

L'extension `ted`

Manuel Pégourié-Gonnard
mpg@math.jussieu.fr

v1.01 (2007/12/12)

1 Introduction

Le nom de l'extension `ted` signifie *token list editor*, soit en français « éditeur de listes de lexèmes ». Comparé à `sed` dont son nom s'inspire, il n'est pas très puissant : il ne sait faire actuellement que deux choses avec les listes de lexèmes, à savoir les afficher en détail, et y opérer des substitutions. Toutefois, sa caractéristique principale est de travailler vraiment au niveau des lexèmes, et non des simples caractères.

L'extension `ted` est capable de faire des substitutions même à l'intérieur d'une paire d'accolades, et n'a pas de problèmes avec des lexèmes particuliers. En fait, elle est conçue pour fonctionner même avec des listes comportant des lexèmes très exotiques.

2 Usage

L'extension `ted` définit essentiellement deux macros publiques : `\Substitute` et `\ShowTokens`. La première est la raison d'être de `ted`, mais au cours du développement de l'extension, la partie la plus difficile et intéressante a été d'être capable d'écrire la deuxième. J'ai donc décidé de la proposer à l'utilisateur, car je pense qu'elle peut être utile, soit pour déboguer du code, soit pour apprendre T_EX.

`\Substitute` La syntaxe de `\Substitute` est la suivante.

```
\Substitute[*][<sortie>]{<entrée>}{<de>}{<vers>}
```

Commençons par l'usage le plus simple : sans étoile ni argument optionel. Sous cette forme, `\Substitute` remplace chaque occurrence de la sous-liste `<de>` par `<vers>` dans l'`<entrée>`, et place le résultat dans le registre à lexèmes `\ted@toks`. Comme `\Substitute` est sans doute destiné à être utilisé par des auteurs de classe ou d'extensions, le `@` dans le nom de ce registre ne devrait pas poser de problèmes la plupart du temps.

Quoi qu'il en soit, vous pouvez changer facilement le nom du registre utilisé pour la sortie avec l'argument optionel en spécifiant le nom d'un registre à lexèmes dans `<sortie>`. En fait, vous voulez peut-être que la sortie soit placée dans une macro et non un registre : dans ce cas, vous pouvez spécifier par exemple `\def\macro` (ou `\long\def\macro` etc.) comme argument optionel. Dans tous les cas, `<sortie>{trucs}` doit être une syntaxe légale d'affectation de `truc` dans la sortie. Bien sûr, si vous demandez à ce que la sortie soit placée dans une macro, vous devez veiller à ce qu'elle ne contienne pas de dièses (caractère de paramètre) mal placés.

La version étoilée de `\Substitute` a pour but de vous simplifier la vie quand vous ne voulez pas spécifier explicitement une liste de lexèmes en entrée, mais plutôt utiliser

le contenu d'une macro ou d'un registre, en développant une fois son premier argument obligatoire avant de commencer à travailler dessus. Ceci vous évite d'avoir à placer vous-même des `\expandafter`, surtout dans le cas où vous souhaitez utiliser l'argument optionnel. Cette fois, l'usage est inversé par rapport à l'argument optionnel : c'est la cas d'une macro qui est le plus naturel. Pour un registre de lexèmes, vous devez écrire le `\the` vous-même : ainsi, votre *(entrée)* pourra être de la forme `\macro`, ou `\the\toksreg`, ou toute autre chose dont le 1-développement donne ce que vous voulez.

La version avec deux étoiles a aussi pour but de vous épargner la peine de contrôler vous-même le développement des arguments. Elle fonctionne comme la version étoilée simple, mais développe cette fois ses trois arguments obligatoires une fois avant de commencer à travailler. La même remarque que pour la version étoilée simple s'applique concernant les macros et les registres à lexèmes. J'espère que ces trois cas (de zéro à deux étoiles) couvriront la plupart des usages courants. Pour une gestion plus souple du développement des arguments, attendez L^AT_EX₃ !

Le comportement de `\Substitute` devrait être évident la plupart du temps. Seul un cas particulier mérite sans doute une petite remarque : dans le cas où la liste *(de)* est vide, la liste *(vers)* est insérée entre chaque paire de lexèmes de l'*(entrée)* : en particulier elle n'est pas insérée au début ni à la fin. Par exemple, `\Substitute{abc}{ }{1}` placera `a1b1c` dans `\ted@toks`.

`\ShowTokens` La syntaxe de `\ShowTokens` est la suivante.

```
\ShowTokens<*>{\liste}
```

Dans sa forme simple, `\ShowTokens` se contente d'afficher la liste de lexème, un par ligne. Pour les lexèmes de type caractère, l'affichage comporte le caractère lui-même, suivi de l'indication de son code de catégorie sous forme parlante (comme `blank space`, `letter`, etc.). Pour les séquences de contrôles et les caractères actifs, le (début du) `\meaning` est également affiché, sans dépasser les 80 colonnes de largeur afin de ne pas perturber l'affichage par ligne.

`\ShowTokensLogonly`
`\ShowTokensOnline`

Le comportement par défaut est d'afficher ces listes à la fois sur le terminal et dans le fichier log. Si vous préférez que l'affichage n'ait lieu que dans le fichier log, vous n'avez qu'à dire `\ShowTokensLogonly`. Si vous souhaitez ensuite revenir au comportement par défaut, dites seulement `\ShowTokensOnline`.

La version étoilée de `\ShowTokens` fonctionne de la même façon que dans le cas de `\Substitute` en développant son argument une fois avant d'en commencer l'analyse. La même remarque que précédemment s'applique concernant les macros et les registres.

J'aimerais conclure par la remarque suivante : au cours de la conception de `ted`, j'ai vraiment essayé de ne faire aucune hypothèse sur les lexèmes présents dans la liste. Ainsi, vous pouvez utiliser librement des accolades, des dièses, des espaces, des `\par`, des `\ifs`, des `\bgroup` ou `\egroup` dans toutes les listes de lexèmes. À ma connaissance à ce jour, la seule limitation est que les séquences de contrôle qui apparaissent ne doivent pas être (ou être `\let`-égales à) certaines des macros intimes de `ted`, commençant par `\ted@@`.

Vous savez maintenant tout ce qu'il y a à savoir sur l'utilisation de `ted`. Si vous souhaitez vous pencher sur son implémentation, il vous faudra lire les commentaires en anglais car je n'ai pas eu le courage de documenter mon code en deux langues.

C'est tout pour cette fois !
Amusez-vous bien avec L^AT_EX !