Ecole Normale Supérieur de Lyon

Rapport final du projet Latexifier

Encadrant : Eddy Caron

Projet Latexifier

Timothée Pecatte

Antoine Mottet

Membres du projet :

Raphaëlle	Crubillé
François	Dross
Stéphane	Durand
Adrien	Durier
Thomas	Grégoire
Benjamin	Hadjibeyli
William	Lochet
Antoine	Mottet
Clément	Moutet
Timothée	Pécatte
Matthieu	Rosenfeld
Mathieu	Schmitt

Table des matières

1 Présentation

1.1 Objectifs

Le but du projet Latexifier est de fournir un logiciel capable d'extraire un document LaTeX à partir d'un document PDF. Plus précisément, il s'agit de fournir un code LaTeX correspondant à un fichier PDF ayant été généré avec le programme `pdflatex`.

Le logiciel génère un code utilisable et structuré, éventuellement différent du code original, notamment du point de vue de la mise en page. Ce point clé permet de simplifier le problème puisqu'il ne s'agit pas d'effectuer une *décompilation Latex*, mais seulement de fournir un code LaTeX équivalent sémantiquement.

L'architecture du logiciel est prévue pour que celui-ci soit facile à maintenir à jour, *via* le système de modules. On pourra donc envisager d'implémenter des évolutions, sans avoir à changer de manière drastique la façon dont l'application fonctionne.

Latexifier est un logiciel libre, distribué sous licence GPL.

1.2 Public visé

Le public visé est assez large puisqu'il s'agit de toutes les personnes qui ont à faire à LaTeX. En effet, il est assez fréquent, pour un individu λ, d'avoir envie de récupérer le contenu d'un PDF afin de le modifier, le ré-arranger, le corriger, etc. Et bien souvent, il est très désagréable de devoir re-définir les sections à la main, la bibliographie ou encore les références.

En bref, il y a plusieurs choses qu'il faudrait automatiser et c'est précisément ce que notre logiciel a comme objectif.

Les premiers contacts avec la communauté LaTeX ont été positifs. Les forumeurs s'accordent à dire que le projet est intéressant, et nous ont présenté leurs besoins premiers qui coïncident avec les modules principaux.

1.3 Produits concurrents

Pour l'instant, il n'existe pas de produit sur le marché capable de générer un tel code LaTeX et c'est pourquoi nous pensons que ce logiciel possède une réelle valeur ajoutée.

Par exemple, le logiciel `pdftotext` permet d'extraire le contenu du PDF d'entrée, mais la sortie est seulement du texte, on perd la structure du document. On peut par exemple citer le numéro de page qui apparaîtra puisque contenu du PDF, mais un humain comprend que ce n'est pas significatif, et même ajouté à la compilation donc pas nécessaire.

Cependant, il existe quelques logiciels libres permettant de manipuler et d'extraire certaines informations de fichiers PDF.

Nous avons notamment choisi d'utiliser PoDoFo, qui est une bibliothèque distribuée sous licence LGPL permettant de manipuler les fichiers au format PDF, car la gestion des fichiers au format PDF (et donc la gestion de toutes les versions de ce standard qui a progressé ces dernières années) n'est pas un objectif en soi.

2 Approche technique

2.1 Problématiques

La première question que l'on peut se poser est celle de la correspondance entre le PDF et le code LaTeX : en effet, pour un même PDF, on pourrait donner une infinité de codes LaTeX le générant. Il a donc fallu faire des choix pertinents quant aux traductions, nous avons donc décidé de prendre la solution qui nous semblait la plus *propre*. De même, compte tenu du temps imparti au projet, des priorités ont dues être sélectionnées, afin de se concentrer sur une partie du problème réalisable dans le temps imparti. Les principaux critères ont été le degré de rébarbativité pour l'usager et la possibilité d'automatisation.

La première problématique technique a été de trouver où et comment extraire les informations du document PDF. Il a donc fallu, dans un premier temps, lire des passages de la spécification de Adobe pour un document PDF afin de comprendre comment il été structuré. Grâce à l'outil *PoDoFo*, nous avons pu parser le document PDF et extraire un équivalent en terme de structure C++. Ce document PDF peut ensuite être directement traité par nos différents modules.

Enfin, pour de nombreux packages traitant d'un aspect particulier du code LaTeX , il a été nécessaire de faire un compromis entre des méthodes générales mais difficiles et chronophages, et des heuristiques plus simples gérant la majorité des cas. Pour certains cas, le problème associé est même indécidable et il a donc fallu faire des suppositions sur le document PDF (par exemple, chaque référence est précédée d'un mot clé).

2.2 Architecture logicielle

Notre approche consiste à utiliser la spécification du format PDF pour extraire les informations pertinentes, afin de séparer ce qui est généré automatiquement par le compilateur LaTeX de ce qui sera présent dans le document final. Le projet a été développé en C++.

Tout d'abord nous avons développé ce que nous appelons le *Core*, qui a pour tâche d'ouvrir un document PDF, de le déchiffrer afin d'en extraire différentes méta-données (polices utilisées dans le document, taille du document, ...) qui serviront aux autres modules par la suite.

Dans un second temps, ce contenu est donné au module appelé *Parser* qui a pour tâche de transformer le flot de code PDF en bouts de texte avec informations structurées, qui permettent aux éléments supérieurs de notre architecture de générer du code LaTeX.

Dans un troisième temps, plus conséquent, un module *Manager* gère différents sous-modules capables d'identifier et de latexifier un type de bloc précis. Il a la tâche d'appeler les différents sous-modules, et de fusionner les différents codes de retour.

```
/R52 7.97011 Tf
1 0 0 1 519.48 207.72 Tm
[(n)-3.41853]TJ
/R30 11.9552 Tf
10.0801 -4.31992 Td
[(t)-1.42541(o)-2.83448]TJ
-457.56 -24 Td
[(t)-1.4249(h)0.975119(e)-307.769(n)0.975119(o)-2.83755(n)0.974098(z)3.3879(e)3.3879(r)-1.42439(o)-324.024(c)3.3879(o)-32.9498(o)-2.83755(r)-1.42439
(d)0.975119(in)0.974098(a)-2.83652(t)-1.42439(e)3.3879(s)-306.582(o)-2.83755(f)-2.12586]TJ
/R36 11.9552 Tf
138.48 0 Td
[(x)3.93724]TJ
/R54 7.97011 Tf
6.60039 4.44023 Td
[(□□)4.03117]TJ
/R30 11.9552 Tf
4.8 -4.44023 Td
[(,)-311.176(a)-2.83857(n)0.974098(d)0.97614]TJ
/R36 11.9552 Tf
29.5203 0 Td
[(A)1.26204]TJ
/R54 7.97011 Tf
8.75977 -1.8 Td
[(N)-4.26398]TJ
/R38 7.97011 Tf
8.16016 0 Td
[(\()6.48174]TJ
/R52 7.97011 Tf
3.23984 0 Td
[(B)3.01725]TJ
/R38 7.97011 Tf
6.71992 0 Td
[(\))6.48174]TJ
/R30 11.9552 Tf
3.83984 1.8 Td
[(,)-321.207(a)-2.83857]TJ
/R36 11.9552 Tf
16.5602 0 Td
[(m)2.81202]TJ
/R32 11.9552 Tf
12.6 0 Td
[(□□)4.11286]TJ
```

FIGURE 1 – Contenu d'un PDF...

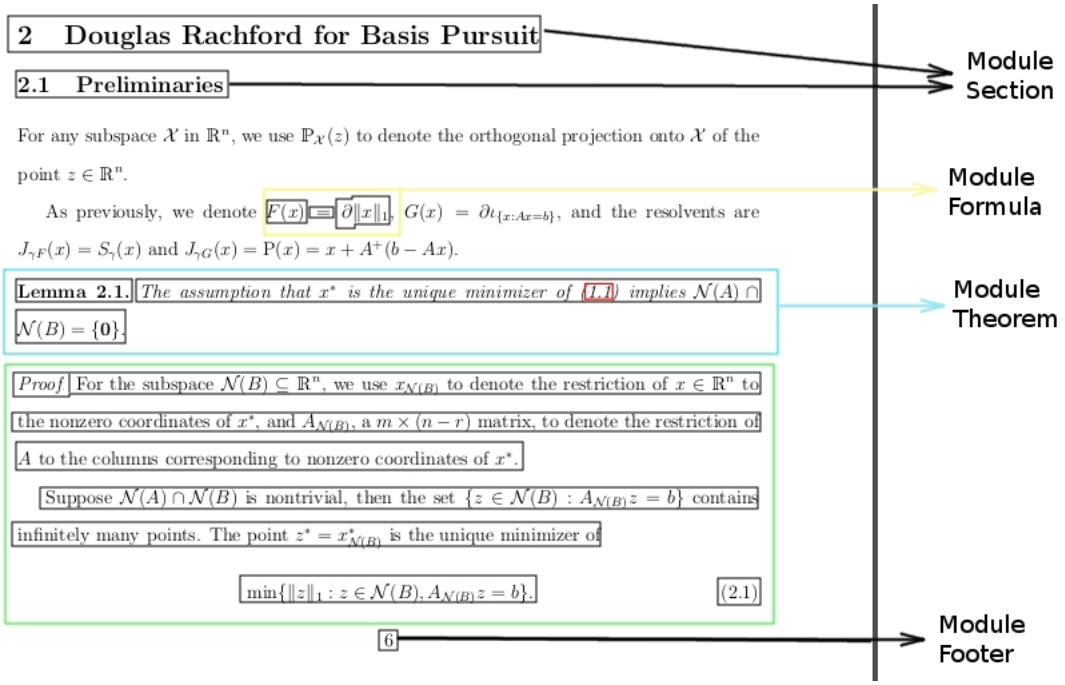

FIGURE 2 – Illustration du fonctionnement général

L'architecture de l'application est résumée dans la figure 3.

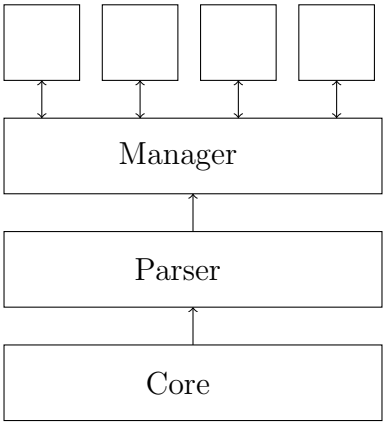

FIGURE 3 – Architecture du logiciel

L'intérêt principal de notre architecture est sa forte modularité. En effet, les diverses couches logicielles sont indépendantes entre elles (ce qui permet de modifier un niveau sans que les autres aient besoin d'être modifié), et les modules sont totalement indépendants (ce qui permet d'en ajouter et d'en supprimer sans problème).

2.3 Dépendances

Le logiciel est dépendant de `PoDoFo`, comme nous l'avons décrit précédemment. Nous avons choisi ce parser de PDF notamment car il est assez stable pour être dans les packages officiels de plusieurs distributions, et aussi car la communauté est assez réactive et preneuse de remarque. Il est également dépendant de `Qt` et de `Cmake` : `Cmake` est nécessaire à la compilation et `Qt` est utilisée pour l'interface graphique du logiciel.

3 Organisation

3.1 Répartition des groupes de travail

Le projet a été divisé en plusieurs groupes de travail. Chaque groupe de travail possède un responsable de travail (en gras dans la liste) et chaque groupe est supervisé par l'un des deux (ou les deux) chef de projet afin d'avoir à tout moment une vision globale de l'avancement du projet. Les groupes étaient les suivants :

- *Core* : François Dross, Thomas Grégoire, **Timothée Pécatte**
- *Parser* : Raphaëlle Crubillé, Antoine Mottet, **Mathieu Schmitt**
- *Manager* : **Thomas Grégoire**
- Module *sections* : William Lochet, **Clément Moutet**, Matthieu Rosenfeld
- Module *listes* : François Dross, **Stéphane Durand**
- Module *formules* : Adrien Durier, Thomas Grégoire, Benjamin Hadjibeyli, **Antoine Mottet**
- Module *images* : **Mathieu Schmitt**
- Module *URLs* : Antoine Mottet, **Matthieu Rosenfeld**
- Module *théorèmes* : **Raphaëlle Crubillé**, Benjamin Hadjibeyli, Antoine Mottet
- Module *bibliographie* : **William Lochet**, Timothée Pécatte
- Module *tableaux* : Thomas Grégoire, **Clément Moutet**
- Module *références* : **François Dross**
- Communication : Adrien Durier, **Benjamin Hadjibeyli**
- Site web : **Adrien Durier**
- Interface : François Dross, **Matthieu Rosenfeld**.

3.2 Calendrier

Un calendrier d'avancement des groupes de travail a été défini. Il y a eu deux types de groupes de travail : les modules, qui se terminent avec un rendu fonctionnel satisfaisant les objectifs pré-établis, et les groupes de long terme, qui impliquent un rendu fonctionnel rapide, mais pouvant nécessiter des évolutions.

Parmi les groupes de long terme, le *Core* et le *Parser* ont été fonctionnels très rapidement. L'avancement rapide des premiers groupes de travail a permis de lancer à la mi-novembre un certain nombre de nouveaux modules.

Le calendrier d'avancement des modules est donné dans la figure 4.

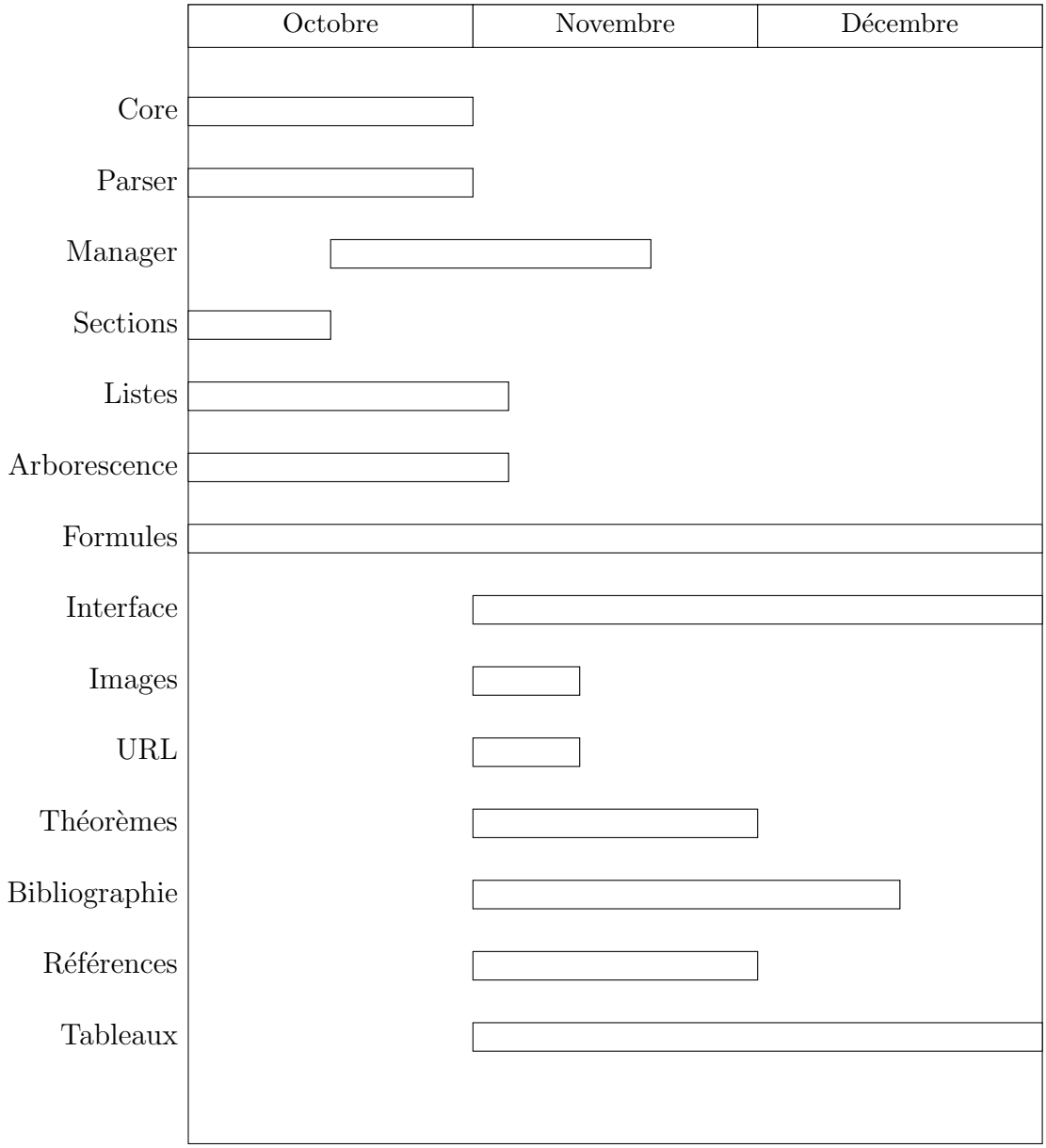

FIGURE 4 – Développement de Latexifier

4 État des lieux

Un logiciel fonctionnel a été développé. Il est téléchargeable sur le site web du projet, et une interface a été créée, permettant de configurer quelques options du logiciel. Les divers groupes de travail ont rencontré plus ou moins de succès. Nous allons donc décrire leur avancée :

4.1 *Core*

Objectifs

Le *Core* effectue un pré-traitement du PDF afin d'extraire certaines informations nécessaires pour les couches supérieures.

Une partie majeure du travail est d'extraire les polices du document PDF, afin de pouvoir associer à un caractère dans le code PDF, un caractère (ou plusieurs pour les ligatures) correspondant à ce que Adobe afficherait à l'écran. Ce travail n'est pas trivial car la spécification PDF permet de *mapper* n'importe quel caractère sur n'importe quel caractère. Par exemple, le symbole racine carré est associé à la lettre p dans une certaine police et donc le *Core* doit récupérer les informations suffisantes pour que le *Parser* puisse extraire le *vrai* contenu du document PDF.

Le reste du travail est principalement l'extraction d'autres informations comme les images, les liens URL, ...Toutes ces informations sont stockées puis transmises aux couches supérieures qui pourront s'en servir. C'est une des rares dépendances verticales dans notre logiciel : si un module a besoin d'informations externes (au texte), il faudra rajouter un petit bout de code dans cette partie du *Core* qui extraira les informations dont il a besoin. La propagation sera automatiquement mis en place.

Difficultés rencontrées

La difficulté principale rencontrée est un problème indécidable pour un certain type de police (type 3) : quel est le caractère correspondant à une image ? En effet, pour certaines polices, il n'y a aucune description textuelle de l'association et les seules informations disponibles sont les images des caractères. Il faut alors extraire un caractère d'une image, ce qui est hautement non-trivial. Heureusement, pour la plupart des cas, les caractères correspondent (l'association est l'identité), mais un des problèmes récurrents est les ligatures : le caractère *fi* est affiché en une seule image et il faut donc réussir à remarquer qu'il s'agit en fait des deux caractères f et i.

Avancée

Pour la majorité des polices, le *Core* arrive à retrouver l'association et permet donc d'extraire le contenu. Cependant, dans certains cas, certains caractères n'arrivent pas à être associé, ce qui a pour conséquence d'avoir des caractères non-ascii en sortie du logiciel (pour les ligatures notamment).

Perspectives

Une technique qui pourrait permettre de contourner le problème serait d'utiliser un OCR (Optical Character Recognization) qui utilise des réseaux de neurones pour détecter le caractère correspondant à l'image. Il faudrait donc mettre en place une interface qui permettrait de faire apprendre le réseau grâce à l'utilisateur qui pourrait corriger les erreurs du réseau.

4.2 *Parser*

Objectifs

Le *Parser* découpe le contenu du fichier PDF en blocs cohérents, qui sont ensuite décodés par les divers modules qui sont ajoutés au logiciel.

Difficultés rencontrées

Le principal enjeu de ce groupe de travail était de prendre en compte les dizaines de commandes de dessin décrites par la spécification PDF afin de les traduire et de les transformer en un langage intermédiaire que nous avons spécifié au début du projet.

Les opérations de très bas niveau, comme traduire les codes utilisés dans le document PDF en leur équivalent graphique (comme par exemple le "p" est transformé en $\surd$ quand il le faut), ainsi que la "simple" tâche de regrouper des groupes de lettres afin de les traduire en un seul mot, sont effectuées à cette étape-là du processus. Ces opérations non triviales sont paramétrées par les dictionnaires de symboles et le fichier de configuration, afin de pouvoir s'adapter à différents documents.

Avancée

Tout d'abord, le *Parser* a pris en compte les commandes d'affichage et de disposition du texte de la spécification PDF, afin de pouvoir rapidement lancer les modules des couches supérieures. Progressivement, le *Parser* a pris en charge les primitives de dessin (courbes de Bézier, segments, rectangles, ...) ainsi que d'affichage d'images.

Perspectives

Tout comme le *Core*, le *Parser* est considéré comme terminé et ne devrait plus être modifié.

4.3 *Manager*

Objectifs

Le but principal du *Manager* est de répartir les bouts de code PDF entre les modules de sorte que tout le document puisse être reconnu. Il doit donc notamment gérer les conflits

(lorsque plusieurs modules peuvent reconnaître le bout de code) pour éviter d'aboutir à des bouts de code non-reconnaissable.

Pour cela, nous avons mis en place un DAG de dépendances qui peut être modifié de manière externe. Ce fichier permet de définir un système de priorité et d'intrication optimisée.

Avancée

Le *Manager* est considéré comme achevé jusqu'à nouvel ordre.

4.4 *Sections*

Objectifs

Le module *sections* repère dans les blocs envoyés par le *Parser* ceux qui représentent un titre (caractérisés souvent par une police en gras, et une taille plus grande).

Difficultés rencontrées

Il est délicat de déterminer quelle partie du texte correspond à un titre ou sous-titre, car il existe de nombreuses présentations différentes pour les PDF. La principale difficulté a donc été de trouver des critères de reconnaissance suffisamment pertinents pour traiter la plupart des cas possibles (en tout cas, ceux sur lesquels le module a pu être testé).

Avancée

Notre algorithme nous donne la taille qu'il pense correspondre aux `section`, `subsection` et `subsubsection`.

A noter que la taille des `subsubsection` est la même que celle du texte.

Dans un premier temps, nous extrayons des informations correspondant à toutes les tailles présentes dans le PDF. Le module va également se servir de statistiques récoltées dans le `manager/statistics` pour les critères secondaires.

Puis nous cherchons la taille la plus utilisée, et en déduisons que c'est la taille principale du PDF, donc la taille des `subsubsection`. Ensuite, nous associons les plus grands tailles aux plus grands titres, donc la plus grande taille à `section`, et la deuxième à `subsection`, modulo le fait qu'elles répondent à quatre critères, configurables :

- Critère `ratioSectionChars` : Il faut qu'il n'y ait pas plus de caractères dans une `section` ou `subsection` qu'un certain ratio des caractères ne correspondant pas à un titre (ce ratio est le paramètre `ratioSectionChars`, à 4 par défaut),
- Critère `avgWordLength` : Il faut que la taille moyenne d'une catégorie de titre soit supérieure à une certaine valeur (paramètre `avgWordLength`, à 6 par défaut),
- Critère `sumLeading` : Il faut que l'espacement moyen entre les titres et le reste du texte soit supérieur à l'espacement moyen du texte normal.

– Critère `totalNumberLines` : Il faut que le nombre de lignes occupées par une catégorie de titre soit supérieure à une certaine fraction du nombre de lignes total du PDF (paramètre `minimumNumberOfLinesForASection`, à 0,001 par défaut).

Perspectives

Il faut continuer de tester le module sur d'autres PDF, afin d'affiner les paramètres, ou d'en créer de nouveaux. On peut éventuellement penser à détecter si le PDF a une bonne structure, et dans le cas contraire, suggérer à l'utilisateur de modifier certains paramètres.

4.5 *Listes*

Objectifs

Ce module a pour but de reconnaître les différents types de liste (`enumerate, itemize, description`) dans le document.

Difficultés rencontrées

La première idée, qui était de reconnaître les listes d'après leur marge, n'a pas pu être utilisée car cette marge varie énormément d'un document à un autre. Finalement, d'autres critères sont utilisés pour la reconnaissance des `enumerate` et `itemize`, mais pour les description, celles-ci étant a peu près équivalente à un début de ligne en gras, nous ne les reconnaissons pas.

Avancée

Le module *listes* traite les cas les plus courants, c'est-à-dire que les blocs provenant d'une énumération (y compris complexe) et d'un `itemize` sont reconnus et réécrits de manière adéquate.

4.6 *Formules*

Objectifs

La gestion des formules est bien sûr l'un des intérêts principaux de LaTeX. Ce groupe de travail avait pour but de gérer tout ce qui est inclus dans l'environnement mathématique. Cependant, on a rencontré de nombreuses difficultés, et le résultat n'est que partiellement satisfaisant.

Difficultés rencontrées

Les difficultés rencontrées ont été nombreuses. Nous présentons ici les quatre principales.

La reconnaissance de l'environnement formules : même si, pour un humain, il est généralement facile de déterminer s'il lit une formule ou pas, ce n'est pas aussi simple informatiquement

parlant. En effet, la représentation dans le code PDF est complexe. Il n'y a pas de claire frontière entre les polices mathématiques et les autres. On ne peut donc pas, simplement à partir de la police, comme le ferait un humain, déterminer si l'on est dans l'environnement mathématique. Par exemple, les nombres ou les signes + ou = sont dans des polices normales. Dés lors, une expression aussi simple que $a + 1 = 3$ n'a pas lieu d'être traduite dans cet environnement. On se restreint donc aux symboles purement mathématiques, c'est-à-dire qui nécessitent forcément un environnement mathématique pour être compilés.

La conversion entre code Adobe et LaTeX : ces codes sont si différents qu'il n'y a pas de moyen exhaustif d'effectuer la conversion. Tous les cas sont possibles : deux codes en bijections, plusieurs codes LaTeX pour un code Adobe, un code LaTeX correspondant à une concaténation de codes Adobe... Un exemple pathologique est la représentation de la racine : dans le code Adobe, une racine est un "p" dans une police exotique, avec un trait horizontal ajusté à la longueur de l'expression. Un passage via l'Unicode a été tenté, mais là encore, les conversion ne sont pas évidentes. On a donc réalisé des choix dans nos traductions, ce qui, dans la majorité des cas, n'atteint pas la cohérence du contenu.

Les codes LaTeX à arguments : certains symboles, comme la racine carrée $\sqrt{3}$, ont en LaTeX des arguments (ici 3). Cela peut amener de la récursivité ($\sqrt{\sqrt{\sqrt{3}}}$...). Alors, la reconnaissance du début et de la fin de l'argument devient difficile, et il peut y avoir imbrication avec d'autres modules.

La gestion des positions : pour un certain nombre de fonctionnalités de l'environnement mathématique, la gestion des positions est nécessaire. C'est le cas par exemple des indices et des exposants, extrêmement utilisés en LaTeX et qui ne sont reconnaissables qu'à leur emplacement.

Avancée

Le module est fonctionnel et compatible dans le projet. Les fonctionnalités de base, comme la traduction via les dictionnaires et la gestion des exposants et des indices, sont en place. Cependant, les dictionnaire de traduction sont encore incomplets : ils ne contiennent que les symboles LaTeX classiques, (environ 250 entrées). Les fonctionnalités plus avancées comme les symboles à arguments (racines, fraction...) sont encore à l'état expérimental.

Perspectives

Toutes les fonctionnalités du module ont été implémentées dans l'idée de pouvoir s'étendre sans engendrer de problèmes à d'autres commandes de positionnement à arguments (comme les indices, fractions...), y compris lorsque de telles commandes sont encastrées. Ainsi, nous avons bon espoir d'intégrer une gestion efficiente de certaines de ces commandes à court terme, en particulier les fractions, les indices de sommes, les flèches étiquetées. De plus, nous espérons que grâce à la communauté, les dictionnaires de traduction seront rapidement complétés, en tout cas en ce qui concerne les symboles les plus critiques. Ce module est donc appelé à beaucoup évoluer ; et si la gestion des formules est difficile, elle ne constitue pas non plus un obstacle à la traduction de documents PDF vers LaTeX .

4.7 *Images*

Objectifs

Le module *images* traite les images du PDF ajoutées avec la commande `includegraphics`.

Difficultés rencontrées

Les images sont stockées dans un "dictionnaire" d'images dans le document PDF. Le problème est que ces images sont compressées. On aurait pu faire le choix d'ajouter une dépendance à un logiciel de compression afin de les décompresser, ou de simplement recopier les images en format PDF (car une image pdf peut être ajouter a l'intérieur d'un PDF).

Avancée

Le module n'est pas encore fonctionnel. Nous avons une fonction qui récupère les images sous leur forme compressée, mais elles ne sont pas encore traitées. La bibliothèque PoDoFo gère depuis les dernières versions la décompression des images, que nous allons donc utiliser pour finaliser ce module.

4.8 *URLs*

Objectifs

Le module *URLs* avait pour but de détecter les liens dans le PDF et d'utiliser la commande `href` pour produire ces liens.

Difficultés rencontrées

Dans le PDF, il n'y a pas d'information indiquant qu'un bout de texte est une URL. D'ailleurs, les blocs extraits par le parser n'étais pas découpés au niveau des limites de l'URL. En fait, le PDF fournit des rectangles qui contiennent les zones cliquables des liens. Il fallait à partir de ces rectangles identifier le contenu correspondant aux liens.

Avancée

Actuellement, le module URL fonctionne sur tout les PDFs sur lesquels il a été testé. Il serait intéressant de faire plus de tests sur d'autres PDFs.

4.9 *Théorèmes*

Objectifs

Le module *Théorèmes* avait pour but de reconnaître l'environnement correspondant, c'est-à-dire le début et la fin des énoncés, et de renvoyer la commande LaTeX nécessaire.

Il a été choisi de laisser à l'utilisateur en option le choix des mots-clés utilisés lors de la reconnaissance.

Difficultés rencontrées

Il n'existe pas de méthode générale et exhaustive pour reconnaître des théorèmes. En effet, le début d'un théorème est généralement marqué par un mot-clé, dans une police particulière. Cependant, le mot-clé peut être utilisé à un autre endroit du texte, et la police également. La fin du théorème est encore plus difficile à reconnaître, car n'est pas marquée. Elle correspond généralement à la fin d'un passage en italique, mais ce n'est pas toujours le cas, si par exemple le théorème termine par une formule. On a donc du faire des choix, qui correspondent à une utilisation réaliste de l'environnement, et ne traitant pas les cas pathologiques.

Avancée

Nous avons actuellement implémenté une heuristique, qui reconnaît des mots-clés, et place l'environnement `theorem` jusqu'à trouver un espacement différent de l'espacement habituel. Cependant, même si la plupart des théorèmes sont ainsi reconnus, cette heuristique n'est pas suffisante dans certains cas pathologiques, comme les énoncés incluant des schémas.

Perspectives

Les deux évolutions envisageables sont d'une part, d'améliorer l'heuristique utilisée afin de gérer plus de cas pathologiques, et d'autre part, de réfléchir à comment reconnaître les démonstrations, bien plus complexes à reconnaître.

4.10 *Bibliographie*

Objectifs

Le module *Bibliographie* avait pour but de reconnaître la partie bibliographie du PDF et d'en extraire un ficher .bib correspondant.

Difficultés rencontrées

Différents problèmes ont été rencontrés. Le premier concerne les nombreux champs de BibTex et la façon dont il les affiche. Si certains champs, tel que l'auteur, le titre, la page ou bien l'année, sont possibles à deviner, d'autres ne le sont pas. Si on met le nom de l'éditeur dans le champ adresse par exemple, on peut obtenir la même mise en page que si on le plaçait dans le champ `editor`. Le choix a été de se concentrer tout d'abord sur les champs accessibles, qui correspondent d'ailleurs aux informations importantes. L'autre problème vient de la gestion des différentes options de mise en page de BibTex, comme par exemple écrire J.DUPOND ou bien DUPOND, J. On a adopté un traitement au cas par cas.

Avancée

Le module n'est pas encore fonctionnel. On a réussi à identifier la partie correspondante dans le pdf ainsi qu'a extraire des blocs correspondant à chaque référence. L'extraction de certaines informations comme le titre et l'auteur fonctionnent, il manque cependant la partie permettant de nommer correctement les références ainsi que celle renvoyant le code LaTeX correspondant.

Perspectives

Outre le code manquant, on pourrait essayer de reconnaître toujours plus de champs de BibTex, ainsi que de deviner le type de référence (article, books...) auquel on a à faire.

4.11 *Tableaux*

Objectifs

Le but de ce module est de reconnaître les tableaux du PDF.

Difficultés rencontrées

Nous avons tout d'abord essayé de les reconnaître en cherchant des régularités dans les espacements entre les caractères, mais nous nous sommes vite aperçus que nous n'arriverions pas, dans le temps imparti, à réaliser un algorithme fonctionnel avec cette approche, car les espacements entre les textes peuvent être très variables, vu qu'ils dépendent de la taille et la répartition des cases, de leur alignement à gauche, à droite, ou au centre, ainsi que de leur taille. Nous avons donc opté pour ne détecter, pour l'instant, que les tableaux ayant leurs cases délimitées par des traits, ce type étant le plus utilisé.

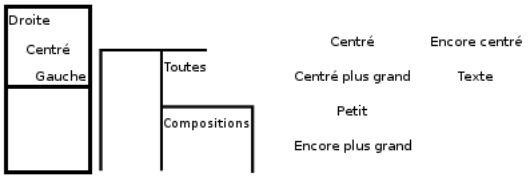

FIGURE 5 – Les différentes configurations possibles

Avancée

Notre module détecte les tableaux tels que demandés.

Pour cela, nous parcourons dans un premier temps le PDF afin d'en extraire toutes les lignes.

Puis nous déterminons quelles lignes se touchent et ainsi, nous les classons en groupes.

Nous déterminons ensuite les rectangles élémentaires de ces groupes, c'est à dire les rectangles n'étant composés d'aucun sous-rectangle, et créons à partir de là des groupes de rectangles.

Ces groupes de rectangles en contact ne sont autres que les tableaux, dont les rectangles sont les cases.

Avec ces données en main, il est aisé de savoir quel texte est dans un tableau, et dans quelle case il est.

On a donc toutes les informations nécessaires pour reconstituer ce type de tableaux.

Perspectives

Il serait intéressant de détecter d'autres types de tableaux, comme ceux n'ayant pas de traits séparateurs, ou bien ceux n'en ayant que pour les colonnes/lignes, voire bien sur étendre la détection à tous les types possiblement reconnaissables.

4.12 *Références*

Objectifs

Le but du module est de reconnaître, dans le PDF, les endroits où il y avait des commandes `label` et `ref` dans le document `.tex` originel.

Difficultés rencontrées

Les commandes label ne génèrent rien dans le PDF et les commandes `ref` sont remplacées par de simples nombres. La difficulté était alors de différentier les nombres correspondant à des références des autres, et les références entre elles (il peut y avoir plusieurs objets ayant le même numéro).

Avancée

Le module utilise des mots-clés pour repérer les références. L'idée est que pour que les lecteurs reconnaissent les différentes références, le PDF comportera généralement un mot-clé devant chaque référence (théorème, section, lemme...). Après la génération du code latex par les modules, on parcours le code pour y repérer les mots-clef et reconnaître ainsi les références. Par contre, il n'est actuellement pas utilisé.

Perspectives

Le module Référence n'est pour l'instant pas utilisé dans les différents modules qui reconnaissent des objets référençables. Ceux-ci pourront lui indiquer quels mots-clefs chercher et quels nombres peuvent leur être associés. Un objectif futur pourrait être de trouver les références à des documents annexes et de ne pas les reconnaître (par exemple si on parle du "théorème 1 de l'annexe 2", le 1 ne doit pas être reconnu par une référence vers le document 1 du document).

4.13 Communication

Objectifs

L'objectif de ce groupe de travail était de gérer les communications externes, et dans certains cas, internes. La communication externe comprenait d'une part la promotion du projet auprès des possibles usagers, mais aussi la prise de renseignement quant aux besoin de la communauté. La communication interne incluait notamment la rédaction des divers rapports.

Avancée

Dès le début, un site web a été créé afin de promouvoir le projet, et des contacts ont été pris avec les forums de la communauté LaTeX afin de s'enquérir de leurs besoins premiers et de l'intérêt suscité. Ces premiers contacts se sont avérés positifs. Cependant, la première release a été retardée car nous estimions que le projet n'était pas encore assez mûr pour une diffusion étendue.

Perspectives

Maintenant que la première release a eu lieu, l'un des prochains objectifs est la fidélisation d'une communauté de beta-testeurs. Cela passe notamment par des contacts plus développés avec l'équipe, ainsi que par l'écriture d'un guide de l'utilisateur plus étoffé.

4.14 Site web

Objectifs

La mise en place d'un site web était nécessaire pour la promotion et la diffusion de latexifier. Il a été lancé dès le début du projet, et fait partie du groupe de travail communication.

Avancée

Deux sites web existent : `latexifier.com` et `latexifier.fr`. Ils ont le même contenu, l'un en français, l'autre en anglais, avec notamment une présentation du projet, de l'équipe et de l'état de développement. Des sections d'actualités, de téléchargements et d'aide sont également présentes. De plus, des outils d'administration faciles à utiliser ont étés mis en place au cours du projet, afin que tout développeur puisse facilement modifier le contenu et la structure des sites, ainsi qu'un wiki, un forum, et des fonctionnalités supplémentaires pour la communication interne et la gestion des groupes de travail.

Perspectives

Le site implique forcément une grosse activité de maintenance post-projet. Une section d'aide plus développée est également prévue, avec guide d'utilisation et guide du développeur.

Enfin, il faudra augmenter la visibilité du projet grâce aux méthodes précédemment citées, mais aussi réfléchir à d'autres méthodes de communication.

4.15 Interface

Objectifs

Tout logiciel nécessite une interface. Nous avons choisis de la diviser en deux parties : l'une correspondant au lancement de la version par défaut de latexifier, et l'autre permettant de personnaliser les options du logiciel. En effet, l'action de certains modules dépend de paramètres que l'utilisateur peut facilement donner, comme par exemple les mot-clés qu'il utilise dans son environnement de théorèmes.

Avancée

L'interface contient deux boutons : le premier permet de lancer directement la version par défaut de latexifier sur un PDF, et le second mène aux paramètres de configurations. L'onglet ouvert pour les paramètres de configuration permet de modifier le contenu du fichier de configuration sans passer par un éditeur de texte. Cependant, l'interface n'est pas encore très stable et nécessite encore du travail.

5 Conclusion

5.1 Organisation interne

Il est ressorti clairement du projet que tout le monde n'est pas à même de coder efficacement, ce qui était bien sûr prévisible. L'équipe a ainsi était répartie de manière à ce que chaque groupe de travail contienne au moins une personne capable de coder. La part de codage dans le projet était bien sûr non négligeable, cependant les tests nécessaires et les nombreux problèmes théoriques ont permis que tout le monde y trouve sa place.

Le travail à 12 n'est pas une chose triviale à gérer. La création d'un structure hiérarchique, avec deux leaders à sa tête, s'est avérée fort utile. Leur recul par rapport à la globalité du projet assurait la qualité et l'interconnexion de chaque module.

5.2 Rendu final

Dans tout projet, on n'est jamais totalement satisfait du rendu final. Latexifier était un projet ambitieux et il était difficile d'obtenir un résultat convenable dans le temps imparti. Un certain nombre de fonctionnalités LaTeX sont traitées, comme les sections et les listes, et globalement, le logiciel fonctionne. Cependant, on aurait espéré avancer davantage sur certains sujets, comme par exemple la gestion des formules, qui n'est pas encore satisfaisante.

5.3 Utilisation du logiciel

La première release du logiciel peut être téléchargée sur notre site web `latexifier.com`. Une fois compilée avec `cmake`, on peut alors choisir de le lancer avec ou sans interface graphique.

Sans l'interface, on utilise la commande `bin/latexifier`, suivie par l'emplacement du PDF à traiter. Pour lancer l'interface graphique, on utilise la commande `bin/qlatexifier`.