Introduction au langage TMPL

Baptiste Mélès

Avril 2008

Première partie

Introduction

Chapitre 1

Présentation générale

Le **TMPL** est un langage de programmation simulant le comportement d'une machine de Turing. Les initiales TMPL signifient Turing Machine Programming Language, et le sigle se prononce comme le mot anglais *temple*.

Dans ce document, après avoir rapidement rappelé les principes de base d'une machine de Turing, nous présenterons ce langage, puis les principales options de l'interprète `tmpl`, c'est-à-dire du logiciel qui exécute les programmes TMPL [1].

1.1 Licence

Écrit en Perl, l'interprète `tmpl` est un logiciel libre sous licence GPL (version 3 ou ultérieure). Vous pouvez donc le télécharger librement et gratuitement [2], et même modifier le code source puis redistribuer votre version personnalisée, à la seule condition que votre logiciel soit également placé sous la licence GPL.

1.2 Téléchargement

Vous pouvez télécharger l'interprète `tmpl` en vous rendant sur le site `http://www.baptiste.meles.free.fr`, puis en suivant les liens. Il vous suffit ensuite d'installer le programme `tmpl` dans un répertoire de votre choix.

Le seul prérequis, mais de taille, est d'**avoir installé Perl** sur votre ordinateur.

1. Il faut donc distinguer TMPL et `tmpl` : le premier désigne le langage, le second un interprète des programmes écrits dans ce langage. Quelqu'un pourrait tout à fait écrire un autre interprète qui s'appellerait, mettons, `contemplation`, et alors on utiliserait TMPL sans `tmpl`.

2. Mais si vous souhaitez absolument témoigner de votre enthousiasme, rien ne vous empêche d'envoyer une carte postale à Baptiste Mélès, 1 villa des Nymphéas, 75020 Paris, surtout si la carte représente des pingouins ou un beau paysage.

Si vous utilisez Linux ou un autre dérivé d'UNIX, l'endroit le plus approprié pour installer le programme est sans doute le répertoire `$HOME/bin/`, où `$HOME` désigne votre répertoire personnel ; ou bien le répertoire `/usr/bin/` si vous avez des droits d'administrateur (root) sur votre machine. Ces remarques valent vraisemblablement pour les utilisateurs de Mac (Mac OS X ou supérieur).

XXXX Et sous Windows ?

Chapitre 2

La machine de Turing

Une machine de Turing, telle qu'elle est décrite dans l'article « Théorie des nombres calculables, suivie d'une application au problème de la décision » (XXXX réfces précises), est composée des éléments suivants :

1. un ruban ;
2. une tête de lecture ;
3. une tête d'écriture ;
4. des états.

On peut effectuer exactement quatre opérations :

1. lire le symbole écrit sous la tête de lecture ;
2. écrire sur le ruban à l'emplacement actuel de la tête de lecture ;
3. modifier l'état de la machine ;
4. déplacer la tête de lecture.

Deuxième partie

Le langage TMPL

Chapitre 3

Démarrer, écrire, terminer

Un exemple simple valant tous les grands discours, voici votre premier programme en tmpl.

3.1 Votre premier programme TMPL

3.1.1 Écriture du programme

Ouvrez votre éditeur de texte favori (Emacs, Vi, Bloc-Notes de Windows, Wordpad...). Tapez la ligne suivante :

```
START: > 1 :STOP
```

Enregistrez maintenant cette ligne dans un fichier nommé `ecrit1.tmpl`, dans un répertoire quelconque de votre ordinateur. Nous ne saurions à ce propos trop vous recommander (XXXX répétition ?) de créer un répertoire nommé par exemple `tmpl-test`, dans lequel vous pourrez mener vos expérimentations, créer vos premiers programmes, etc.

3.1.2 Exécution du programme

Fermez maintenant votre éditeur de texte. Ouvrez un terminal. Exécutez votre programme en tapant la commande

```
tmpl ecrit1.tmpl
```

Si tout se passe bien [1], vous verrez simplement ceci :

```
hylas bap ~ $ tmpl ecrit1.tmpl
1
hylas bap ~ $
```

1. Dans le cas contraire, c'est sans doute que votre fichier `tmpl` n'est pas exécutable (voir `chmod`), ou qu'il ne se trouve pas dans votre `$PATH`, ou encore que vous ne vous trouvez pas dans le répertoire du fichier `ecrit1.tmpl`.

Vous avez écrit un programme, certes modeste, dont la seule fonction est d'écrire le symbole 1 sur le ruban de la machine de Turing. Commentons maintenant, pas à pas, ce premier programme.

3.2 Description du programme

3.2.1 L'état de la machine

Le programme `ecrit1.tmpl` commence par « `START:` », mot spécial du langage qui désigne l'état de départ de la machine. Un programme doit contenir *une fois et une seule*[2] l'instruction `START:`, pour la simple et bonne raison qu'il ne peut avoir qu'un seul et même point de départ.

Plus généralement, un mot suivi du signe deux-points (`:`) et placé en début de ligne désigne un **état** de la machine — pour adopter le lexique de Turing, c'est une « m-configuration ». `START:` est donc un état de la machine analogue à tous les autres, à ceci près qu'il est le premier.

3.2.2 L'écriture sur le ruban

Notre programme contient ensuite l'instruction `> 1`, qui signifie « écrire le symbole 1 »[3]. Si vous remplacez 1 par n'importe quel autre caractère, la machine l'écrira.

La principale contrainte — nous verrons les autres plus tard — est que ce caractère doit être unique. Vous pouvez écrire `> 9`, mais pas `> 10` : pour écrire 10, vous devrez écrire 1 puis 0, comme nous le verrons plus loin.

3.2.3 Le changement d'état de la machine

Enfin, la ligne de notre programme se termine par le mot `:STOP`. Un mot précédé du signe deux-points et placé en fin de ligne désigne un **changement d'état** de notre machine. En l'occurrence, `STOP:` est un mot spécial du langage TMPL qui désigne l'arrêt de la machine.

Un programme standard contient une et une seule fois `START:`, et au moins une fois l'instruction `:STOP`. Mais vous pouvez parfaitement concevoir

2. À proprement parler, nous ne dirions pas avec la même sévérité les expressions « une fois » et « une seule ». Que `START:` figure « une fois » au moins dans votre programme, c'est une nécessité absolue, à défaut de laquelle votre machine ne se mettra pas même en marche. Un programme sans `START:` équivaut donc strictement à un programme absolument vide, ou même inexistant. En revanche, vous pourriez tout à fait, si le cœur vous en dit, créer un programme comportant plusieurs fois l'instruction `START:`. Ce ne serait pas interdit, mais tout simplement vain ; car seule la première occurrence serait exécutée. Il est donc indispensable que `START:` figure « une fois » dans votre programme, et raisonnable qu'il n'y figure qu'« une seule ».

3. Ceux qui connaissent un peu UNIX reconnaîtront le symbole > qui sert à rediriger la sortie d'un programme, typiquement dans un fichier — donc, d'une certaine façon, à « écrire ».

un programme sans `START:` — un programme vide —, aussi bien qu'un programme sans `:STOP`, tel qu'une boucle infinie [4].

Vous avez donc réalisé votre premier programme en TMPL. Félicitations ! N'hésitez pas, d'ores et déjà, à mener des expériences, à écrire de petits programmes en modifiant ou en ôtant tel ou tel mot.

4. Tous les programmes dépourvus de `:STOP` sont infinis, mais la réciproque n'est pas vraie : un programme peut être infini pour la simple et bonne raison qu'il ne passe jamais par `:STOP`, quand bien même ce changement d'état figure sur l'une des lignes du code.

Chapitre 4

Se déplacer

Jusqu'ici, votre usage de la machine de Turing est assez limité : vous pouvez seulement démarrer, écrire un caractère, et terminer. En d'autres termes, vous ne pouvez manipuler qu'une seule case du ruban de la machine. Passons maintenant à des programmes un peu plus complexes, et voyons comment exécuter plusieurs instructions successives, et se déplacer sur le ruban.

4.0.4 Se déplacer d'un caractère

Enregistrez dans le fichier `123.tmpl` le programme suivant.

```
START: > 1 -> :suite
suite: > 2 -> :suite2
suite2: > 3 :STOP
```

On reconnaît ici l'état `START:` et le changement d'état `:STOP`, qui vous sont désormais familiers. Mais cette fois, on ne passe pas directement de `START:` à `:STOP` au sein de la même ligne d'instruction.

Au début, notre machine est dans l'état `START:`. Elle exécute donc la première ligne d'instruction, qui consiste à écrire le symbole 1.

Vient ensuite, toujours sur la première ligne, le symbole `->`, composé d'un tiret et du signe « plus grand que ». Il signifie « se déplacer d'une case vers la droite » ; il imite en effet une flèche tournée vers la droite. Comme vous pouvez vous en douter, pour se déplacer vers la gauche, on utilise le symbole `<-`. Dans le programme qui nous intéresse actuellement, nous nous déplaçons donc d'une case vers la droite. Enfin, nous passons à l'état `:suite`. Notre ruban présente l'état suivant :

1_

où le symbole « _ » désigne l'emplacement actuel du ruban.

Notre machine étant à l'état `suite`, elle va lire chaque ligne d'instruction pour voir s'il convient de l'exécuter. La première ligne ne correspond pas,

mais la deuxième, si ; la machine écrit donc 2 à l'emplacement actuel, puis se déplace vers la droite d'une case, et passe à l'état `suivant2`. Notre ruban ressemble maintenant à ceci :

12_

Enfin, on exécute la troisième ligne du programme, qui écrit 3 sur le ruban, et la machine s'arrête. Le ruban porte donc la suite de caractères

123

.

Chapitre 5

Ergonomie du TMPL

Si les machines de Turing étaient faites pour être simples d'utilisation, cela se saurait. Aussi tout programme en TMPL, pour peu qu'il dépasse la dizaine de lignes, deviendrait vite illisible.

5.1 Ordre des lignes d'instruction

5.1.1 Un ordre arbitraire

L'ordre des lignes d'instruction dans un programme est, dans une large mesure, arbitraire. On peut donc presque toujours permuter sans dommage les lignes d'un programme. Le programme

```
START: > 1 -> :suite
suite: > 2 -> :suite2
suite2: > 3 :STOP
```

est donc strictement équivalent au programme

```
START: > 1 -> :suite
suite2: > 3 :STOP
suite: > 2 -> :suite2
```

qui est à son tour identique au programme

```
suite: > 2 -> :suite2
suite2: > 3 :STOP
START: > 1 -> :suite
```

et au programme

```
suite2: > 3 :STOP
suite: > 2 -> :suite2
START: > 1 -> :suite
```

Vous pouvez donc écrire votre programme dans l'ordre que vous voulez. Mais nous vous recommandons tout de même de suivre, dans la mesure du possible, un ordre un peu logique, et ce pour deux raisons. La première est simplement psychologique : votre code sera bien plus lisible, partant plus simple à déboguer. La seconde raison est en revanche bel et bien technique : en suivant un ordre logique, vous éviterez les surprises — entendez par là : les bogues — liées au caractère glouton des lignes d'instruction. Précisons ce dernier point.

5.1.2 Des lignes gloutonnes

Il existe un seul cas de figure où l'ordre des lignes ait une importance : celui où une ligne est « engloutie » par une autre. Prenons le programme

```
START: > 1 -> :suite
suite: > 3 :STOP
suite: > 7 :STOP
```

Ce programme écrit 1, avance d'une case, et fait passer la machine à l'état `suite` (première ligne). Ensuite, la première ligne qui corresponde à cet état est la deuxième ; c'est donc elle qui est exécutée. Le ruban, lorsque la machine s'arrête, porte donc le nombre 13. Si en revanche on permute l'ordre des deux dernières lignes de ce programme, alors notre ruban portera le nombre 17.

Les deux lignes sont en effet concurrentes, et la première dans l'ordre du programme « engloutit » l'autre. C'est le seul cas dans lequel l'ordre des lignes d'un programme ait un effet. Mais si aucun couple de lignes de votre programme ne crée de concurrence, alors l'ordre est rigoureusement arbitraire.

Remarque : l'analyse des lignes reprend toujours depuis le début du programme. XXXX

5.2 Espaces

Vous pouvez insérer autant d'espaces et de tabulations que vous le souhaitez au début, au sein ou à la fin d'une ligne. Le langage les ignore purement et simplement. Non seulement ce n'est pas interdit, mais nous ne saurions trop vous le recomander, tant un programme aéré gagne en lisibilité. Qu'il vous suffise de comparer, même si pour l'instant vous ne comprenez pas leur signification, le programme

```
START:<>1->:suite
suite:>2->:suite2
suite2:>3:STOP
```

avec le programme suivant, qui produit rigoureusement le même résultat :

```
START:      <      > 1      ->     :suite
suite:      > 2             ->     :suite2
suite2:     > 3                    :STOP
```

Comme vous pouvez le constater par vous-même, la structure du second apparaît bien plus nettement que celle du premier.

5.3 Sauts de ligne

Autant les espaces sont tolérés et encouragés, autant les sauts de ligne sont interdits à l'intérieur d'une instruction. Le programme

```
START:      < 0       > 1
->          :STOP
```

vous renverra tout simplement l'erreur suivante :

```
hylas bap ~ $ tmpl saut_ligne.tmpl
Erreur ("saut_ligne.tmpl":1) : Format de ligne invalide.
```

5.4 Commentaires

Commençons tout de suite par une instruction qui ne se trouve pas dans la machine telle que Turing l'a décrite : les commentaires. Afin de rendre vos programmes plus ilsibles, vous pouvez en effet insérer des remarques qui ne seront ni lues, ni exécutées. Pour cela, votre ligne doit commencer par le caractère # (dièse). Exemple :

```
# Ceci est un commentaire ; il ne sera pas lu par notre machine de
# Turing.
```

Une ligne, absolument facultative, a un caractère particulier : le shebang.
XXXXX
XXXXX
Le langage TMPL contient une et une seule instruction pour chacune de ces actions, chaque instruction devant occuper une et une seule ligne. Une ligne typique d'instructions en TMPL présente l'aspect suivant :

```
etat:       < 0       > 2      ->3       :suite
```

5.5 État de la machine

Chaque ligne doit commencer par la mention d'un état de la machine, qui est un nom quelconque, contenant un nombre quelconque de caractères alphanumériques, et dont la fin est marquée par le caractère deux-points. Exemples :

```
START:
a:
1729:
abc123:
```

Parmi les noms d'état, un seul a un statut particulier, à savoir l'état START, qui désigne l'état initial du programme.

5.6 Changement d'état de la machine

Toute ligne du programme doit également se terminer par un changement d'état, chaîne alphanumérique quelconque **commençant** par le caractère deux-points. Exemples :

```
:STOP
:abc123
:1729
:a
```

Un seul changement d'état a un statut particulier, à savoir STOP, qui indique à la machine de s'arrêter. Le plus court programme terminant est donc :

```
# Programme vide, qui se termine instantanément.
START: :STOP
```

Toute ligne du programme doit impérativement contenir un état et un changement d'état. Notez que les deux états ne sont pas nécessairement distincts : on peut donc parfaitement écrire des lignes récursives comme les suivantes :

```
# Ces lignes sont vides et constituent des boucles infinies.
abc123:  :abc123
START:   :START
```

5.7 Écrire un symbole

Pour que notre machine écrive un symbole, notre instruction doit comporter, entre l'état et le changement d'état, le signe « > » (« plus grand que ») suivi du caractère qui nous intéresse. Voici par exemple comment écrire exclusivement un 1 :

```
# Écrire 1 et terminer.
START:    >1    :STOP
```

Remarquons tout de suite que le nombre d'espaces n'est pas pris en compte par le langage TMPL. Les lignes suivantes sont donc rigoureusement équivalentes :

```
# Formulations équivalentes.
START:        >1       :STOP
START: >1 :STOP
START: > 1 :STOP
```

Il est souvent très utile d'exploiter cette possibilité ouverte par le langage, car vous vous apercevrez très rapidement que vos programmes gagneront en lisibilité. Les tabulations vous permettent en effet d'aligner verticalement toutes les instructions de même type.

5.8 Lire un caractère

Avant d'exécuter une ligne d'instructions, notre machine effectue deux vérifications :

1. que son état soit bien celui qui est indiqué au début de la ligne ;
2. qu'elle lise bien sous la tête de lecture le caractère qu'on lui indique.

Par exemple, prenons le programme suivant, qui écrit d'abord un 0, puis, s'il lit un 0 (ce qui, convenons-en, sera nécessairement le cas), écrit un 1 à la place :

```
# J'écris d'abord 0.
START:         > 0   :boucle

# Puis, si je suis dans l'état "boucle" ET que je lis 0, alors j'écris 1.
boucle: < 0  > 1   :STOP
```

On peut donc réaliser le programme infini suivant, qui remplace un 0 par 1 et vice versa :

```
START:       >0  :boucle
boucle:  <1  >0  :boucle
boucle:  <0  >1  :boucle
```

Si l'on suit le déroulement de ce programme, nous voyons qu'il exécute d'abord la ligne de l'état START, donc écrit 0. Il entre alors dans l'état « boucle », comme indiqué à la fin de la même ligne. Il va donc chercher si l'un des lignes correspondant à l'état « boucle » peut être exécutée. La deuxième ligne de notre programme ne remplit pas les conditions nécessaires, car elle ne peut être exécutée que si le programme lit le symbole 1 ; or, nous avons écrit, pour l'instant, le symbole 0. La ligne suivante, en revanche, peut être exécutée, car nous sommes bien dans l'état « boucle » et lisons bien le caractère 0. Nous pouvons donc, comme le veut la troisième ligne du programme, écrire le symbole 1. Ensuite, nous voyons que la deuxième ligne peut être exécutée, puis la troisième, puis la deuxième, etc. à l'infini. Notre machine écrira puis effacera des 0 et des 1 pour l'éternité.

< : lire un caractère non nul

non précisé (APRÈS les autres cas, sinon il est glouton et absorbe tout) : toutes les autres possibilités

5.9 Déplacer la tête

La dernière commande du langage TMPL permet de déplacer la tête de lecture et d'écriture. Pour cela, il faut taper une flèche, soit avec le signe « plus petit que » suivi d'un tiret (<-), soit avec un tiret suivi du signe « plus grand que » (->), le tout étant suivi d'un nombre indiquant de combien de cases la tête doit se déplacer dans l'une ou l'autre direction. Exemples :

```
# Ce programme va écrire les chiffres 123 en commençant par le 2.
START:  >2   ->1   :nb2
nb2:    >3   <-2   :nb3
nb3:    >1              :STOP
```

Troisième partie

L'interprète tmpl

L'interprète `tmpl` a pour principale fonction d'exécuter vos programmes TMPL, mais ne s'y limite pas, tant s'en faut. Aussi allons-nous vous présenter un panorama de ses fonctionnalités.

5.10 Version de l'interprète

L'option `-v` affiche la version de `tmpl` que vous utilisez actuellement.

```
hylas bap ~ $ tmpl -v
tmpl 1.0
hylas bap ~ $
```

L'information est d'importance, car le langage TMPL devrait recevoir une extension importante entre les versions 1.0 — dernière version en date — et 2.0.

5.11 Aide

L'option `-h` affiche l'aide de `tmpl`. C'est un pense-bête pratique pour réviser les options les plus exotiques de notre interprète.

```
hylas bap ~ $ tmpl -h
tmpl : interprète du langage TMPL (Turing Machine Programming Language).
Version 1.0

Syntaxe :
tmpl [-v|-h]
tmpl [-p|-t|-x|-d secondes|-s étapes] [prog1 prog2 ...]

Options
  -v            Version de tmpl
  -h            Cette aide
  -p            Attend que l'on appuie sur Entrée entre deux opérations
  -t            Affichage du programme sous forme de tableau
  -x            Exécution eXplicite du programme (numéro de ligne)
  -d secondes   Nombre (entier ou décimal) de secondes entre deux
                instructions
  -s étapes     Nombre (entier) d'instructions à exécuter
hylas bap ~ $
```

5.12 Tabularisation d'un programme

L'option `-t` de `tmpl`, suivie d'un ou plusieurs noms de programmes, est très préciseuse pour le débogage de programmes TMPL. Elle en affiche en

effet la structure sous forme de tableau. En ceci, l'affichage se rapproche de celle que propose Turing dans son article.

Prenons le programme suivant, conçu pour afficher la suite de nombres

0 1 0 1 0 1 0 1 0 1 0 1 ...

Les lecteurs de Turing auront vraisemblablement reconnu le premier programme proposé dans la « Théorie des nombres calculables » [1]. Nous l'enregistrons dans un fichier `nb_entiers.tmpl` (l'extension *tmpl*, que nous adoptons par convention, est absolument arbitraire : elle n'a strictement aucune signification pour notre interprète).

```
#!/home/bap/bin/tmpl

# Si je lis un 0, j'avance de 2 cases et écris 1.
START:                  :b
b:      <0      ->2     :b2
b2:         >1          :b

# Si je lis un 1, j'avance de deux cases et écris 0.
b:      <1      ->2     :b3
b3:         >0          :b

# Et si je ne lis rien du tout, alors j'écris 0.
b:      <   >0          :b
```

Un programme

à lire et à comprendre au premier coup d'œil. Nous pouvons en construire une représentation plus régulière, indépendante des partis pris stylistiques du développeur (espaces, tabulations, commentaires, sauts de lignes...), avec `tmpl -t` :

```
hylas bap ~ $ ./tmpl -t nb_entiers.tmpl
Ligne  État            Lect.  Écr.  Dépl.   Nouvel état
-------------------------------------------------------
4      START                                  b
5      b               0            2       b2
6      b2                      1             b
9      b               1            2       b3
10     b3                      0             b
13     b               " "     0             b
hylas bap ~ $
```

1. Plus précisément, il s'agit de la deuxième formulation du premier programme. XXXX réfces

tmpl -e < foo.tmpl
équivaut à
tmpl foo.tmpl

Quatrième partie

Annexes

5.13 Les programmes de Turing

5.14 Une machine universelle

5.15 Un Hello world

5.16 Messages d'erreur

5.17 Manuel technique

Section destinée aux développeurs.
La structure du programme tmpl.
Les principales structures de données, et surtout le ruban.