes représentant tous les éléments de la liste avec pour valeur "Vide".
- `combinaison` : `formule_syllogisme list -> diagramme list`
  Combinaison des diagrammes issus de plusieurs formules de syllogisme.
- `est_compatible_premisses_conc` : `formule_syllogisme list -> formule_syllogisme -> bool`
  Test de compatibilité entre une liste de prémisses et une conclusion.
- `temoin_incompatibilite_premisses_conc_opt` : `formule_syllogisme list -> formule_syllogisme -> diagramme option`
  Identification d'un diagramme de combinaison de prémisses incompatible avec une conclusion s'il existe.
- `temoins_incompatibilite_premisses_conc` : `formule_syllogisme list -> formule_syllogisme -> diagramme list` Récupération des diagrammes incompatibles de combinaison de prémisses avec une conclusion.

# 4 Réponse aux questions

## 4.1 Comment calculer des diagrammes de Venn associés à une valeur f de type formule_syllogisme?

1. **Diagrammes de Venn fonctionnels :**
   - Chaque diagramme de Venn fonctionnel représente un ensemble spécifique basé sur une condition logique ou une table de vérité.
2. **Intersection des ensembles :**
   - Pour calculer la conjonction des diagrammes de Venn 'A' et 'B', il faut trouver la zone où les deux diagrammes se chevauchent ou se croisent. Cette zone représente l'intersection des ensembles.
3. **Analyser le quantificateur :**
   - Si `f` est de la forme `PourTout formule_log_prop`, cela signifie que pour chaque élément dans l'ensemble de valeurs de `formule_log_prop`, la formule doit être vraie.
   - Si `f` est de la forme `IlExiste formule_log_prop`, cela signifie qu'il existe au moins un élément dans l'ensemble de valeurs de `formule_log_prop` pour lequel la formule est vraie.
4. **Utiliser la table de vérité :**
   - La table de vérité associée à `f'` permet d'évaluer la véracité de `f'` pour différentes combinaisons de valeurs de ses atomes. Chaque ligne de la table de vérité représente une telle combinaison.
5. **Interpréter la table de vérité pour les diagrammes de Venn :**
   - Si `f` utilise le quantificateur `PourTout`, alors tous les atomes qui satisfont `f'` doivent être inclus dans le diagramme de Venn associé à `f`.
   - Si `f` utilise le quantificateur `IlExiste`, alors au moins un atome satisfaisant `f'` doit être inclus dans le diagramme de Venn associé à `f`.
6. **Construction des diagrammes de Venn :**
   - Les ensembles représentés dans les diagrammes de Venn sont définis par les valeurs de vérité de `f'` pour différentes combinaisons d'atomes.
   - Chaque ensemble dans le diagramme de Venn peut être construit en fonction de la table de vérité. Les ensembles sont représentés par des zones qui sont vides ou non vides, selon les valeurs de vérité de `f'` pour différentes combinaisons d'atomes.

## 4.2 Comment calculer la conjonction de deux diagrammes de Venn?

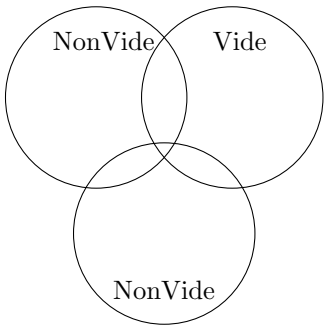
Nous allons utiliser la fonction "conj_diag" qui est chargée de calculer la conjonction de deux diagrammes de Venn. Voici comment elle fonctionne :

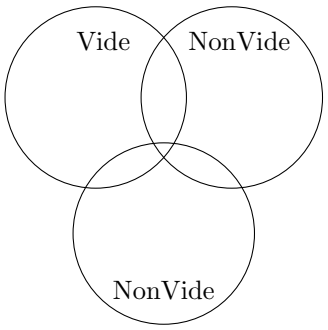
**Comparaison des éléments dans les deux diagrammes:** Chaque élément (ensemble) dans le premier diagramme est comparé à son équivalent (s'il existe) dans le deuxième diagramme.

**Vérification de la compatibilité:** Si un élément existe dans les deux diagrammes, on compare les valeurs associées à ces éléments dans les diagrammes. Les valeurs sont Vide ou NonVide. Si les deux diagrammes définissent la même zone comme vide (Vide) ou non vide (NonVide), alors cette zone peut être présente dans le résultat de la conjonction. Si, cependant, une zone est définie comme vide (Vide) dans l'un des diagrammes et non vide (NonVide) dans l'autre, cela crée une incompatibilité, car les contraintes sont contradictoires.

**Résultat:** Si aucune incompatibilité n'est détectée pendant cette comparaison, un nouveau diagramme est créé, contenant les zones compatibles définies par les deux diagrammes d'origine. Sinon, si une incompatibilité est rencontrée, la fonction renvoie `None` pour indiquer que la conjonction est impossible en raison de contraintes contradictoires.

**Exemple :**
**Diagramme 1 (`d1`):**
Zone A : NonVide
Zone B : Vide
Zone C : NonVide
**Diagramme 2 (`d2`):**
Zone A : Vide
Zone B : NonVide
Zone C : NonVide



Lorsqu'on appele la fonction `conj_diag d1 d2`, elle parcourt les zones correspondantes dans les deux diagrammes pour créer un nouveau diagramme résultant de la conjonction.

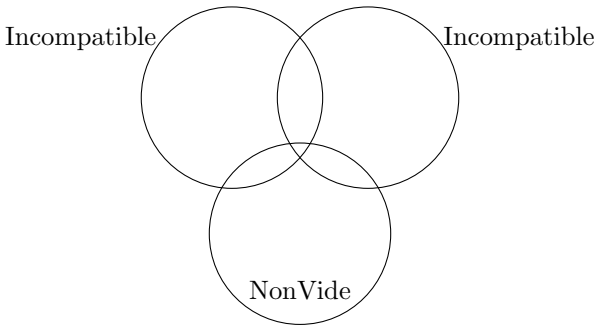
Voici comment cela se passerait :

La fonction commence par examiner la zone A.

Dans `d1`, la zone A est non vide.

Dans `d2`, la zone A est vide.

Comme ces deux diagrammes définissent des états différents pour la même zone, la conjonction est impossible.




La fonction renvoie `None`.

## 4.3 Comment déterminer la compatibilité de deux diagrammes?

**Création des Diagrammes de Venn :**

Les formules logiques sont converties en ensembles de prédicats représentés sous forme de diagrammes de Venn. Chaque prédicat est associé à un ensemble dans le diagramme, soit "Vide" s'il est exclu, soit "NonVide" s'il est inclus.

**Conjonction de Diagrammes :**

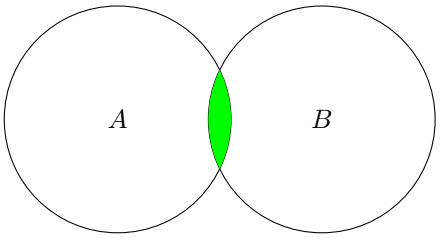
Comme expliqué précedemment, la fonction `conj_diag` calcule la conjonction de deux diagrammes de Venn, représentant une combinaison logique. Elle vérifie la compatibilité des ensembles entre les deux diagrammes pour déterminer la conjonction, renvoyant `None` en cas d'incompatibilité.

**Vérification de la Compatibilité :**

La fonction `est_compatible_diag_diag` vérifie la compatibilité entre deux diagrammes spécifiques. `est_compatible_diag_list` teste la compatibilité d'un diagramme avec une liste de diagrammes. `est_compatible_list_list` vérifie si chaque diagramme d'une liste est compatible avec au moins un diagramme d'une autre liste.

**Illustration de l'Asymétrie de la Compatibilité :**

Considérons deux ensembles :
A représente l'ensemble des nombres premiers.
B représente l'ensemble des nombres impairs.




Tous les nombres impairs (sauf 2)
sont premiers,
mais certains
nombres premiers
ne sont pas impairs

Dans cette illustration, la zone verte représente l'ensemble des nombres impairs inclus dans l'ensemble des nombres premiers. Cela démontre que tous les nombres impairs (à l'exception de 2) sont inclus dans l'ensemble des nombres premiers, tandis qu'il existe des nombres premiers qui ne sont pas impairs, illustrant ainsi l'asymétrie de la relation entre les deux ensembles.

# 5 Conclusion

Ce rapport a exploré en détail les fondements et les applications des diagrammes de Venn dans le cadre de la logique propositionnelle et des syllogismes. En plongeant dans les concepts d'analyse des quantificateurs, de manipulation des tables de vérité, de construction et de manipulation des diagrammes de Venn, il a offert une perspective approfondie sur ces outils visuels cruciaux.

Les fonctions implantées dans ce projet en OCaml ont concrétisé ces concepts en offrant des méthodes pour représenter, calculer et évaluer des diagrammes de Venn associés à des formules logiques ou des syllogismes. Ces outils ont permis la création de tables de vérité, la conjonction de diagrammes, ainsi que la détermination de la compatibilité entre eux.

Ainsi, ce rapport constitue une ressource complète pour appréhender les mécanismes sous-jacents à la représentation visuelle des relations entre ensembles. Il propose une vision claire sur l'utilisation des diagrammes de Venn dans le contexte de la logique formelle. Les explications détaillées, accompagnées d'exemples concrets, et la description des fonctions implémentées, fournissent une base solide pour l'application et la poursuite de l'exploration de ces concepts.

En définitive, ce rapport ouvre la voie à une utilisation plus avancée des diagrammes de Venn dans la logique formelle et offre un point de départ solide pour des recherches ultérieures et des applications pratiques dans ce domaine.