

Rapport du Projet 3A

Hugo Barreiro M1 MPRI
Sous la direction de Samuel Mimram

1ère période : Septembre - Décembre 2024

Table des matières

1	Introduction	3
2	État de l'art	3
3	Problème traité et solution	5
3.1	Problème	5
3.2	Première approche	5
3.3	Orientation	6
3.4	Générateur	6
3.5	Implications	7
4	Conclusion	7
4.1	Résultats	7
4.2	Perspectives	8
4.3	Remerciements	8
	Références	9
	Annexes	10

1 Introduction

Le mathématicien *Terence Tao*, le 25 septembre 2024, publie sur son *Blog* un projet collaboratif visant à cartographier les relations entre différentes théories équationnelles de Magmas. En d’autres termes, il s’agit de déterminer toutes les implications engendrées par un ensemble donné d’équations.

On peut examiner manuellement toutes les implications entre les équations, deux à deux. Cela reste envisageable pour un petit ensemble d’équations, mais devient rapidement impraticable lorsque l’ensemble s’agrandit. En effet, *Terence Tao* avait initialement analysé, à la main, un ensemble de 11 équations, avant de s’attaquer à un ensemble beaucoup plus vaste, composé de 4694 équations.

C’est ici qu’intervient l’aspect collaboratif du projet. Ainsi, chacun était libre de proposer ses solutions, à condition qu’elles soient rédigées en *Lean*, permettant ainsi de vérifier automatiquement leur correction. De plus, face au très grand nombre d’implications à considérer (22028942), il était essentiel de concevoir une solution automatisable et informatisée.

C’est alors que l’on s’est tourné vers la réécriture de termes, notamment la méthode de *Complétion de Knuth-Bendix*. Cette dernière permet de déterminer les équations impliquées par une équation donnée, tout en fournissant les étapes de calcul nécessaires pour parvenir à ce résultat. On peut ainsi envisager de générer directement les preuves en *Lean* pour chaque implication.

2 État de l’art

Afin d’apprendre les bases de la réécriture, je me suis intéressé au livre *Term Rewriting and All That*, de *Franz Baader* et *Tobias Nipkow*.

La réécriture est une branche de l’informatique théorique qui s’intéresse à la transformation d’objets syntaxiques en appliquant une succession de règles. Cela peut concerner des mots, des termes, des lambda-termes, des programmes, des preuves, etc. Ainsi, la réécriture est utilisée dans des domaines tels que la preuve automatique, l’optimisation de programmes ou encore la sémantique formelle des langages.

Ce qui distingue la réécriture de la logique équationnelle, c’est que les équations sont orientées, de gauche à droite. Autrement dit, on ne peut remplacer que le côté gauche par le côté droit, mais pas l’inverse. Cela constitue un modèle de calcul Turing-complet, proche de la programmation fonctionnelle.

Un système de réécriture est constitué d’un ensemble de générateurs et de règles de réécriture. Les générateurs définissent la forme des termes. Une règle de réécriture s’écrit sous la forme $l \rightarrow r$, ce qui signifie que l’on peut remplacer chaque occurrence de l par r dans un terme. Par exemple, pour les mots, si l’on dispose d’un alphabet $\Sigma = a, b, c, \dots$ et d’une règle $aa \rightarrow b$, on peut réécrire le mot $bcaacb$ en $bcbcb$.

Cependant, un système de réécriture peut conduire à des réécritures qui ne terminent pas. Par exemple, si l'on dispose de la règle $a \rightarrow ab$ et que l'on applique cette règle au mot a , on obtient $a \rightarrow ab \rightarrow abb \rightarrow abbb \rightarrow \dots$, ce qui montre que la réécriture ne termine pas. Malheureusement, la propriété de terminaison pour un système fini de réécriture est souvent un problème indécidable.

Pour garantir la terminaison, on peut essayer de construire un ordre strict bien fondé $<$ tel que, si $t \rightarrow t'$, alors $t > t'$. Un tel ordre est appelé ordre de réduction. Un exemple bien connu est l'ordre lexicographique, qui, dans le cas des mots, oriente une règle vers le mot le plus court s'il existe une différence de longueur, ou vers le mot le plus petit au sens du dictionnaire.

De plus, un système de réécriture n'est pas forcément confluent a priori. Par exemple, si l'on dispose des règles $ab \rightarrow a$ et $ba \rightarrow b$, on peut obtenir $aba \rightarrow aa$, mais aussi $aba \rightarrow ab \rightarrow a$, avec aa et a comme formes normales. Cela montre qu'un système de réécriture n'est pas toujours confluent.

Pour vérifier si un système est confluent, il suffit de vérifier sa confluence locale dans le cas où il termine (selon le lemme de Newman). Dans ce cas, tester la confluence est décidable : il suffit d'examiner un nombre fini de configurations appelées paires critiques. Par exemple, pour un système composé uniquement des règles $ab \rightarrow \epsilon$ et $ba \rightarrow \epsilon$, il faut vérifier la confluence locale pour les mots aba et bab .

Ensuite, on peut rendre un système de réécriture confluent s'il termine, grâce à la *Complétion de Knuth-Bendix*, qui repose en partie sur l'étude des paires critiques. Ainsi, tout système qui termine peut devenir confluent. Un système qui termine et qui est confluent est appelé convergent, c'est-à-dire que la réécriture termine et produit toujours le même résultat.

L'objectif est donc d'obtenir un système de réécriture qui termine, afin de le compléter pour qu'il devienne confluent, et donc convergent. Comme mentionné précédemment, on peut tenter de construire un ordre de réduction. Si cela échoue, on peut avoir recours aux transformations de Tietze.

Il existe quatre transformations de Tietze :

1. Ajouter une règle de réécriture qui peut être déduite des règles déjà présentes dans le système.
2. Supprimer une règle de réécriture si elle est dérivable à partir des autres règles.
3. Ajouter un générateur (par exemple, dans le cas des mots, un nouvel élément de l'alphabet). Par exemple, si l'on dispose de l'alphabet $\Sigma = \{a, b\}$ et de la règle $aa \rightarrow b$, on peut ajouter c à l'alphabet et introduire la règle $aa = c$.
4. Supprimer un générateur s'il est dérivable à partir des autres générateurs.

3 Problème traité et solution

L’archive du code est accessible sur *GitHub*. Le calcul des paires critiques et le fonctionnement de la *Complétion de Knuth-Bendix* sont expliqués en détail en annexes.

3.1 Problème

Avant de s’attaquer à la réécriture des termes pour répondre au problème posé par *Terence Tao*, nous avons d’abord étudié la réécriture des mots afin de poser des bases solides. L’objectif est d’observer le comportement de la réécriture sur des objets plus simples que les termes : en l’occurrence, les mots.

Ainsi, nous avons mis en œuvre la méthode de *Complétion de Knuth-Bendix* et généré le diagramme de Hasse résumant les implications entre les différentes équations. Notre démarche est la suivante :

Pour chaque équation de notre ensemble, nous formons un système composé uniquement de cette équation, que nous complétons ensuite à l’aide de la *Complétion de Knuth-Bendix*. Le système complété est alors utilisé pour identifier les équations impliquées. Pour cela, chaque côté de l’équation est normalisé en appliquant les règles de réécriture. Si les deux côtés normalisés sont identiques, alors l’équation est impliquée par le système de réécriture, et donc par l’équation d’origine.

Informellement, la *Complétion de Knuth-Bendix* peut être vue comme une procédure visant à ajouter au système de réécriture toutes les équations déductibles du système de base, afin de rendre ce dernier convergent. Ainsi, en normalisant les deux côtés d’une équation à l’aide d’un système complété, on cherche à vérifier si l’égalité est obtenue en tenant compte de toutes nos hypothèses. Par ailleurs, puisque le système de réécriture est convergent, la forme normale est unique ; si les deux côtés ne sont pas égaux, on sait alors que l’équation n’est pas impliquée par le système.

Cependant, un problème peut survenir : la *Complétion de Knuth-Bendix* ne termine que si le système de réécriture termine. Or, comme nous l’avons constaté, ce n’est pas toujours le cas. Par conséquent, si la complétion ne termine pas, il sera nécessaire de trouver des moyens de garantir sa terminaison. Une fois tous les systèmes complétés, il ne restera plus qu’à générer le graphe des implications.

Nous nous concentrerons plus particulièrement sur l’ensemble des mots de longueur au plus 3, avec l’alphabet $\{a, b\}$. De plus, nous considérerons les équations à symétrie et à renommage près.

3.2 Première approche

La première étape a consisté à implémenter la méthode de *Complétion de Knuth-Bendix*, sans adopter de techniques particulières pour l’orientation ou les générateurs. Cependant, nous avons introduit quelques optimisations.

En effet, toutes les règles de réécriture sont normalisées des deux côtés chaque fois qu’une nouvelle règle est ajoutée, ce qui favorise la terminaison. De plus, les règles qui se réduisent à elles-mêmes, comme $a \rightarrow a$, sont systématiquement supprimées.

Malgré ces précautions, certains systèmes ne parviennent pas à être complétés (environ 60% des systèmes sont complétés). Dans certains cas, la normalisation semble ne pas terminer ; dans d’autres, le nombre de paires critiques explose. Ainsi, si un seuil prédéfini pour le nombre de pas de normalisation ou le nombre de paires critiques est atteint, nous abandonnons la complétion.

3.3 Orientation

Jusqu’ici, l’orientation des règles n’était pas prise en compte. Il est possible de remédier à cela et d’espérer obtenir de meilleurs résultats. Une première approche consiste à orienter toutes les règles à chaque étape vers le mot le plus court, s’il existe, ou, à défaut, à ne pas leur attribuer de direction particulière. Avec cette méthode, environ 70% des systèmes sont complétés.

Une amélioration consiste à implémenter l’ordre lexicographique. Cela revient à orienter les règles vers le mot le plus court, s’il existe, ou, sinon, vers le mot le plus petit selon l’ordre du dictionnaire. Cette approche permet de compléter environ 80% des systèmes. Cependant, même cet ordre bien connu en réécriture ne permet pas de compléter tous les systèmes. Il devient alors nécessaire d’explorer des ordres spécifiques pour les systèmes restants.

Après de nombreux tests, il apparaît que des ordres contre-intuitifs peuvent parfois permettre de terminer la complétion. Par exemple, orienter systématiquement vers le mot le plus long ou le plus court, ou encore vers le mot le plus grand ou le plus petit au sens du dictionnaire. Dans certains cas, orienter les règles vers le mot le plus long empêche la formation de nouvelles paires critiques (le système était déjà complet), tandis que dans l’autre sens, la complétion ne parvenait pas à terminer.

Ainsi, aucun ordre ne semble garantir la terminaison de tous les systèmes de réécriture. Une approche pragmatique consiste à tester la complétion avec plusieurs ordres successifs, en espérant qu’un ordre convienne pour chaque système. Avec cette méthode, environ 95% des systèmes sont complétés.

3.4 Générateur

Parmi les quatre systèmes restants à ce stade, figure l’équation $aba = bab$, bien connue dans le domaine de la réécriture. En effet, il n’existe aucune manière d’orienter cette règle pour permettre la terminaison de la complétion.

Cependant, une méthode permet de compléter ce système : utiliser les transformations de Tietze, en particulier l’ajout de générateurs. Dans ce cas précis, il suffit d’ajouter le générateur c et la règle de réécriture $ab = c$. Avec cette modification, la

complétion parvient effectivement à son terme.

Malheureusement, il n'existe pas de méthode systématique pour déterminer quel générateur et quelle règle ajouter afin de garantir la terminaison de la complétion. Par conséquent, pour les systèmes de réécriture suivants, nous avons expérimenté en ajoutant un générateur et une règle de manière arbitraire.

Prenons l'exemple mentionné plus haut. Si l'alphabet contient déjà a et b , on introduit c comme nouveau générateur. Ensuite, nous testons différentes règles : nous commençons par ajouter $a = c$. Si cela ne fonctionne pas, nous essayons $ab = c$, et si cela échoue également, nous tentons $aba = c$. Enfin, si ces essais ne donnent toujours pas de résultat concluant, nous testons avec le côté droit de l'équation.

Grâce à cette technique, nous parvenons à compléter 100% des systèmes de réécriture.

3.5 Implications

À présent, nous avons réussi à compléter tous les systèmes de réécriture. Nous pouvons alors examiner quelles équations impliquent quelles autres équations. Pour ce faire, nous prenons une équation ainsi que son système de réécriture complété. Ensuite, pour chaque autre équation, nous normalisons ses deux côtés à l'aide des règles de réécriture du système. Si les deux côtés sont égaux, alors il y a implication. Sinon, l'équation associée au système n'implique pas l'équation en question.

Une fois que nous avons déterminé toutes les implications, nous pouvons générer le diagramme de Hasse. Pour cela, il faut supprimer les implications qui apparaissent en double par transitivité. Il suffit ensuite d'effectuer une exploration récursive des implications de chaque règle pour simplifier correctement les relations.

Enfin, nous pouvons utiliser Graphviz, notamment son outil *Finite Automaton*, pour représenter simplement le diagramme final. Nous générons également un fichier annexe qui liste les équations n'apparaissant pas dans les implications, c'est-à-dire celles qui n'impliquent aucune autre équation et qui ne sont impliquées par aucune équation.

Une observation concernant le diagramme de Hasse des mots de longueur au plus 3 engendrés par l'alphabet $\Sigma = \{a, b\}$: filtrer les mots à symétrie et renommage près suffit pour obtenir un graphe sans cycle, c'est-à-dire un préordre acyclique sur l'ensemble, ce qui n'est pas évident au départ.

4 Conclusion

4.1 Résultats

L'implémentation de la *Complétion de Knuth-Bendix* et les expérimentations sur les ordres de réduction et les générateurs nous ont permis de compléter automatiquement

tous les systèmes que nous avons étudiés sur les mots. Ainsi, nous avons pu déterminer toutes les implications engendrées par ces ensembles et générer les diagrammes de Hasse.

Nous avons ainsi pu montrer un premier résultat : tout système de réécriture d’une règle composée de mots de longueur au plus 3 peut être complété de façon convergente.

Nous avons également montré un deuxième résultat : la symétrie et le renommage suffisent pour obtenir un préordre acyclique sur cet ensemble.

4.2 Perspectives

Un objectif à court terme est d’étudier la *Complétion de Knuth-Bendix* pour les mots de longueur au plus 4. Un autre objectif est d’étudier plus en détail le diagramme de Hasse pour les mots de longueur au plus 3, notamment le sous-graphe homogène, le sous-graphe des règles égales par renversement et le graphe des présentations à plusieurs relations.

À plus long terme, deux options s’offrent à nous : continuer l’étude des mots ou revenir au problème posé par *Terence Tao*. Dans le premier cas, on se pencherait sur le problème ouvert de la décidabilité du mot pour les monoïdes mono-relationnés. Cela passerait notamment par la compréhension concrète des limitations de la calculabilité du problème, et en particulier par l’étude du papier [NB21]. Dans le deuxième cas, on pourrait essayer d’appliquer nos méthodes de complétion sur les termes, puis d’implémenter la génération automatique des preuves en *Lean*.

4.3 Remerciements

Je tiens à remercier Samuel Mimram pour son encadrement et son aide tout au long de cette première période de projet, qui s’est révélée particulièrement enrichissante.

Références

- [BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
- [NB21] Carl-Fredrik Nyberg-Brodde. The word problem for one-relation monoids : A survey, 2021.
- [Tao24] Terence Tao. A pilot project in universal algebra to explore new ways to collaborate and use machine assistance? Blog, 2024.

Annexes

Paires critiques

Dans cette section, nous expliquons en détail le processus d'obtention des paires critiques.

Les paires critiques sont calculées à partir de deux règles, r_1 et r_2 . Pour chaque règle, il est nécessaire d'extraire le mot qui peut être réécrit en un autre. Par exemple, pour la règle $a \rightarrow b$, on extrait a . Ces mots seront notés w_1 et w_2 .

Ensuite, nous superposons w_1 sur w_2 afin d'identifier tous les mots qui se superposent correctement. Dans l'exemple suivant, le caractère $_$ représente une position vide. Par exemple, avec les règles $r_1 : ab \rightarrow c$ et $r_2 : ba \rightarrow d$, nous essayons de superposer $ab_$ sur $_ba$. Cette superposition est correcte, car a et $_$ sont compatibles, b et b sont égaux, et $_$ et a sont compatibles. Ainsi, nous obtenons le mot aba . En revanche, superposer ab sur ba ne donne pas une correspondance valide. Ensuite, nous superposons $_ab$ sur $ba_$, ce qui est une superposition correcte, nous donnant le mot bab .

Pour chaque mot obtenu, nous calculons toutes les formes normales en utilisant les deux règles r_1 et r_2 . Les ensembles des formes normales obtenues sont notés n_1 et n_2 . Les paires critiques sont ensuite calculées de la manière suivante : $\{\forall fn_1 \in n_1, \forall fn_2 \in n_2, (fn_1, fn_2)\}$. En reprenant l'exemple précédent, aba se réécrit en ca avec r_1 et en ad avec r_2 , tandis que bab se réécrit en bc avec r_1 et en db avec r_2 . Nous obtenons alors les paires critiques (ca, ad) et (bc, db) .

Complétion de Knuth-Bendix

Dans cette section, on va expliquer plus en détail le fonctionnement de la *Complétion de Knuth-Bendix*.

La *Complétion de Knuth-Bendix* est calculée à partir d'un système de réécriture et d'un ordre de réduction. La première étape consiste à orienter toutes les règles du système en fonction de l'ordre de réduction donné. Ensuite, on ajoute dans une queue toutes les règles du système de réécriture.

Tant qu'il reste des règles dans la queue, on prend la première règle de la queue (et on la retire donc de celle-ci), puis on calcule toutes ses paires critiques avec l'ensemble des règles du système (y compris elle-même). Ensuite, on ajoute toutes les nouvelles règles, obtenues à partir des paires critiques, au système de réécriture et à la queue. Une règle est obtenue à partir d'une paire critique (w_1, w_2) en normalisant w_1 et w_2 , puis en orientant la règle $w_1 \rightarrow w_2$ en fonction de l'ordre de réduction donné pour la complétion.