Projet MIDL 1

Implémentation de la procédure de décision dans la théorie des Ordres Denses

DAVID--MULLER Robin

MARCHAND Nicolas

VASSEUR-LAURENS Odin

Aperçu du Projet

Contenu :

Détails du projet :

- **UE :** Projet MIDL 1
- **Date de rendu :** 09/02/2026
- **Langage :** OCaml

Lot 1

Ce projet s'inscrit dans le cadre de la vérification de formules logiques pour la théorie des ordres denses sans bornes.

Le problème central est de déterminer la vérité d'une formule mathématique contenant, dans un premier temps, des comparaisons et des quantificateurs. La difficulté dans la vérification de ces formules réside dans le fait que ces quantificateurs portent sur des domaines « infinis », ne permettant pas de vérifier naïvement chaque solution par un algorithme dit de « bruteforce ». Nous résolvons donc ce problème par la manipulation algébrique des formules, permettant une élimination systématique des quantificateurs.

Suite à l'approbation de nos professeurs, nous avons choisi de développer notre projet dans le langage OCaml, contrairement aux recommandations initiales d'utiliser Python. Il y a deux raisons à cela :

- La représentation algébrique d'une formule se fait beaucoup plus naturellement sur OCaml, grâce au principe mathématique de « types inductifs », naturellement introduits par OCaml. Ceux-ci, combinés au très puissant « pattern matching », nous permettent d'obtenir un code bien plus lisible et court qu'en Python.
- Le troisième semestre de MIDL nous a introduit à l'UE ILU1. Nous y avons appris les bases d'OCaml, et ce projet nous a permis de nous familiariser avec ce langage dans un cadre approprié ainsi que d'en apprendre certaines fonctionnalités telles que les types inductifs et le pipelining.

Nous avons donc commencé par traduire le script python fourni par nos professeurs dans notre langage de choix.

Ensuite, nous avons assez facilement implémenté le prétraitement. Notre choix a été de séparer chaque clause de notre forme normale disjonctive dans une liste après avoir poussé les quantificateurs existentiels dans celle-ci.

Nous voilà donc en train de traiter un ensemble de disjonctions. Nous avons donc choisi de classifier dans un dictionnaire toutes les contraintes sur chaque variable quantifiée : égalités, majorantions, minorations, formules indépendantes (χ). Maintenant que cela est fait, nous avons appliqué toutes les règles de suppression de quantification universelle, à l'exception de la seconde (« si ψ contient $x < x$: $(\exists x.\psi) \leftrightarrow \bot$ ») que nous avons supprimé lors de notre implémentation du Lot 2 ; cela sera donc détaillé plus tard.

Lot 2

Maintenant que nous pouvons appliquer notre procédure à toute proposition de la théorie des ordres denses, nous pouvons nous pencher sur l'arithmétique des nombres rationnels. On remarque que peu de choses changent, il n'y a pas de collision entre le prétraitement et la complexification des (in)égalités « générales » qui se résumaient à des comparaisons entre deux variables.

Ce qui change le plus, c'est que l'on ne peut pas simplement avoir de majoration/minoration sur une variable x en regardant le terme de gauche ou le terme de droite. Dans le cas d'une égalité, c'est facile ! Nous avons appris la méthode du pivot de Gauss qui convient parfaitement à cette situation, il nous suffira de l'appliquer à notre égalité pour isoler x. Dans le cas d'une inégalité, nous appliquons la même chose, la seule différence est qu'il faut faire attention aux changements de signe (c'est ce que l'on appelle l'« Élimination de Fourier-Motzkin »). Ainsi, nous obtenons des majorations/minorations de x par rapport aux autres variables.

Supposons l'inégalité : $A < B$. Dans ce cas, celle-ci est équivalente à $A - B < 0$. Nous pouvons effectuer une extraction du coefficient x dans $A - B$ (cela pourrait s'apparenter à une sorte de division euclidienne d'un polynôme de degré 1 par x). Ainsi, nous aurons un coefficient scalaire q et un reste R ne dépendant pas de x. Cela nous donnera : $qx + R < 0 \iff qx < -R$.
- Dans le cas où $q = 0$, nous avons une proposition indépendante de x : $R < 0$.
- Dans le cas où $q > 0$, nous avons $x < -\frac{R}{q}$
- Dans le cas où $q < 0$, nous avons $x > -\frac{R}{q}$

Le cas de l'égalité s'effectue de la même manière.

C'est cette étape de normalisation qui rend l'étape 3.2 mentionnée précedemment (suppression triviale de $x < x$) obsolète : notre algorithme transforme naturellement $x < x$ en $0 < 0$, qui est identifié comme une formule indépendante (fausse) et traitée lors de l'étape 3.6.

Nous avons d'abord introduit une fonction `simplify_term` qui simplifiera au maximum les formules algébriques, notamment à l'aide de la règle d'associativité, pour simplifier les calculs. Ensuite, nous avons créé la fonction d'extraction de coefficients `get_coeff` pour finalement l'appliquer à la fonction de normalisation `isolate`.

Il a aussi fallu changer les fonctions implémentées précédemment : après avoir supprimé l'étape 3.2, nous avons rajouté les comparaisons arithmétiques à la fonction `simplify`. Nous pensions avoir fini lorsque l'exemple de proposition « $\exists x.x + 5 > x + 2$ » ne fonctionnait pas. Le dysfonctionnement venait de la fonction `check_var` car elle ne devait plus seulement regarder à droite et à gauche de l'inégalité, mais bien parcourir récursivement l'arbre syntaxique créé par les expressions arithmétiques à gauche et à droite. Ainsi, après avoir fait ce changement, notre procédure s'est mise à fonctionner sur tous nos tests.

En dernière retouche, nous avons finalement remplacé nos flottants par un type `Rational` afin de représenter nos nombres par des fractions, ce qui nous permet de ne pas avoir d'approximations lors des calculs. Par exemple : $\exists x.x + 0.3 < x + 0.1 + 0.2$ qui est faux, mais renvoyait vrai car $0.1 + 0.2$ est stocké comme « 0.30000000000000044 » au lieu de $0,3$:

```
utop # 0.1 +. 0.2;;
- : float = 0.30000000000000044
```

Ouvertures

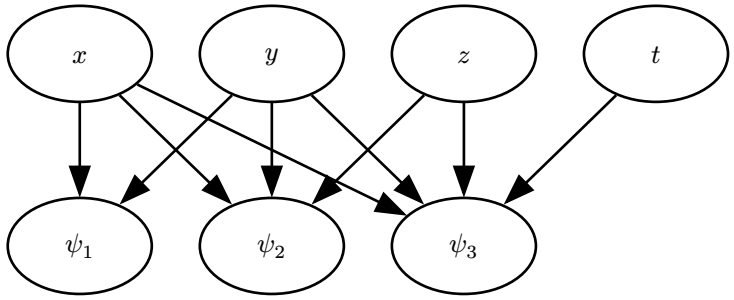
- Bien que notre implémentation de l'élimination des quantificateurs soit maintenant fonctionnelle pour l'Arithmétique Linéaire Rationnelle, elle repose sur une version standard de l'algorithme de Fourier-Motzkin. Cette méthode est « naïve » car elle peut créer des inégalités redondantes, et en crée beaucoup. La page Wikipedia nous introduit alors à une accélération de l'algorithme créée par Jean-Louis Imbert en 1990 qui permet d'éliminer certaines contraintes selon la manière dont elles ont été construites, réduisant considérablement les formules intermédiaires de la procédure. L'intégration de cet algorithme aurait cependant nécessité une refonte profonde des structures de données que l'on utilise car elle nécessite une traçabilité des inégalités prises en compte.
- Nous aurions pu généraliser notre algorithme à la Théorie des corps réels clos, qui aurait nécessité l'implémentation de l'algorithme de Décomposition Algébrique Cylindrique de Collins.
- Un cadre d'étude intéressant est aussi l'Arithmétique Linéaire Entière, dite de « Presburger », beaucoup plus proche de l'Arithmétique Linéaire Rationnelle, à l'exception de

Lot 3

Élaboration de la méthode

Il nous est demandé de faire un jeu consistant pour le joueur J de trouver un exemple validant une formule satisfiable ou invalidant un résultat non satisfiable. L'ordinateur O doit alors trouver une stratégie pour bloquer les choix du joueur. Dans un premier temps, l'ordinateur doit vérifier la véracité de la formule en appliquant totalement la suppression des quantificateurs. Après avoir enregistré l'ordre des quantificateurs dans la formule de base (sous forme prénexe), il faut que O ait une bonne vue d'ensemble des contraintes imposées à chaque variable. Notre implémentation des clauses après le prétraitement est déjà parfaite, elle représente tout ce dont on a besoin. Raisonnons par l'exemple :

Soit la formule $\exists x.\forall y.\exists z.\forall t.P(x,y,z,t)$ avec le prédicat $P(x,y,z,t) \equiv \psi_1(x,y) \vee \psi_2(x,y,z) \vee \psi_3(x,y,z,t)$ où les ψ_i sont des prédicats représentants un ensemble de disjonction. Représentons l'arbre $(\{x,y,z,t\} \cup \{(\psi_i)\}, \{(a,\psi) \mid \psi \text{ dépend de } a\})$:



Au premier tour, J choisit une valeur valide pour tous les ψ. O va donc devoir choisir une valeur qui va influencer ψ_2, ψ_3. Son intérêt est d'invalider tous les ψ_i. Il est obligé d'agir en priorité sur les variables pour lesquelles y est la seule variable dans son camps à avoir un rôle sur l'invalidité, c'est-à-dire ψ_1 et ψ_2. Il peut éventuellement choisir des valeurs qui servent à invalider ψ_3, mais ce n'est pas la priorité.

- Sur ψ_1, tout ne sera que sous une forme équivalente à une égalité/inégalité linéaire sur y. Notons $E_1 = \{y \in \mathbb{Q} \mid \psi_1(x,y)\}$ l'ensemble des valeurs vérifiant.
- De même pour $\psi_2(x,y,z)$, qui dépend encore de z (choisi par J au prochain tour). Pour bloquer le joueur, O doit choisir y tel qu'il devienne impossible pour J de trouver un z satisfaisant ψ_2 plus tard. Notons $E_2 = \{y \in \mathbb{Q} \mid \exists z \in \mathbb{Q} : \psi_2(x,y,z)\}$ l'ensemble des valeurs.

Ainsi, si $E_1 \cup E_2 \neq \mathbb{Q}$, alors on peut prendre $y \in \mathbb{Q} \setminus (E_1 \cup E_2)$, sinon J a gagné. On peut facilement vérifier si $E_1 \cup E_2 = \varnothing$ en éliminant les quantificateurs de la formule. Si $E_1 \cup E_2 \neq \mathbb{Q}$, on peut donc trouver des valeurs qui vont falsifier ψ_1 et ψ_2. Cela ressemble à de l'algèbre linéaire : en supposant qu'il n'y a que des égalités, on tombe sur un sous-espace affine (ici, de dimension 1), alors rajouter des inégalités va créer une forme géométrique. On peut parler de « polyèdre convexe ». Trouver un point extérieur est intuitif : le polyèdre étant défini par des « facettes », il suffirait de choisir une valeur de y qui permet d franchir l'une de ces bornes. L'ordinateur peut en réalité prendre directement l'union des polyèdres/sous-espaces affines générés par ψ_1, ψ_2, ψ_3 afin de les invalider par la théorie des ordres denses sans bornes; l'élimination des quantificateurs va projeter l'union de formes géométriques et la projeter sur l'axe de y. Ainsi, O gagne forcément, car la subsitution d'une variable quantifiée par une constante n'est qu'une projection de cet ensemble de formes sur une dimension plus faible. Il peut ainsi jouer une valeur quelconque jusqu'à la fin. Cette approche géométrique (intersections de polyèdres/sous-espaces affines et projections) correspond exactement à ce que calcule l'algorithme d'élimination des quantificateurs avec Fourier-Motzkin et Gauss-Jordan implémentés dans le Lot 2.

L'ensemble I récupéré en sortie de l'élimination de $\exists z.\forall t.\psi_1(x,y) \vee \psi_2(x,y,z) \vee \psi_3(x,y,z,t)$ est la représentation de l'union des polyèdres et/ou espaces affines représentés par les clauses $(E_1 \cup E_2 \cup E_3)$ projetée sur l'axe (Oy). Il suffit de prendre $y \in \mathbb{Q} \setminus I$ pour gagner.

Pour terminer, supposons que O soit dans le rôle inverse (il doit prouver une existence). Alors il suffit de faire la même chose, mais en prenant une valeur dans I, et perdra si la projection sur l'axe (Oy) est l'ensemble vide. Cependant, dans ce cas, il pourrait être mené à choisir une valeur d'existence lors du choix de la valeur de t.

Méthode

Ainsi, la méthode est finalement assez simple :

Si l'on note l'ensemble des variables $(x_i)_{1 \leqslant i \leqslant n}$ et $(\psi_i)_{1 \leqslant i \leqslant m}$ l'ensemble des clauses de la DNF de P. Si l'ordinateur est au tour k, les variables x_i sont fixées pour tout $i < k$. Ainsi, on élimine les quantificateurs des variables $x_{k+1}, ..., x_n$ de la proposition de départ en substituant les $k-1$ premières variables par les valeurs choisies. Supposons que nous sommes au tour de O.

1) Si O doit réfuter la formule :
   a) Si le résultat de l'élimination donne $\top$, O a perdu, il joue une valeur par défaut. Si le résultat donne $\bot$, il a gagné, il joue aussi une valeur par défaut. Sinon, Notons I l'intervalle représentée par le résultat ; qui est soit une union d'intervalles, soit un singleton.
   b) O choisit $x \in \mathbb{Q} \setminus I$, J n'a plus la possibilité de gagner (O a « neutralisé » J en falsifiant chaque clause, comme dans l'exemple).
2) Si O doit valider la formule :
   a) Si le résultat de l'élimination donne $\bot$, O a perdu, il joue une valeur par défaut et le fera jusqu'à la fin de la partie. S'il donne $\top$, il joue aussi une valeur par défaut. Sinon O choisit $x \in I$.

Manuel

Pour lancer le code dans un REPL, vous pouvez exécuter la commande : `./run_repl.sh`. Sinon, vous pouvez lancer le REPL manuellement en exécutant `utop -init syntax.ml`

Construction d'une formule

Nous avons créé des fonctions macro permettant une construction assez simple des propositions. Voici un tableau les regroupant :

Constructeur logique	Équivalent
$\top$	`top`
$\bot$	`bottom`
$x < y$	`lt (var "x") (var "y")`
$x = y$	`equal (var "x") (var "y")`
$P \vee Q$	`disj P Q`
$P \wedge Q$	`conj P Q`
$\neg P$	`notf P`
$P \to Q$	`implies P Q`
$\exists x.P$	`exists "x" P`
$\forall x.P$	`forall "x" P`
x (variable)	`var x`
$n \in \mathbb{Z}$	`val_ n`
$\frac{p}{q}$ avec $(p, q) \in \mathbb{Z}^2$	`frac p q`
$x + y$	`add x y`
$x - y$	`sub x y`
$a \cdot x$ avec $a \in \mathbb{Z}$	`mul a x`
$\frac{p}{q} \cdot x$	`mul_frac p q x`

L'exécution de la procédure se fera via la fonction `final_test` avec le nom de la formule (voir exemples).

Exemples

- $\forall x.\forall y.(x < y \rightarrow \exists z.(x < z \wedge z < y))$ s'écrira :

```
let density = forall "x" (
  forall "y" (
    implies (
        lt (var "x") (var "y")
      )
      (
        exists "z" (
          conj (
              lt (var "x") (var "z")
            )
            (
              lt (var "z") (var "y")
            )
          )
        )
      )
    )
```

- $\forall x.\exists y.(x < y)$ s'écrira :

```
let no_sup_extremum = forall "x" (exists "y" (lt (var "x") (var "y")))
```

Exemples d'exécution de la procédure :

```
final_test density "Densité"
```

Qui renverra :

```
Nom de la formule : "Densité"
Input initial : (∀x.(∀y.(¬(x < y) ∨ (∃z.((x < z) ∧ (z < y))))))
Output final : T
```