UFR des sciences et techniques - site du Madrillet, Université de Rouen Normandie

Représentation fonctionnelle de diagrammes de Venn de dimension arbitraire

Rapport de projet

Mamadou Lamine DIALLO
Massinissa BRAHIMI

Introduction:

Ce rapport présente la conception et l'implémentation d'un programme en Ocaml dédié à la création de diagrammes de Venn de manière fonctionnelle, permettant de traiter un nombre variable de prédicats et d'évaluer la compatibilité d'une combinaison de prémisses avec une conclusion. L'approche fonctionnelle donc, comme mentionné dans le sujet, repose sur des fonctions partielles, qui associent des ensembles de zones du diagramme à des ensembles de contraintes de remplissage. Ces contraintes, définies sur l'ensemble {Vide, Non Vide}, indiquent si une zone donnée doit être hachurée (Vide) ou comporter une croix (Non Vide). Les zones sans image désignent les zones blanches .

Tout au long de ce document, nous explorerons en détail l'architecture de notre implémentation, présentant les idées et les algorithmes qui sous-tendent notre approche.

Structure et Description :

Le projet est organisé en fichiers, chacun renfermant des fonctions qui contribuent progressivement à la construction des fonctions partielles représentant les diagrammes de Venn.

Fichier : Formule_Log_Prop.ml

Dans ce fichier, nous introduisons deux fonctions clés : la fonction "table_verite" et la fonction “string_of_formule_log_prop_var “.

Cette fonction, de prototype et de spécification :

```
val table_verite : string list -> formule_log_prop -> (string list * bool) list
```

construit la table de vérité de la formule logique *f* sur les atomes issus de *f* ou de la liste *alpha*. La table de vérité est représentée comme une liste de couples *(as, b)*, où *b* correspond à l'évaluation de *f* dans l'interprétation où seuls les atomes contenus dans *as* sont vrais, les autres faux.

L'idée derrière cette fonction est de générer tous les couples d'atomes issus de la combinaison de la liste *alpha* et des atomes de la formule *f*. Elle construit ensuite la table de vérité en évaluant la formule *f* pour chaque combinaison d'atomes.

Nous avons donc défini des fonctions utilitaires qui jouent un rôle essentiel dans la construction de la table de vérité à savoir :

La fonction `eval` ayant pour prototype :
val eval : string list -> formule_log_prop -> bool

Elle permet d'évaluer une formule logique propositionnelle en fonction d'une interprétation donnée par la liste d'atomes `i`.

La fonction `all_interpretations` de prototype :

val all_interpretations : string list -> string list list

Elle permet de calculer toutes les interprétations possibles pour une liste d'atomes donnée en génère, pour ces derniers (les atomes), toutes les combinaisons possibles.

La fonction extract de prototype :

val extract : formule_log_prop -> string list

Elle permet d'extraire la liste des atomes d'une formule logique propositionnelle.

La deuxième fonction de ce module est la fonction "`string_of_formule_log_prop_var`" de prototype :
val string_of_formule_log_prop_var : string -> formule_log_prop -> string

Elle convertit la formule en une chaîne de caractères où les atomes sont représentés comme des prédicats unaires appliqués sur des occurrences de la variable de type string.

Fichier : Formule_Syllogisme.ml

Ce fichier contient une seule fonction de prototype :
val string_of_formule : formule_syllogisme -> string

Elle convertir une formule syllogiste en chaîne de caractère .

Fichier : DiagVenn.ml

Ce fichier constitue le cœur du programme, regroupant les fonctions essentielles permettant la représentation des diagrammes de Venn sous forme de fonctions partielles.

Nous avons la fonction “`string_of_diag`” de prototype :
val string_of_diag : diagramme -> string

Elle convertit un diagramme donné en une chaîne de caractères.

Ensuite nous avons une fonction centrale “diag_from_formule” de signature :
val diag_from_formule : string list -> formule_syllogisme -> diagramme list

Elle réalise la conversion d'une formule syllogistique en une liste de diagrammes de Venn.

Question 1 :

Expliquer comment calculer les diagrammes de Venn associés à une valeur f de type formule_syllogisme en fonction du quantificateur et de la table de vérité de la valeur f’ de type formule_log_prop définissant f.

Réponse :
La fonction à savoir “diag_from_formule” est conçue pour manipuler deux types de quantificateurs, à savoir le quantificateur universel et le quantificateur existentiel.

Quantificateur Universel

Le principe sous-jacent est de hachurer toutes les parties du diagramme associées aux atomes pour lesquels la formule syllogistique est fausse.

Pour accomplir cela, la fonction diag_from_formule consulte la table de vérité associée à la formule logique propositionnelle correspondante. Elle sélectionne les listes des atomes pour lesquels la formule est fausse (booléen `false`) et crée un diagramme unique avec ces listes dans lequel chaque liste représentant ainsi une clé de la map est associée à la valeur *Vide* et stocké ce seul diagramme dans une liste.

Quantificateur Existentiel

En présence d'un quantificateur existentiel, le processus devient un peu plus complexe. La fonction doit générer une liste de diagrammes. Cette fois-ci, les diagrammes correspondent aux atomes pour lesquels la formule est vraie (booléen `true`).

La fonction parcourt la table de vérité et crée un diagramme distinct pour chaque liste d’atomes dont la formule est vraie en associant cette liste représentant la clé de la map à la valeur *Non Vide*. Ces diagrammes individuels sont ensuite regroupés dans la liste finale de diagrammes de Venn.

Ensuite nous avons la fonction : conj_diag d1 d2 calcule la conjonction de deux diagrammes.

Question2 :

Comment se calcule la conjonction de deux diagrammes de Venn fonctionnels ?

Reponse:

Le processus de calcul de la conjonction entre deux diagrammes de Venn fonctionnels, représentés par d1 et d2, peut être exposé de manière plus explicite.

1. **Parcours des Zones du Premier Diagramme (d1)** :
    - Chaque zone du premier diagramme d1 est soigneusement inspectée.
    - L'existence de chaque zone dans le deuxième diagramme d2 est vérifiée.
    - En cas de correspondance, la zone est ajoutée à l'accumulateur en fonction de la cohérence de ses valeurs de remplissage avec celles de d2.
2. **Construction de l'Accumulateur** :
    - Les zones qui satisfont les conditions de correspondance sont consignées dans l'accumulateur.
    - Les zones de d1 qui ne trouvent pas d'équivalent dans d2 sont également incluses dans l'accumulateur, conformément au principe de conjonction.
3. **Intégration des Zones du Deuxième Diagramme (d2)** :
    - Les zones de d2 qui n'ont pas été incluses dans l'accumulateur pendant l'itération avec d1 sont ensuite ajoutées.
    - Le résultat final représente la conjonction des deux diagrammes.

nous avons ensuite la fonction :
est_compatible_diag_diag dp dc teste si le diagramme dp est compatible avec le diagramme dc, c'est-à-dire si les contraintes de dc sont incluses dans celles de dp : les zones vides (respectivement non vides) de dc doivent être vides (respectivement non vides) dans dp.

Question 3:
Expliquer comment déterminer la compatibilité de deux diagrammes en OCaml. Donner un exemple illustrant le fait que la compatibilité n'est pas symétrique.

Réponse :
Le principe fondamental de cette fonction réside dans la vérification de l'inclusion des contraintes du diagramme de conclusion (dc) dans celles du diagramme de prémisse (dp). Plus précisément, chaque zone spécifiée dans dc doit être présente dans dp avec la même valeur de remplissage (vide ou non vide).

1. **Parcours des Zones de la Conclusion (dc)** :

    - Chaque zone définie dans le diagramme de conclusion (`dc`) est examinée.
2. **Vérification dans le Diagramme de Prémisse (dp)** :
    - On vérifie si la même zone existe dans le diagramme de prémisse (`dp`).
    - Si la zone est trouvée, la compatibilité dépend de l'égalité des valeurs de remplissage dans les deux diagrammes.
    - Si la zone n'est pas trouvée dans `dp`, les diagrammes ne sont pas compatibles.
3. **Répétition pour Toutes les Zones de la Conclusion** :
    - Cette étape est répétée pour toutes les zones spécifiées dans le diagramme de conclusion (`dc`).

Le résultat final indique si les contraintes de `dc` sont incluses dans celles de `dp`, déterminant ainsi la compatibilité des deux diagrammes de Venn.

La non-symétrie de la compatibilité entre deux diagrammes de Venn peut être illustrée par un exemple concret. Considérons deux diagrammes, `dp` et `dc`. La compatibilité de `dp` avec `dc` peut différer de la compatibilité de `dc` avec `dp`.

Supposons `dp` les zones suivantes :

- {a;b} -> Vide
- {b} -> Non Vide
- {c} -> Vide

Et que `dc` soit une conclusion avec les zones suivantes :

- {a;b} -> Vide
- {b} -> Non Vide

Exemple d'Incompatibilité Symétrique

1. **Compatibilité dp avec dc** :
    - `est_compatible_diag_diag dp dc` renvoie `true` car toutes les zones de `dc` sont incluses dans `dp`.
2. ****Compatibilité** `dc` avec `dp` (Symétrique)` :
    - `est_compatible_diag_diag dc dp` renvoie `false` car toutes les zones de `dp` ne sont pas incluses dans `dc`. La zone `{c} -> Vide` de `dp` n'est pas présente dans `dc`.

Nous avons ainsi les autres fonctions de teste de compatibilité qui se déduisent de la précédente :

est_compatible_diag_list dp dcs teste si le diagramme dp est compatible avec au moins un des diagrammes de la liste dcs.

est_compatible_list_list dps dcs teste si chacun des diagrammes de la liste dps est compatible avec au moins un des diagrammes de la liste dcs.

est_compatible_premisses_conc ps c teste si chacun des diagrammes de la combinaison des prémisses de la liste ps est compatible avec au moins un des diagrammes de la conclusion c.

Pour élaborer la fonction `est_compatible_premisses_conc`, plusieurs fonctions auxiliaires ont été conçues afin de générer les diagrammes nécessaires pour la vérification de la compatibilité. Chacune de ces fonctions accomplit une tâche spécifique, contribuant ainsi à la modularité et à la clarté du code.

La fonction use_atomes extrait la liste des atomes présents dans les formules de la liste ps. Elle est utilisée pour identifier les atomes associés aux quantificateurs universels et existentiels.

La fonction combine_diag_premisses génère la liste de diagrammes résultant de la combinaison des prémisses. Elle prend en compte les quantificateurs universels et existentiels, utilisant les fonctions diag_from_formule et conj_diag pour créer les diagrammes nécessaires.

La fonction diag_conclusion génère la liste de diagrammes associés à la conclusion c en fonction des atomes présents dans les formules de la liste ps.

Ces fonctions auxiliaires sont essentielles pour construire les diagrammes nécessaires à la vérification de la compatibilité entre les prémisses et la conclusion et plus tard dans la fonction test du fichier Test.ml.

Nous avons ensuite défini les fonctions :

temoin_incompatibilite_premisses_conc_opt ps c renvoie un diagramme de la combinaison des prémisses ps qui n'est pas compatible avec ceux de la conclusion c s'il en existe un, et renvoie None s'il n'en existe pas.

temoins_incompatibilite_premisses_conc ps c renvoie tous les diagrammes de la combinaison des prémisses ps qui ne sont pas compatibles avec ceux de la conclusion c.

Nous avons défini une fonction utilitaire aux deux fonctions précedentes diag_incompatible qui détecte un témoin d'incompatibilité parmi les diagrammes de dp qui ne sont pas compatibles avec ceux de dc.

Fichier : Test.ml

Dans ce dernier fichier regorge une seule fonction test qui, conformément à sa spécification, teste si chaque diagramme de la combinaison des prémisses est

compatible avec au moins un des diagrammes de la conclusion. De plus, cette fonction trace les calculs réalisés et affiche tous les contre-exemples en cas d'incompatibilité. Cette fonction utilise toutes les fonctionnalités des modules précédemment définis, démontrant ainsi l'intégration réussie des différentes parties du programme.

Difficulté rencontrée

La conception et l'implémentation de la fonction `conj_diag` ont constitué un défi majeur au cours du développement du programme. Initialement, une implémentation partielle avait été fournie, mais elle présentait des lacunes significatives, affectant la précision des résultats obtenus. La difficulté principale résidait dans la gestion de toutes les zones des deux diagrammes et leur combinaison de manière exhaustive.

Problème Rencontré

L'implémentation initiale de `conj_diag` parcourait correctement les zones du premier diagramme (d1) pour les stocker dans l'accumulateur en fonction du deuxième diagramme (d2). Cependant, une lacune majeure avait été négligée : la vérification des zones du diagramme d2 qui n'avaient pas encore été ajoutées à l'accumulateur. Ces zones nécessitaient également une inclusion dans le résultat final, mais cela avait été omis initialement.

Solution Adoptée

Une fois que cette lacune a été identifiée lors de tests approfondis, une solution immédiate a été mise en œuvre. La vérification des zones de d2 non encore incluses dans l'accumulateur a été ajoutée.