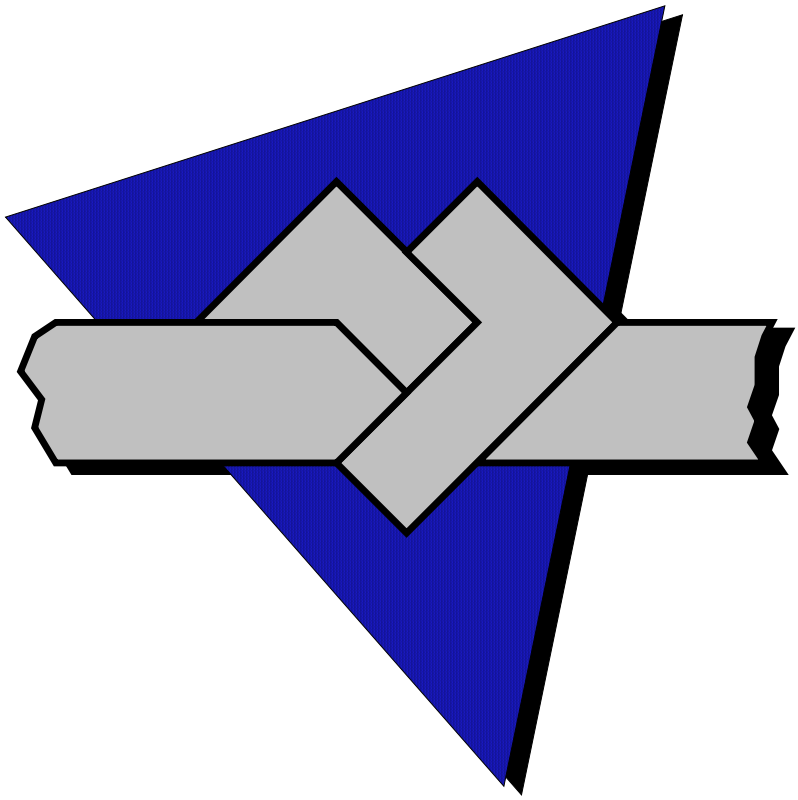
Finite State Machine Generator



Conçu et réalisé par Philippe PRADOS

Version 1.0 du 22 octobre 99.

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.>
Copyright (C) 19yy <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it under certain conditions; type `show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program `Gnomovision' (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989
Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may

consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Table des matières

1. Accueil

Merci d'avoir choisi fsmg, le générateur d'automate à états finis (Finite State Machine Generator). J'espère qu'il vous apportera satisfaction. Ce programme devrait vous permettre d'améliorer vos productions. Il facilitera l'écriture d'automates et leurs mises au point. Il introduira une rigueur de programmation sans pour autant perdre en puissance.

1.1 Introduction

Ce manuel est organisé en deux grandes parties. Dans un premier temps, une explication pas à pas du programme vous permettra de créer votre premier automate. Chaque étape importante est illustrée d'un exemple. Dans un deuxième temps, vous trouverez une explication détaillée du programme. Enfin, une annexe vous permettra de retrouver facilement les informations dont vous aurez besoin lors de l'utilisation quotidienne de fsmg. Un des chapitres de l'annexe regroupe une liste de questions/réponses classiques. Je vous recommande vivement de la consulter. Ce document suppose que vous soyez familiarisé avec le langage C et avec votre système d'exploitation. Si vous avez des remarques à faire ou des suggestions à proposer, n'hésitez pas à me contacter : `philippe@prados-obj.nom.fr`.

1.2 Avant propos

1.2.1 Droit d'utilisation

Ce programme est sous licence GNU.

1.2.2 Règle typographique

Le texte explicatif est écrit dans la police que vous lisez en ce moment. Pour les exemples de sources ou pour les affichages écran, la police `courrier` est utilisée. Le texte en *italique* indique qu'il ne s'agit pas d'une représentation littérale. Ce texte doit être remplacé par un autre, correspondant au contexte. Par exemple, la description de la commande DIR du MSDOS peut être traduite en : « `DIR` *`d:file.ext`* » ce qui se lit `DIR` suivi d'un disque, du caractère deux points et enfin d'un nom de fichier avec son extension. Le texte « *`d:file.ext`* » doit être remplacé par le nom du fichier que vous voulez. Les touches du clavier sont écrites dans cette police.

1.2.3 Conventions

- Dans ce document, l'expression « langage C », sous-entend langage C ou C++.
- L'expression « Automate » sous-entend, automate à états.

1.3 Installation

1.3.1 Liste des fichiers

Voici la liste de la livraison exécutable pour Windows.

`fsmg.exe`	Le programme lui-même
`README`	Un fichier texte expliquant les toutes dernières modifications du programme. A lire absolument !
`COPYING`	La licence du programme.
`NEWS`	L'historique des modifications.
`All.skl`	L'union de tous les squelettes.
`ArrSwitc.skl`	Le squelette ArrSwitch.
`FnArr.skl`	Le squelette FnArr.
`ArrFnArr.skl`	Le squelette ArrFnArr.
`Switch.skl`	Le squelette Switch.

`samples/`	Le répertoire contenant des exemples de sources.
`time1.fsm`	Premier exemple du pas à pas.
`time2.fsm`	Deuxième exemple du pas à pas.
`time3.fsm`	Dernier exemple du pas à pas.
`idate.fsm`	Exemple demandant une date à l'utilisateur.
`ccomment.fsm`	Exemple analysant un source C ou C++ pour en supprimer les commentaires.
`statfsm/`	
`stat.bat`	Batch d'analyse d'un automate sous MSDos/Windows.
`stat.sh`	Batch d'analyse d'un automate sous Unix.
`stat.skl`	Squelette d'analyse d'un automate.

La version « source » propose une démarche similaire aux différents utilitaires GNU. Le programme « `configure` » doit être lancé une fois pour identifier les spécificités du compilateur. Ensuite, l'appel de « `make` » compile le programme. « `make install` » l'installe sur le poste.

Avec le compilateur de Microsoft, il faut initialiser les variables d'environnements à l'aide de l'utilitaire « `vcvars32.bat` » fourni par Microsoft, puis utiliser les commandes « `configure.bat` » et « `nmake` ». L'utilitaire `sed` doit être présent dans le PATH.

1.3.2 Read.Me

Le fichier « `README` » est très important. Il contient les toutes dernières informations n'apparaissant pas dans cette documentation. Il faut absolument le consulter.

2. Pour commencer

2.1 Le principe

Le programme `fsmg` est un pré processeur. C'est-à-dire qu'il prend un source dans un format donné et le retraduit en un fichier C/C++. Ce fichier peut ensuite être compilé pour être inclus dans un programme. La syntaxe du langage est une sur-couche du langage C/C++. Certaines zones du source sont réservées à du C/C++ classique, d'autres sont spécifiques au langage. Le programme `fsmg` prend le source et l'analyse. Puis il extrait les éléments C/C++ que vous y avez inclus, y rajoute des paramètres internes, un squelette d'automate et ressort le tout dans un fichier C/C++. La syntaxe `fsmg` est spécialisée dans l'écriture d'automate à états.

Un automate est une routine classique recevant un événement ; Suivant l'état de la routine, un traitement sera exécuté. Celui-ci peut changer l'état courant pour modifier la réaction de la routine lors du prochain passage. Par exemple, une boîte de vitesses peut être assimilée à un automate. Les événements pouvant arriver sont : «Passer à la vitesse suivante » ou «Passer à la vitesse précédente». L'état est la vitesse en cours. Lorsque l'événement «Passer la vitesse suivante» arrive, la boîte au point mort, l'état devient «première» etc...

Il s'agit d'une programmation par événements. Les serveurs Internet exigent souvent un cheminement particulier de l'utilisateur. Il est possible d'atteindre certaines pages si l'on vient d'autres pages particulières. Si l'utilisateur ne respecte pas cela, une page d'erreur est retournée. Cela peut être décrit par un automate. Chaque page correspond à un état ; chaque événement, à la page d'où l'utilisateur arrive.

`fsmg` est là pour faciliter la programmation de ce type de routine et permet également de modifier l'algorithme de l'automate suivant son type. En effet, un automate est, dans le principe, un tableau à deux dimensions, une pour les états et l'autre pour les événements. A chaque intersection correspond un traitement à exécuter qui pourra éventuellement modifier l'état courant.

Si l'automate possède beaucoup d'états et/ou d'événements, le tableau peut être très gros et posséder beaucoup de combinaisons vides. Il existe différents algorithmes pour économiser soit du temps d'exécution, soit de la place

mémoire. Chacun est performant pour un type d'automate, mais pas pour un autre. `fsmg` se propose de vous faciliter l'écriture de l'automate mais surtout de choisir au mieux l'algorithme à utiliser. Lors de l'évolution de votre programme, l'algorithme pourra changer sans avoir à modifier votre source.

`fsmg` s'appuie sur des squelettes, c'est-à-dire des coquilles vides d'automate. Il est possible d'inventer d'autres algorithmes adaptés à vos besoins en modifiant le squelette et en utilisant les différentes possibilités qu'offre `fsmg` pour cela (voir page 26).

2.2 Premiers pas

Nous allons dans un premier temps examiner un petit exemple qui vous familiarisera avec la syntaxe et l'utilisation du langage. Celui-ci utilise par commodité des événements caractères. Ce n'est pas le seul usage possible de `fsmg`, mais cela permet d'être tous de suite dans le bain.

Les sources de `fsmg` possèdent l'extension «.`fsm`». Vous trouverez dans le répertoire «`samples`» de votre disque le fichier «`time1.fsm`».

Exemple 1:

```
/* ----- Section déclarative ----- */
%{

static char  buf[6];  /* Buffer de l'heure */
static short  pos=0;  /* Position courante du caractère */

%}

%%  /* ----- Section définition de l'automate ----- */

/**********/
/* $HOUR1 */
/**********/

/* Traitement du premier caractère de l'heure */

$HOUR1  : '0' '1' '2'
  { putchar(buf[pos++]=$event$);
    return($HOUR2);
  }
  | '3' '4' '5' '6' '7' '8' '9'
  { putchar(buf[pos++]=$event$);
    return($SEPAR);
  }
  ;

/**********/
/* $HOUR2 */
/**********/

/* Traitement du deuxième caractère de l'heure */

$HOUR2  : '0' '1' '2' '3' '4' '5' '6' '7' '8' '9'
  { if ((buf[0]=='2') && ($event$>'3'))
    { putch('\a');
      return($state$);
    }
    else
    { putchar(buf[pos++]=$event$);
      return($SEPAR);
    }
  }
  ;
```

```
/**********/
/* $SEPAR */
/**********/

/* Traitement du caractère de séparation */

$HOUR2,$SEPAR  : ':'
  { putchar(buf[pos++]=$event$);
    return($MIN1);
  }
  ;

/*********/
/* $MIN1 */
/*********/

/* Traitement du premier caractère des minutes */

$MIN1  : '0' '1' '2' '3' '4' '5'
  { putchar(buf[pos++]=$event$);
    return($MIN2);
  }
  | '6' '7' '8' '9'
  { putchar(buf[pos++]=$event$);
    return($RETURN);
  }
  ;

/*********/
/* $MIN2 */
/*********/

/* Traitement du deuxième caractère des minutes */

$MIN2  : '0' '1' '2' '3' '4' '5' '6' '7' '8' '9'
  { putchar(buf[pos++]=$event$);
    return($RETURN);
  }
  ;

/***********/
/* $RETURN */
/***********/

/* Traitement du caractère de validation */

$MIN2,$RETURN : '\r'
  { buf[pos]='\0';
    printf("\nTime=%s\n",buf);
    return(-1);
  }
  ;

/***********/
/* Default */
/***********/

/* Traitement par défaut */
  : /* Vide */
  { putchar('\a');   /* Emet un bip pour les   */
    return($state$); /* caractères incorrectes */
  }
  ;

%%       /* ----- Section C classique ----- */
```

```
/********/
/* main */
/********/
int main()
{
  FSM_STATETYPE etat;

  printf("Enter time (HH:MM) : ");

/* Boucle d'appel de l'automate */
  for (etat=0;etat>=0;)
  { etat=fsm(etat,getch());
  }
  return 0;
}
```

Ce fichier est un source `fsmg`, il est séparé en trois grandes sections. La première est déclarative, elle commence au début du fichier et finit en rencontrant le premier marqueur composé de deux caractères pour-cent («`%%`»). La deuxième section permet de décrire l'automate. C'est la plus importante. Enfin, la dernière, précédée du dernier «`%%`», est optionnelle et permet de se retrouver dans un contexte C ou C++ classique.

Dans la première section, il est possible de déclarer un source C ou C++ qui sera inclus au début du fichier généré. Pour cela il faut l'encadrer entre «`%{`» et «`%}`».

Entre les deux marqueurs («`%%`») se trouve la définition de l'automate. Pour chaque combinaison état/événement un traitement est déclaré.

Les différents états sont identifiés par le caractère dollars ('`$`') en début de nom.

Les événements sont des mots commençant par un caractère, des nombres ou un caractère encadré par des apostrophes ('`'`').

Les deux points permettent de séparer la zone concernant les états de celle des événements.

Après cette liste, soit vous terminez la déclaration par un point virgule, dans ce cas il n'y a pas de traitement à effectuer; soit vous indiquez entre accolades le source C à exécuter suivi d'un point virgule. La structure simplifiée de la déclaration peut être traduite en : «*<états>* `:` *<événements>* [`{` *traitement* `}`]`;`».

Vous pouvez ensuite re-déclarer d'autres combinaisons, ceci définira votre automate. Le caractère «pipe» ('`|`') permet de déclarer un nouveau traitement pour de nouveaux événements concernant le même état.

Pour tester cet exemple, vous allez tout d'abord demander à `fsmg` de générer un fichier C. Pour cela placez-vous dans le répertoire `samples`, et tapez «`fsmg time1`» suivi d'un retour chariot (`fsmg` doit être accessible). Le fichier «`time1.c`» est créé. Il faut maintenant le compiler. Utilisez votre compilateur habituel.

Le programme «`time1`» a été créé. Lancez le, en tapant «`time1`» suivi d'un Retour-Chariot.

Il attend l'entrée d'une heure. A chaque caractère incorrect, le programme émet un bip. Essayez de taper des heures incorrectes, le programme réagit dès le premier caractère impossible. Tapez par exemple «`25:00`», «`23:60`»... Pour arrêter le programme tapez Ctrl-Pause.

Les événements de ce programme sont des caractères tapés au clavier. Cet automate possède six états, un par caractère attendu. Les traitements retournent l'état correspondant à l'attente du caractère suivant.

Les états «`$HOUR1`» et «`$HOUR2`» s'occupent de l'entrée de l'heure, les états «`$MIN1`» et «`$MIN2`» s'occupent de l'entrée des minutes. L'état «`$SEPAR`» traite le caractère de séparation, et «`$RETURN`» attend le caractère Retour-Chariot.

«`$HOUR1`» attend un chiffre. Si celui-ci est compris entre zéro et deux, l'utilisateur peut vouloir en entrer un deuxième pour composer les nombres compris entre dix et vingt-trois. Ces caractères sont donc séparés des autres et

retournent l'état «$HOUR2» pour attendre éventuellement un deuxième chiffre. Pour les chiffres compris entre trois et neuf, pas de doute, l'utilisateur ne peut que rentrer le caractère de séparation. Dans ce cas, le traitement retourne l'état «$SEPAR» après avoir mémorisé l'événement.

«$HOUR2» attend le deuxième chiffre de l'heure et vérifie si le nombre ainsi composé n'est pas supérieur à vingt-trois, dans ce cas le traitement retourne l'état «$SEPAR», sinon, il émet un bip !

«$SEPAR» attend le caractère ':' avant de retourner l'état «$MIN1» pour attendre les minutes. La syntaxe déclare que le même traitement est effectué pour l'événement ':' avec les états «$SEPAR» et «$HOUR2». En effet, après le premier chiffre, si celui-ci est compris entre zéro et deux, l'utilisateur passe en état «$HOUR2». Mais si à ce moment, l'utilisateur tape le caractère de séparation, le programme doit l'accepter et retourner l'état «$MIN1», ce que fait l'événement «$SEPAR».

Les états «$MIN1» et «$MIN2» procèdent de la même manière. L'état «$VALIDE» attend le caractère retour-chariot puis retourne -1 pour signaler au programme appelant que l'automate est terminé.

Le traitement commenté `Default` déclare un traitement pour tous les événements non déclarés précédemment, ce traitement émet un bip pour tous les caractères incorrects.

`fsmg` traduit ce source en un fichier C. Ce fichier déclare un tableau d'entiers, avec en ordonnées tous les caractères déclarés dans le source, et en abscisses tous les événements. Dans le tableau se trouvent des entiers indiquant le traitement à effectuer.

Une routine «`fsm`» est créée. Elle reçoit deux paramètres, l'état en cours et l'événement. Ces deux valeurs servent à consulter le tableau, puis le traitement correspondant est exécuté.

Voici le tableau des transitions de cet exemple. En abscisses, vous avez les différents états, et en ordonnées, les différents événements possibles. Chaque case du tableau indique le nouvel état.

	$HOUR1	$HOUR2	$SEPAR	$MIN1	$MIN2	$RETURN
'0'	$HOUR2	$SEPAR	Erreur	$MIN2	$RETURN	Erreur
'1'	$HOUR2	$SEPAR	Erreur	$MIN2	$RETURN	Erreur
'2'	$HOUR2	$SEPAR	Erreur	$MIN2	$RETURN	Erreur
'3'	$SEPAR	$SEPAR	Erreur	$MIN2	$RETURN	Erreur
'4'	$SEPAR	$SEPAR	Erreur	$MIN2	$RETURN	Erreur
'5'	$SEPAR	$SEPAR	Erreur	$MIN2	$RETURN	Erreur
'6'	$SEPAR	$SEPAR	Erreur	$MIN2	$RETURN	Erreur
'7'	$SEPAR	$SEPAR	Erreur	$RETURN	$RETURN	Erreur
'8'	$SEPAR	$SEPAR	Erreur	$RETURN	$RETURN	Erreur
'9'	$SEPAR	$SEPAR	Erreur	$RETURN	$RETURN	Erreur
':'	Erreur	$MIN1	$MIN1	Erreur	Erreur	Erreur
'\n'	Erreur	Erreur	Erreur	Erreur	-1	-1

Les traitements ne sont pas représentés ici, mais il est facile de parcourir le tableau en fonction des différents événements pouvant arriver. Avec le caractère tapé par l'utilisateur et l'état de l'automate, le tableau indique le nouvel état courant. `Erreur` indique l'émission d'un bip. L'état courant ne change pas.

La création de ce tableau est fastidieuse, il n'est pas aisé de rajouter un état ou un événement car il faut pour cela vérifier toutes les transitions pour tous les états et/ou pour tous les événements. `fsmg` permet de décrire ce tableau sans se soucier de son implémentation.

`fsmg` permet de modifier facilement un automate. Par exemple, si l'on désire la possibilité de sortir de l'application à tout moment par le caractère Échappe, il suffit d'ajouter la règle :

```
: '\x1B' { return(-1); }
```

Tous les états peuvent maintenant recevoir le nouvel événement. Il n'est pas nécessaire de reconstruire le tableau, `fsmg` se charge de cela.

Le fichier «time2.fsm» du répertoire «samples» possède la version modifiée du programme. Suivez la même procédure pour le compiler, puis testez-le. Maintenant, pour sortir du programme, vous pouvez taper la touche Échappe.

Nous allons rajouter la possibilité de taper la touche Efface pour corriger un caractère incorrect. Le fichier «time3.fsm» du répertoire «samples» possède cette nouvelle version.

Pour cela il faut mémoriser les états par lesquels le programme est passé afin de pouvoir revenir aux états précédents. Nous ajoutons donc un tableau, et à chaque traitement, nous mémorisons l'état courant.

Nous informons que pour le caractère Efface et quel que soit l'état en cours, le traitement doit effacer le dernier caractère et retourner le dernier état mémorisé.

Nous déclarons également un cas particulier pour le premier état ; Si l'utilisateur tape sur le caractère Efface nous refusons de le traiter et le signalons par un bip.

Traduisez, compilez et testez cette version. Comme vous pouvez le constater, il est très facile de modifier l'automate sans pour autant remettre en cause le programme.

Un des avantages de fsmg est d'adapter le source généré suivant les caractéristiques de l'automate. fsmg peut choisir un squelette d'automate à un moment. L'ajout d'une nouvelle règle peut l'entraîner à en choisir un nouveau plus performant pour la nouvelle version, sans que vous ayez, pour cela, changé une seule ligne de votre source.

•

Nous allons maintenant voir plus en détail la syntaxe et les possibilités du programme.

Les traitements sont encadrés par des accolades et correspondent à du C/C++ presque classique. Seuls les mots clefs commençant par un dollar seront convertis.

Il est possible d'indiquer plusieurs états associés à un ou plusieurs événements. Pour cela il suffit d'indiquer la liste des états séparés par des espaces ou par une virgule.

Exemple 2:

```
%%
$DEBUT , $SUITE : CLAVIER SOURIE
                { ...
                }
                ;
```

Déclare d'exécuter le même traitement pour les combinaisons «$DEBUT:CLAVIER», «$DEBUT:SOURIE», «$SUITE:CLAVIER» et «$SUITE:SOURIE».

Pour une liste d'états il est possible de définir en une seule déclaration des traitements pour plusieurs événements. Pour cela, il faut séparer les événements par le caractère «pipe» ('|').

Exemple 3:

```
%%
$DEBUT : CLAVIER
       { ...
       }
       | SOURIE
       { ...
       }
       ;
```

déclare un traitement pour la combinaison «$DEBUT:CLAVIER» et un autre pour la combinaison «$DEBUT:SOURIE».

Il est possible de déclarer des traitements par défaut pour des événements ou pour des états. Pour cela laissez vide la liste des états et/ou des événements.

Exemple 4:

```
%%
$DEBUT :
       { ...
       }
       ;
```

Déclare un traitement par défaut pour tous les événements non traités par l'état «$DEBUT».

Exemple 5:

```
%%
  : CLAVIER
  { ...
  }
  ;
```

Déclare un traitement par défaut pour tous les événements «CLAVIER» quel que soit l'état, si et seulement si, un autre traitement n'a pas été déclaré.

Exemple 6:

```
%%
  :
  { ...
  }
  ;
```

Déclare un traitement par défaut pour toutes les combinaisons non déclarées par ailleurs. C'est un système simple pour gérer les erreurs de l'automate. Toutes les combinaisons non déclarées seront prises en charge par ce traitement.

Attention, il peut y avoir conflit entre deux traitements par défaut.

Exemple 7:

```
%%
$DEBUT :
       { ...
       }
       ;

  : HORLOGE
  { ...
  }
  ;
```

Si l'événement «HORLOGE» arrive à l'état «$DEBUT» quel traitement effectuer ? Pour résoudre ce problème, le programme choisit par défaut de privilégier l'état. Il exécutera donc le traitement associé à «$DEBUT:». Il est possible de modifier la priorité entre l'état et l'événement par un paramètre de la ligne de commande (voir page 24). Si une situation ambiguë comme celle-là se présente, un message «warning» apparaît lors de la traduction.

fsmg déclare un événement fantôme, il s'agit d'un événement correspondant à tous les événements inconnus. Pour déclarer un traitement pour cet événement utilisez les syntaxes «par défaut». Ceci est très utilise pour les émulations de terminaux ou les protocoles de communications, tous les caractères correspondant à la norme du terminal sont traités par des règles, les autres sont gérés par les règles «défaut».

Le traitement associé à chaque combinaison «*état:événement*» doit éventuellement pouvoir modifier l'état ou l'événement courant. Pour cela le programme analyse le traitement et modifie les chaînes commençant par un dollar ('$') par les valeurs associées de l'automate.

Les variables «$state$» et «$event$» sont l'état et l'événement courants. Pour modifier un état il suffit d'écrire dans le source «$event$=$NEXT;». Les chaînes de caractères ne sont pas modifiées par fsmg. Vous pouvez donc écrire "printf("$state$=%d\n", $state$);" pour afficher l'état courant (par exemple "$state$=3").

Par défaut, le programme déclare le premier état du fichier comme étant l'état zéro, l'état de démarrage. Il est possible de choisir un autre état zéro. Pour cela, dans la zone déclarative du fichier, utilisez la commande %start suivie du nom de l'état concerné.

Exemple 8:

```
%start $FIN
%%
$DEBUT :
       { ...
       }
       ;

$FIN : CLAVIER
     { ...
     }
     ;
```

Les événements peuvent être de deux types : les «internes» et les «externes».

Les internes sont tous les événements « chaînes de caractères » définis dans les déclarations. Les externes sont les nombres ou les caractères ainsi que les chaînes déclarées par «%extern» (voir page 15).

Les internes possèdent une valeur choisie par fsmg. Les externes ont une valeur choisie par le source.

Pour éviter un conflit entre ces données, fsmg commence par la valeur «256». Toutes les valeurs externes seront converties par l'automate en une valeur interne avant de servir d'index dans le tableau.

Exemple 9:

```
%%
$DEBUT : SOURIE 'a' 123
       ;
```

Déclare «SOURIE» comme interne, le caractère 'a' et la valeur 123 comme externe. «SOURIE» sera traduit en «257», 'a' en 258 et 123 en 259 (256 traduit l'événement fantôme). La valeur «$event$-256» servira d'index dans le tableau de l'automate. Si vous désirez utiliser des valeurs définies dans un «#define» comme valeur externe, il faudra l'indiquer dans la zone déclarative du fichier à l'aide du mot clef «%extern».

Exemple 10:

```
%{
#define LETTRE_A 'a'
%}
%extern LETTRE_A
%%
$DEBUT : LETTRE_A
       { ...
       }
       ;
```

Les événements externes déclarés par `%extern` peuvent être séparés par des virgules ou des espaces. Il peut y avoir autant de mot clefs «`%extern`» que vous le voulez.

Exemple 11:

```
%extern LETTRE_A, LETTRE_B
%extern CHIFFRE_0
%%
```

Attention, les événements doivent être des valeurs entières pré-calculées. Ils doivent pouvoir être placés à la droite d'un «`case`».

Exemple 12:

```
%{
#define EXT_1 errno  /* Incorrect */
#define EXT_2 10*3   /* Correct */
#define EXT_3 "test" /* Incorrect */
%}
%extern EXT_1 EXT_2 EXT_3
%%
$DEBUT : EXT_1 EXT_2 EXT_3
       ;
```

Ce type d'erreurs n'est pas détecté par `fsmg`. Elles apparaîtront lors de la compilation du fichier C.

Il est possible de modifier la valeur maximum des événements externes par un paramètre de la ligne de commande (voir page 24). Il est également possible de le faire dans le source, pour cela utilisez la commande «`%max` *`num`*» dans la zone déclarative du fichier.

Exemple 13:

```
%{
#define EXT 1
%}
%max 2
%extern EXT
%%
$DEBUT : EXT INTERNE
       ;
```

`INTERNE` prend la valeur `3` et `EXT` prend la valeur `4`.

Il est également possible dans le source de demander à `fsmg` d'utiliser les traitements par défaut concernant les états plutôt que les événements. Pour cela utilisez la commande «`%priority event`» ou «`%priority state`» dans la zone déclarative du fichier.

Exemple 14:

```
%priority event
%%
       : 'A'
       ;
$DEBUT :
       ;
$FIN   : 'A'
       ;
```

L'automate fonctionne sur le principe d'une boucle sans fin du type «`while (1)`» ou «`for (;;)`». Si vous modifiez la valeur d'un état ou d'un événement à l'aide des variables «`$event$`» et «`$state$`», l'automate exécutera la nouvelle combinaison après votre traitement.

Exemple 15:

```
%{
#define CLAVIER   1
%}
%extern CLAVIER
%%
$DEBUT : CLAVIER
       { int c=getch(); /* Premier traitement */
         if (c=='[') $state$=$SUITE;
         else return(0);
       }
       ;
$SUITE : CLAVIER
       { int c=getch(); /* Deuxième traitement */
         if (c==']') $state$=$DEBUT;
       }
%%
int main()
{
  fsm(0,CLAVIER);       /* Exécute l'automate */
  return 0;
}
```

Ce petit exemple attend un caractère au clavier. Si le caractère est un crochet ouvrant, il attend des caractères jusqu'au crochet fermant, sinon, il sort de l'automate. L'événement «`CLAVIER`» a été déclaré externe. La routine «`main`» lance l'automate en commençant avec l'état 0 (le premier état déclaré, ou bien, celui déclaré par `%start`) et l'événement «`CLAVIER`». Le premier traitement est donc exécuté. Ces lignes modifient l'état courant si le caractère est un crochet, dans ce cas, après la dernière ligne du traitement, l'automate boucle et détecte la combinaison «`$SUITE:CLAVIER`». Il exécute donc le traitement correspondant.

Un automate doit être alimenté par des événements. Il possède un état courant ce qui lui permet d'effectuer un traitement. Il est souvent utile de sortir de l'automate pour exécuter autre chose et revenir plus tard sur celui-ci. Pour cela, il est possible de garder l'état de l'automate en sortant avec la valeur courante de l'état. Par la suite le programme appelant retournera dans l'automate avec l'état précédemment retourné et le nouvel événement.

Exemple 16:

```
%%
$DEBUT : '['
       { return($SUITE);
       }
       | /* Par défaut */
       { return(-1);
       }
       ;
$SUITE : ']'
       { return($DEBUT);
       }
       | /* Par défaut */
       { return($state$);
       }
       ;
%%
int main()
{
  int etat;

  for (etat=0;etat!=-1;etat=fsm(etat,getch()));
  return 0;
}
```

Ce programme exécute exactement le même traitement que le précédant mais la lecture des caractères est centralisée dans la routine «`main`». Ceci permet d'effectuer des traitements entre chaque événement ou d'exécuter par exemple plusieurs fois le même automate en alternance.

Si l'automate détecte une combinaison « *événement:état* » inconnu, il se retrouve en erreur. Il affiche dans ce cas un message. L'automate retourne alors la valeur «`-1`». La gestion des erreurs ainsi que le traitement pointu de l'automate peuvent être modifiés à l'aide de «`#define`» (voir page 26). Pour déclarer un traitement par défaut général utilisez la syntaxe « `: { ... } ;`» qui déclare que quel que soit l'état ou l'événement non géré, il faut exécuter le traitement indiqué.

LE LANGAGE

Dans ce chapitre, la syntaxe et les fonctions du langage sont détaillés. Vous trouverez la syntaxe des fichiers sources, les paramètres de la ligne de commande, les différents squelettes par défaut ainsi que des exemples d'utilisations.

3. La syntaxe

La syntaxe générale est :

1. Syntaxe

```
[
 [%{ ... C classique ... %}]
 [%start  <state>]
 [%priority state|event]
 [%max     <value>]
 [%extern <event>[[,]<event>]*]
]*
%%
[[<state>[[,]<state>]*] : [<event>[[,]<event>]*]
                          [{ ... C convertit ...}]
                          [| [<event>[[,]<event>]*]
                             [{ ... C convertit ... }]
                          ]*
                        ;
]+
[%%
  ... C classique ...
]
```

où <state> est un état de l'automate, et <event> un événement. Les paramètres entre crochets sont optionnels.

L'étoile indique que le paramètre précédent peut être présent une infinité de fois. Le *plus* indique que le paramètre précédent peut être présent une ou plusieurs fois.

Le code C ou C++ entre accolades est le traitement exécuté si l'un des événements se présente pour ces états. Les code C ou C++ sont détectés par la présence d'accolades ouvrantes et fermantes imbriquées.

Les chaînes de caractères entre guillemets ou les caractères entre apostrophes ne sont pas pris en compte dans la détection des accolades, ainsi que les codes d'échappements précédés d'un slash inverse ('\').

Les commentaires encadrés par des «/*» et «*/» ainsi que ceux commençant par un double slash («//») et finissant par un retour chariot sont également ignorés.

Un <state> commence par le caractère dollars suivi de un ou plusieurs caractères alphanumériques, souligné compris ($[A-Za-z0-9_]+).

Un <event> commence par une lettre suivie ou non de caractères alphanumériques, souligné inclus, ou bien, est composé de chiffres, enfin, il peut être composé du caractère apostrophe, suivi d'un caractère et terminé par un autre caractère apostrophe ([A-Za-z_][A-Za-z0-9_]* | [0-9]+ | '{car}').

Le programme n'analyse pas le pré-processeur du C. Il ne faut donc pas écrire un code C pouvant porter à confusion compte tenu du pré-processeur. Le seul traitement effectué est la détection des lignes commençant par un dièse ('#') pour ne pas en tenir compte. Par exemple, un code possédant un «#define BEGIN {» afin de remplacer les accolades ne pourra pas fonctionner.

La commande «%start state» permet de déclarer l'état zéro, c'est-à-dire l'état de démarrage.

La commande «%priority» avec «event» ou «state» permet d'utiliser le traitement par défaut concernant les événements ou les états en priorité.

La commande «%max» permet d'assumer que la valeur maximum des événements externes ne dépasse pas une certaine limite.

La commande «%extern» permet de déclarer certains événements comme déclaré hors de l'automate.

Exemple 17:

```
%%
$DEBUT : CLAVIER
       { printf("Clavier=%c\n",getch());
         return($DEBUT);
       }
       ;
```

Ce programme sera traduit en un fichier C et déclarera la fonction «fsm» qui reçoit deux paramètres : l'état en cours et l'événement. Cette fonction exécutera le code suivant ces deux conditions et retournera le prochain état de la fonction. Une utilisation classique de cette fonction est celle-ci :

```
int main()
{
  int etat=0;

  while (etat!=-1)
  { if (kbhit()) etat=fsm(etat,CLAVIER);
  }
  return 0;
}
```

Voici un nouvel exemple :

Exemple 18:

```
%{
#include <stdio.h>
#define  CLAVIER 1
%}
%extern  CLAVIER
%%
$DEBUT      : CLAVIER
      { printf("Clavier=%c\n",
          getch());
        return($DEBUT);
      }
      ;
%%
int main()
{ int etat=0;

  while(etat!=-1)
  { if (kbhit()) etat=fsm(etat,CLAVIER);
  }
  return 0;
}
```

Le texte des lignes entre «%{» et «%}» est recopié tel quel dans le fichier C. A partir du premier «%%» le programme définit les états possibles ainsi que les événements qui s'y rattachent. Un morceau de code C ou C++ y est éventuellement associé. Après le deuxième «%%» le source est de nouveau recopié. Le fichier généré possède le squelette de l'automate en y modifiant certains mots clefs. Cela permet d'inclure les «#define» de la ligne de commande, les différents codes encadrés par «%{» et «%}», certains «#define» déduits du source, le tableau de l'automate etc...

Voici les deux fichiers en parallèles.

Source Fsm	Fichier C

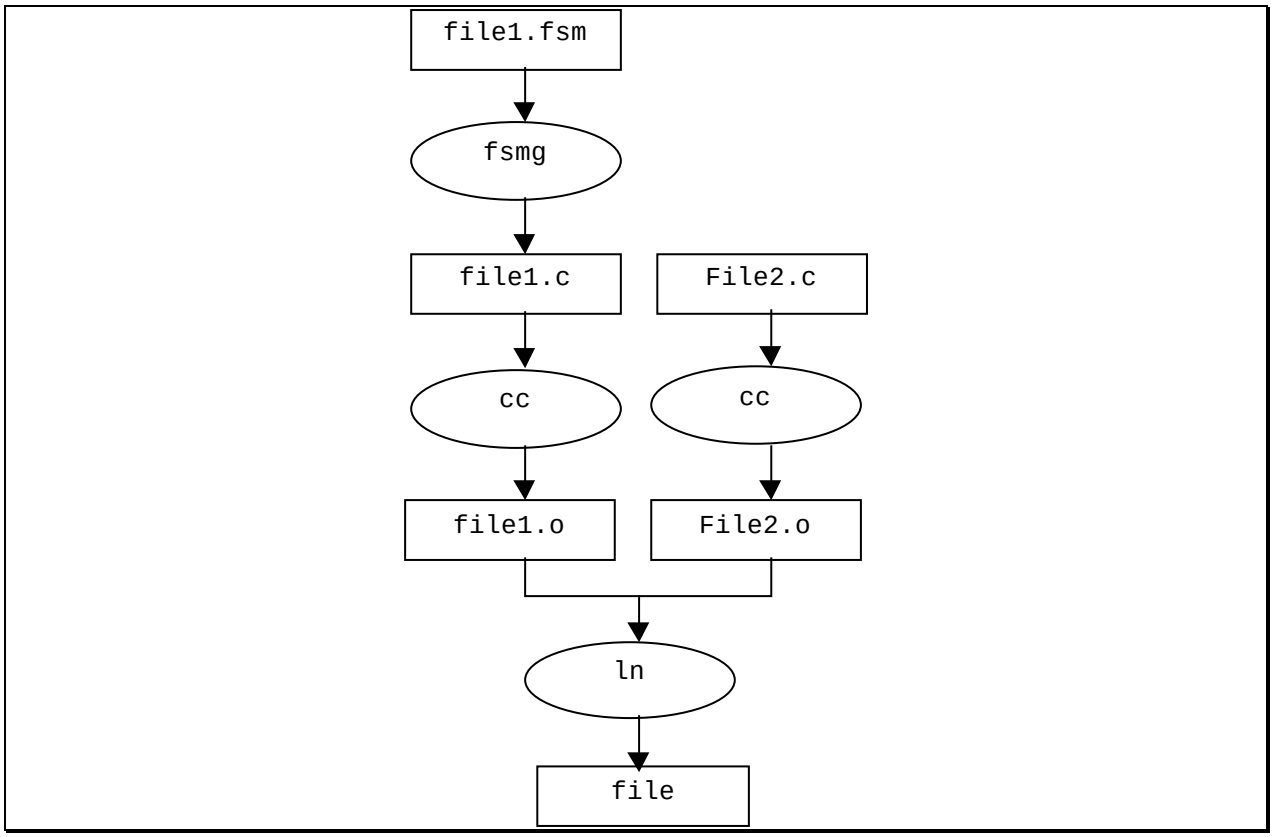
%{
#include <stdio.h>
#define CLAVIER 1
%}
%extern CLAVIER
%%
 	
$DEBUT : CLAVIER
{ printf("Clavier=%c\n",
getch());
return($DEBUT);
}
;
 	
%%
int main()
{ int etat=0;
 	
while(etat!=-1)
{ if (kbhit())
etat=fsm(etat,CLAVIER);
}
return 0;
}
	Squelette
	#define de la ligne de commande
	
	#include <stdio.h>
	#define CLAVIER 1
	
	#define pour le squelette
	#define des événements
	tableau de l'automate
	conversions
	switch
	{ printf("Clavier=%c\n",
	getch());
	return(0); Conversion
	}
	
	Suite du squelette
	
	int main()
	{ int etat=0;
	
	while(etat!=-1)
	{ if (kbhit())
	etat=fsm(etat,CLAVIER);
	}
	return 0;
	}
	fin squelette

3.1 Conseils et style d'écriture.

Il est conseillé de déclarer sur une ligne la règle et de déclarer le traitement sur d'autres. Le point-virgule de fin d'une règle doit être déclaré sur une autre ligne et si possible aligné avec le caractère deux-points de la règle. Les états et les événements doivent être en majuscules pour ne pas les confondre avec des variables. Les exemples de ce document respectent ce style.

3.2 Ligne de commandes

`fsmg` est un pré processeur, c'est-à-dire qu'il génère un fichier C/C++. Il faut donc ensuite le compiler avec votre compilateur habituel. Voici le schéma classique de développement.



Le programme se lance par «`fsmg`» suivi de paramètres et terminé par un Retour-chariot. Le seul paramètre obligatoire est le nom du fichier source.

```
fsmg test.fsm
```

Affiche la conversion du fichier «`test.fsm`».

Les paramètres du programme sont précédés du caractère *moins*. Plusieurs paramètres peuvent être indiqués en même temps. Les paramètres demandant un complément d'informations doivent dans ce cas être placés en dernier.

Pour modifier le nom du fichier de sortie, il suffit de l'indiquer sur la ligne de commande.

```
fsmg -otst.c test.fsm
```

Convertit le fichier «`test.fsm`» en «`tst.c`».

```
fsmg –otmp test*.fsm file
```

Convertie les fichiers «`test*.fsm`» et «`file.fsm`» en «`tmp/test*.c`» et «`tmp/file.c`»

L'extension par défaut du fichier source est «`.fsm`». Le nom par défaut du fichier de sortie est le même que le nom du fichier d'entrée mais avec une autre extension.

- Si l'extension en entrée est «`.fsm`», l'extension par défaut du fichier de sortie est «`.c`»,
- si l'extension est «`.fpp`», l'extension en sortie est, par défaut «`.cpp`»,
- et si l'extension est «`.fxx`», «`.cxx`» est utilisé.

Pour diriger le fichier de sortie vers l'écran, n'utilisez pas de paramètre `–o` ou indiquez comme nom le signe *moins*.

```
fsmg –o- test.fsm
```

Affiche à l'écran la traduction du fichier «`test.fsm`». Cette option est utile pour les «pipes».

Pour supprimer les «`#line`» du fichier résultat, ajoutez le paramètre «`-l`» (lettre l) ou «`--no-lines`».

Pour supprimer les «warnings» du programme, dus entre autres aux confusions possibles entre les traitements par défaut des états et des événements (voir page 14) utilisez le paramètre «-w» ou «--no-warning» .

```
fsmg -lw test
```

Convertit le fichier «test.fsm» en supprimant les «#line» et en n'affichant pas de «warning».

Pour forcer la priorité des événements en cas de confusion, utilisez le paramètre «-e» ou «--priority-event».

Le paramètre «-x» ou «--max-external» suivi d'un chiffre indique la valeur maximale des événements extérieurs. Par défaut, cette valeur est à 256.

Le paramètre «-s» ou «--size» demande à fsmg de privilégier la taille du programme généré tandis que «-t» ou «--time» privilégie la vitesse d'exécution.

```
fsmg -tx10 test
```

Convertit le fichier «test.fsm» en assumant que la valeur maximale des paramètres externes est inférieure à 10. Demande également de favoriser la vitesse d'exécution du code généré.

Pour choisir un des squelettes de base fournis avec le programme, utilisez le paramètre «-m» ou «-skelette» suivit d'un chiffre représentant le squelette à utiliser.

```
fsmg -s -m4 test.fpp
```

Convertit le fichier «test.fpp» en utilisant le squelette «ArrFnArr» et en lui demandant d'optimiser la taille du code généré. Les différents squelettes disponibles par sont :

Code	Nom	
0	All	L'ensemble des squelettes possibles (Par défaut).
1	ArrSwitch	Tableau d'entiers suivi d'un switch.
2	FnArr	Tableau de fonctions.
3	ArrFnArr	Tableau d'entiers suivi d'un tableau de fonctions.
4	Switch	Cascade de switch.

Vous pouvez également utiliser le nom du squelette précédé d'un double moin.

```
fsmg --FnArr test.fpp
```

Pour indiquer un nouveau squelette, utilisez le paramètre «-k» ou «-skelette-file» suivi d'un nom de fichier. Celui-ci doit bien sur être compatible avec la syntaxe des squelettes (voir page 26). L'extension par défaut est «.skl».

```
fsmg -kmy test
```

Convertit le fichier «test.fsm» en utilisant le squelette «my.skl».

Pour ressortir les événements internes créés par fsmg dans un fichier, utilisez le paramètre «-p» ou «--header» suivi du nom du fichier. Par défaut, l'extension utilisée est «.hfs».

```
fsmg -ptest test
```

Crée le fichier «test.hfs» avec la liste de #define définissant les événements internes.

Le paramètre «-c» ou « --check-only » demande à fsmg de ne pas créer de fichier en sortie. Ceci permet par exemple de ne créer que le fichier «.hfs» en le combinant avec le paramètre «-p».

```
fsmg -cptest test
```

Crée le fichier «test.hfs» uniquement.

Pour afficher le nom de chaque fichier traité, utilisez le paramètre «-v» ou «--verbose».

Le paramètre «`-D`» ou «`--define`» permet de rajouter des «`#define`» au début du fichier généré. Le texte indiqué après le paramètre sera recopié dans le fichier C. Le premier caractère égal est converti en espace.

```
fsmg -DFSM_DEBUG -DFSM_TRACE=3 test
```

Convertit le fichier «`test.fsm`» en déclarant «`#define FSM_DEBUG`» et «`#define FSM_TRACE 3`». Cet exemple a pour effet de générer un source gardant une trace des trois derniers événements, et vérifiant la validité de chaque événement.

Le paramètre «`-r`» ou «`--error-file`» permet d'indiquer le fichier ou seront écrit tous les messages d'erreurs. C'est très utiles pour les systèmes ne permettant pas de redirection de *stderr*.

```
fsmg -rerror.txt test
```

Convertit le fichier «`test.fsm`» en écrivant les messages `error` et `warning` dans le fichier «`error.txt`».

Le paramètre «`-h`» ou «`--help`» permet d'avoir un rapide résumé sur l'utilisation du programme.

```
fsmg -h
```

Affiche les paramètres du programme.

En résumé, voici la liste des paramètres disponibles :

2. Ligne de commande

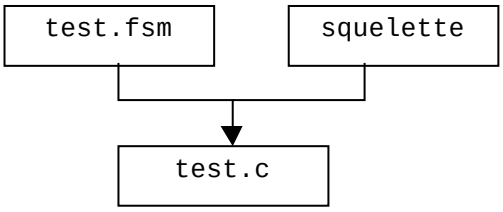
`-c` `--check-only`	Teste uniquement la syntaxe du source sans créer de fichier C.
`-D<def>` `-define <def>`	Rajoute un `#define` au source.
`-e` `--priority-event`	Priorité pour les événements.
`-h` `--help`	Affiche l'aide du programme.
`-k<file>` `--skelette-file <file>`	Indique le fichier squelette à utiliser.
`-l` `--no-lines`	Supprime les `#line`s.

`-m<0..4>`
`--skelette <0..4>`
`--All`
`--ArrSwitch`
`--FnArr`
`--ArrFnArr`
`--Switch`
	Sélectionne un squelette standard.
	0: All
	1: ArrSwitch
	2: FnArr
	3: ArrFnArr
	4: Switch

`-o<file>` `--output <file>`	Écrit les messages d'erreur dans le fichier indiqué.
`-p<file>` `--header <file>`	Indique le nom d'un fichier «`.hfs`» pour la liste des événements internes.
`-r<file>` `--error-file <file>`	Indique le nom d'un fichier pour y inscrire les erreurs.
`-s` `--size`	Optimise la taille du programme.
`-t` `--time`	Optimise la vitesse d'exécution.
`-v` `--verbose`	Affiche le nom de tous les fichiers traités.
`-w` `--no-warning`	Supprime l'affichage des warnings.
`-x<num>` `--max-external <num>`	Déclare une valeur maximum des événements extérieurs.

3.3 Les squelettes

Un squelette est un fichier «.skl» qui possède la structure générale d'un automate. Il s'agit d'un source C/C++ dont certaines parties inconnues sont remplacées par des mots clefs. Ces mots seront traduits par le programme fsmg afin d'adapter le source aux spécificités de l'automate à générer. fsmg traduit le source «.fsm» en un fichier C après l'avoir analysé.



Il recopie le squelette en modifiant les mots clefs pour inclure les lignes encadrées par «%{» et «%}», rajoute quelques «#define», etc... Le squelette est donc très important. Un paramètre de la ligne de commande (voir page 24) permet de choisir un autre squelette que le squelette par défaut. Il vous est possible de créer un automate personnalisé. Les automates fournis avec le programme cherchent à être le plus facilement adaptables à différents contextes sans avoir à recréer un nouveau squelette. Cela n'en facilite pas la lecture mais offre des possibilités intéressantes. Tous les squelettes de base possèdent une structure générale permettant d'adapter le programme généré. Pour modifier certains éléments de l'automate, il suffit d'inclure dans l'entête du fichier source, entre «%{» et «%}» certains «#define».

Voici les «#define» et leurs fonctions que vous pouvez modifier :

3. #define des squelettes

#define du squelette		Fonction
FSM_MAIN	fsm	Nom de l'automate
FSM_STATETYPE	short	Type des données « *State* »
FSM_EVENTTYPE	short	Type des données « *Event* »
FSM_FNTYPE		Type de la fonction de l'automate (cdecl, pascal,…)
FSM_BEFORE	/*Before*/	Action à exécuter avant le traitement d'un événement de l'automate
FSM_AFTER	/*After*/	Action à exécuter après le traitement d'un événement de l'automate s'il n'y a pas de «return».
FSM_PERROR(num)	fprintf(...	Traitement à exécuter en cas d'erreur. «num» est un entier identifiant l'erreur.
FSM_DEBUG		Si présent, vérifie avant chaque traitement de l'automate si les valeurs fournies sont cohérentes.
FSM_TRACE	n	Si présent avec FSM_DEBUG, indique le nombre de couples « événement/état » à garder pour trace en mémoire.
FSM_PRINT_TRACE		Affiche la liste courante des traces mémorisées. Valide uniquement si FSM_TRACE est déclaré.

fsmg rajoute avant le squelette certain «#define». En voici la liste ainsi que leurs rôles.

#define	Fonction
FSM_NAME	Nom du fichier source.
FSM_MAJOR	Numéro de version majeur de fsmg
FSM_MINOR	Numéro de version mineur de fsmg
FSM_NBEVENT	Nombre d'événements de l'automate.
FSM_NBSTATE	Nombre d'états de l'automate.
FSM_NBTRANS	Nombre de translations de l'automate.
FSM_MAXEXTERN	Valeur maximum des événements externes.
FSM_SIZE	Si présent, demande d'optimiser la taille du code généré.
FSM_TIME	Si présent, demande d'optimiser la vitesse du code.
FSM_USEERROR	Est présent si fsmg détecte une erreur possible dans l'automate.
FSM_USENOTHING	Est présent si fsmg détecte un traitement vide.

Les squelettes déclarent une fonction qui reçoit en paramètre un état et un événement. La fonction boucle avec un «while(fsm_state>=0)», et si FSM_DEBUG et FSM_TRACE sont déclarés, recopie dans un tableau l'état et l'événement courant. Ce tableau sert à avoir une trace des passages dans l'automate. Ensuite, le programme teste éventuellement les paramètres fournis, puis exécute le traitement déclaré dans le fichier «.fsm».

Voici un extrait des squelettes. Il s'agit de la boucle principale.

```
/************/
/* FSM_AUTO */
/************/

FSM_STATETYPE FSM_FNTYPE FSM_AUTO(FSM_STATETYPE fsm_state,
                                  FSM_EVENTTYPE fsm_event)
{ while (fsm_state!=-1)
  {
#if defined(FSM_DEBUG) && defined(FSM_TRACE)
/* Copie l'état et l'événement courant dans un tableau */
  ...
#endif
#ifdef FSM_DEBUG
/* Vérifie les valeurs de fsm_state et fsm_event */
  ...
#endif
    FSM_BEFORE;
/* - Traitement de l'automate - */
   ...
/* - Fin du traitement de l'automate - */
    FSM_AFTER
  }
  return(-1);
}
```

Attention, il ne faut pas utiliser les variables fsm_event et fsm_state dans les traitements. En effet, si l'automate utilise des appels de fonction, $state$ est converti en *fsm_state, sinon il est converti en fsm_state, de même pour fsm_event.

Voici la liste des mots clefs présents dans certains squelettes et traduits par fsmg.

4. Conversion des squelettes

Mots clefs	Traduction
`$define$`	*Liste des* `#define` *de la ligne de commande.*
`$end$`	*Copie la fin du fichier.*

`$exec$`	`case n:`
	Copie le source...
	`break;`
	`case n:`...

`$function$`	`FSM_DCLFN(0)`
	`FSM_DCLFN(1)`
	`FSM_DCLFN(`...

`$header$`	*Copie les sources encadrés par* `%{` *et* `%}`.

`$levent$`	`FSM_LEVENT(event1,0)`
	`FSM_LEVENT(event2,1)`
	`FSM_LEVENT(`...

`$lfunction$`	`FSM_FN(0)`
	`FSM_FN(1)`
	`FSM_FN(`...

`$local_event$`	*Copie la liste des événements locaux.*

`$lstate$`	`FSM_LSTATE(state1,0)`
	`FSM_LSTATE(state2,1)`
	`FSM_LSTATE(`...

`$switch$`	`switch(fsm_event)`
	`{ case` *n* `:`
	`switch(fsm_state)`
	`{ case` *m* `:`
	`ex_1:`
	`Copie le source...`
	`}`
	`break;`
	`...`
	`}`
	`...`

`$array$`	`FSM_ARR(0)`
	`FSM_ARR(1)`
	`FSM_ARR(`...

`$translate$`	`FSM_TRANSLATE(event1,0)`
	`FSM_TRANSLATE(event2,1)`
	`FSM_TRANSLATE(`...

`$var$`	*Liste des variables du programme.*

Il est donc facile d'adapter certains «`#define`» du squelette pour modifier l'algorithme de l'automate. Le mieux est de regarder le source de chaque squelette pour comprendre leurs fonctionnements et leurs particularités.

Suivant les squelettes, la boucle principale commence par convertir l'événement. Elle utilise pour cela la macro `FSM_TRANSLATE`. L'événement fantôme est converti en *max* (256 par défaut, ou la valeur du paramètre `-x`), les autres événements sont convertis en valeur supérieure.

Dans les paragraphes suivants, vous trouverez les différents squelettes fournis par défaut avec le programme. Chacun est expliqué dans son principe. Les avantages et les inconvénients de chaque algorithme sont discutés afin de vous aider à choisir l'automate correspondant le mieux à vos besoins. L'automate «`All`» fait le choix pour vous. Il est recommandé pour mieux comprendre les différentes versions de regarder un fichier C généré et éventuellement d'en ressortir la version *préprocesseur* (`cpp`, `gcc -E` ou `cl /EP ...` pour Microsoft).

Dans les squelettes, vous trouverez des commandes `#line`s. Elles servent à re-synchroniser le compteur de lignes du compilateur lors d'un `#if` ayant échoué. Si vous rajoutez ou supprimez des lignes aux squelettes, n'oubliez pas de modifier les `#line`s qui en résultent afin de pouvoir suivre correctement le source lors du déverminage.

3.3.1 All

3.3.1.1 Introduction

Ce squelette est le squelette par défaut. Il est la concaténation des différents autres squelettes. Un calcul est effectué au départ pour estimer la taille et la vitesse des différentes versions, puis le meilleur choix est fait.

3.3.1.2 Avantage

- Permet de choisir le meilleur algorithme suivant les caractéristiques de l'automate.

3.3.1.3 Inconvénients

- Les inconvénients de l'algorithme choisi.

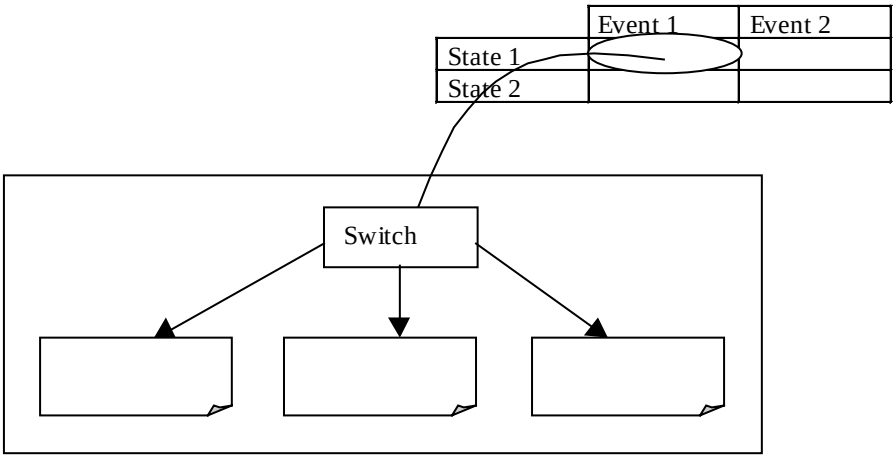
3.3.1.4 Source

Le source de ce squelette s'appelle «`All.skl`».

3.3.2 ArrSwitch

3.3.2.1 Introduction

Ce type d'automate est composé d'un tableau d'entiers à deux entrées, une pour l'état et l'autre pour l'événement. La valeur à l'intersection du tableau est ventilée vers les traitements à exécuter par un `switch`.



3.3.2.2 Avantages

- Les traitements étant dans un `switch`; il y a peu de codes supplémentaires rajoutés.

3.3.2.3 Inconvénients

- Il demande une conversion des événements extérieurs.
- Le fichier généré possède une fonction qui peut être très grande et empêcher les optimisations globales du fichier ou la compilation (suivant les compilateurs).
- Si l'automate est en grande partie vide, il y a perte de place mémoire.

3.3.2.4 Exemple

```
switch(tableau[evenement][etat])
{ case 1 :
    /* Traitement un */
      ...
    break;
  ...
```

```
  default:
    /* Erreur */
    ...
    break;
}
```

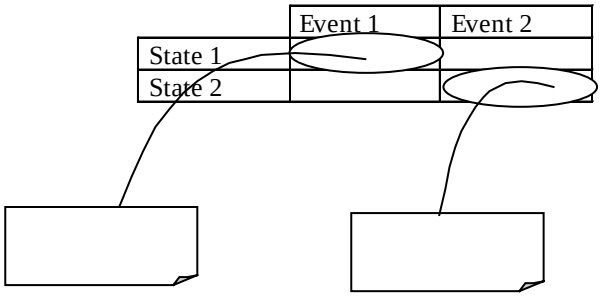
3.3.2.5 Source

Le source de ce squelette s'appelle «`ArrSwitc.skl`».

3.3.3 FnArr

3.3.3.1 Introduction

Ce type d'automate est composé d'un tableau de fonctions. Ce tableau possède deux entrées, l'état et l'événement. A l'intersection se trouve un pointeur sur la fonction à exécuter.



3.3.3.2 Avantages

- Si la taille d'un pointeur de fonction est similaire à taille d'un entier, il ne prend pas plus de place qu'un tableau d'entiers.
- Très rapide.

3.3.3.3 Inconvénients

- Demande une conversion des événements extérieurs.
- Les traitements sont définis dans des fonctions, ce qui rajoute du code.
- Si l'automate est en grande partie vide, il perd de la place mémoire.

3.3.3.4 Exemple

```
short fn_0(short* etat,short* evenement)
{ /* Traitement un */
  ...
}

short fsm(short etat,short evenement)
{ ...
  tableau[etat][evenement](&etat,&evenement);
  ...
}
```

3.3.3.5 Source

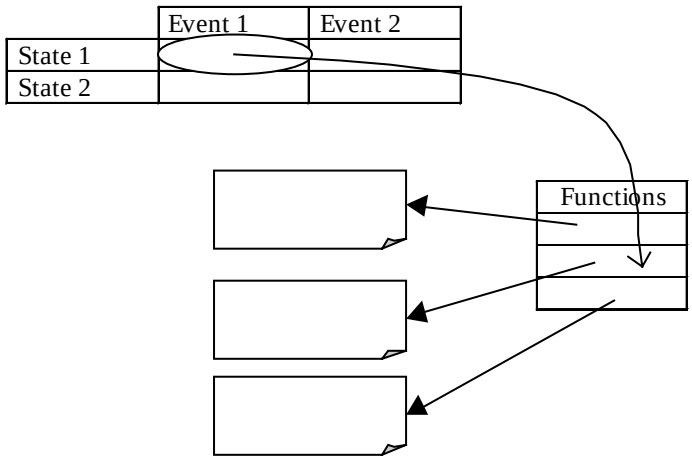
Le source de ce squelette s'appelle «`FnArr.skl`».

3.3.4 ArrFnArr

3.3.4.1 Introduction

Ce type d'automate est composé d'un tableau d'entiers possédant deux entrées : en abscisses les états et en ordonnées les événements. A l'intersection de l'état et de l'événement se trouve un entier identifiant la

routine à exécuter. Cet entier est utilisé comme index dans un deuxième tableau composé de la liste des fonctions possibles. Ce deuxième tableau est un tableau de fonctions.



3.3.4.2 Avantages

- Cet algorithme permet d'économiser de la place mémoire si le tableau de l'automate est grand pour peu de fonctions différentes. Si la taille d'un pointeur de fonction est supérieur à la taille d'un `int` ou d'un `char`, cela permet de réduire la taille de l'automate.
- Assez rapide.

3.3.4.3 Inconvénients

- Demande une conversion des événements extérieurs.
- Les traitements sont définis dans des fonctions, ce qui ajoute du code.
- Si l'automate est en grande partie vide, il perd de la place mémoire.
- Si l'automate possède beaucoup de traitements différents, le deuxième tableau peut être aussi grand que le premier, ce qui double ou triple la taille de l'automate.

3.3.4.4 Exemple

```
short fn_0(short* etat,short* evenement)
{ /* Traitement un */
  ...
}

short fsm(short etat,short evenement)
{ ...
  fonction[tableau[etat][evenement]](&etat,&evenement);
  ...
}
```

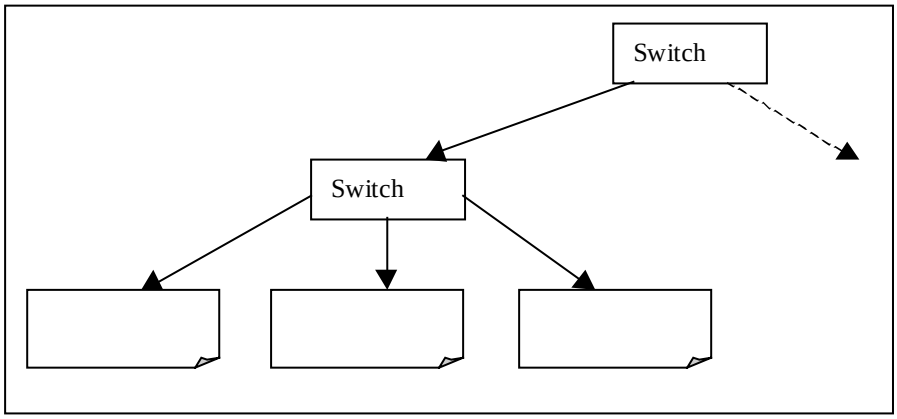
3.3.4.5 Source

Le source de ce squelette s'appelle «`ArrFnArr.skl`».

3.3.5 Switch

3.3.5.1 Introduction

Ce type d'automate est composé de `switch` en cascade menant aux traitements. Un premier `switch` ventile les états, et pour chaque état un `switch` ventile les événements.



3.3.5.2 Avantages

- Il n'y a pas besoin de convertir les événements extérieurs.
- Les optimisations possibles suivant l'ordonnancement des valeurs sont faites par le compilateur.
- Si l'automate est en grande partie vide, il n'y a pas de place perdue.

3.3.5.3 Inconvénients

- Le fichier généré possède une fonction qui peut être très grande et empêcher les optimisations globales du fichier ou la compilation.
- L'algorithme peut être moins rapide que d'autres suivant les cas.

3.3.5.4 Exemple

```
short fsm(short etat,short evenement)
{
  ...
  switch(etat)
  { case 0 :
      switch(evenement)
      { case 256 :
          fn_1:
          /* Traitement un */
          ...
          goto suite;
        case '[' :
          fn_2:
          /* Traitement deux */
          ...
        ...
      }
      break;
    ...
  }
}
```

3.3.5.5 Source

Le source de ce squelette se nomme «`Switc.skl`».

4. Comment déverminer

Plusieurs solutions s'offrent à vous. Vous pouvez déclarer un `#define FSM_DEBUG` pour que l'automate teste la validité des paramètres fournis (état ou événement corrects ?). Si vous rajoutez également un `#define FSM_TRACE` suivi d'un chiffre, l'automate mémorise les `FSM_TRACE` dernières combinaison

« *état:événements* » reçus. La macro `FSM_PRINT_TRACE` permet d'afficher la trace des derniers événements reçus. Il est donc facile de rajouter une règle affichant la trace si elle est exécutée et savoir ainsi les événements précédents.

Autrement, vous pouvez utiliser les débuggeurs du commerce pour tracer votre automate ou placer des points d'arrêt dans les traitements. `fsmg` tente dans la mesure du possible de « *cacher* » au niveau de la trace les lignes de code qu'il rajoute.

Attention, les squelettes internes ne peuvent pas être tracés car il n'existe pas de fichier Ascii les possédants, par contre, si vous utilisez un squelette externe (*paramètre* `-k`), `fsmg` déclare correctement les changements de fichier entre le source et le squelette. Vous pouvez donc les utiliser pour regarder dans le détail l'exécution de l'automate.

Enfin, si vous utilisez le paramètre «`-l`» (*lettre l)*, `fsmg` ne déclare pas de commande `#line`, vous pouvez donc à loisir parcourir le fichier C/C++ généré.

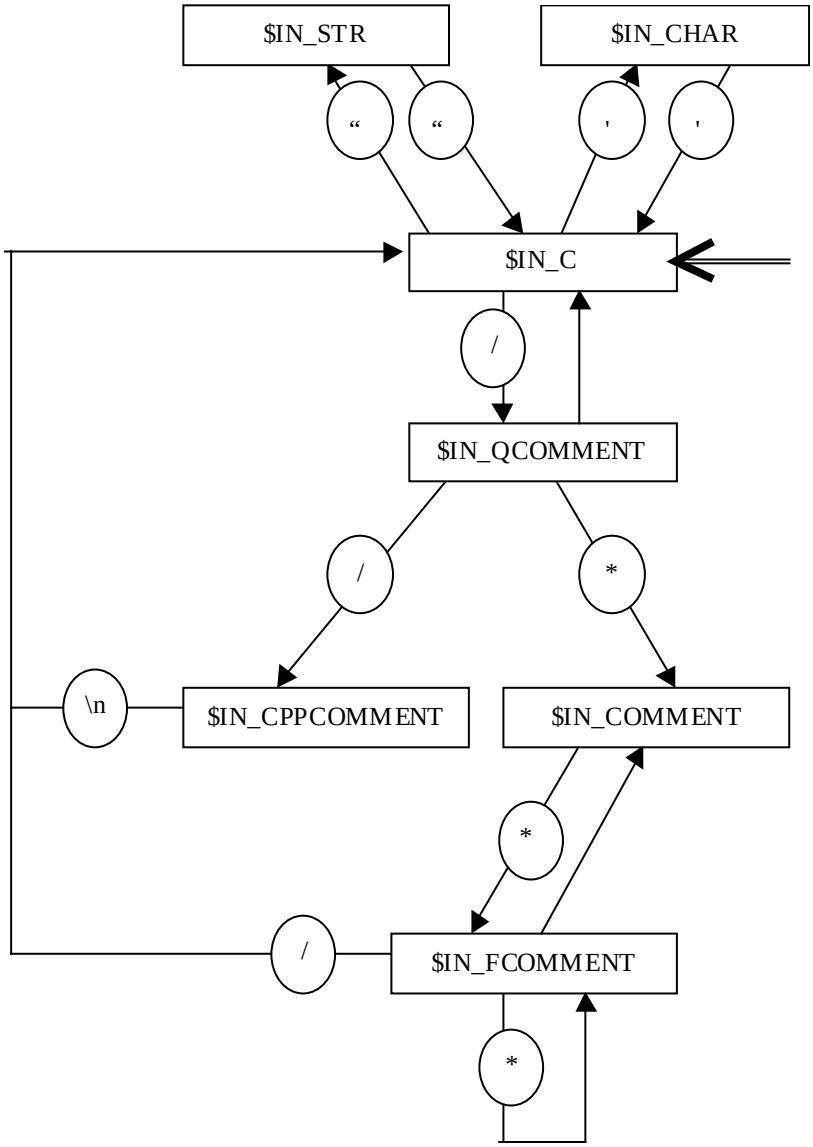
5. Exemples d'utilisations

Vous trouverez dans ce chapitre différents exemples d'utilisation de `fsmg`. Ces sources se trouvent dans le répertoire «`samples`» et doivent être traduits puis compilés pour être exécutés.

5.1 CComment

Ce programme se trouve dans le répertoire «`samples`». Il se compile par «`fsmg ccomment`» suivi de «`make ccomment`» ou «`nmake ccomment.exe`». Il s'agit d'un filtre de code C, le programme reçoit les caractères successifs d'un fichier, et le programme analyse les caractères pour reconnaître les commentaires, les chaînes de caractères, les caractères entre *cottes* afin de supprimer tous les commentaires du fichier. Cela permet dans l'état `$IN_C` de détecter certains mots clefs pour les interpréter ou pour ajouter du code.

Pour utiliser ce programme, tapez «`ccomment file.ext`» où `file.ext` est le fichier C à analyser. Voici le graphe de transition des états de ce programme.



Les traitements des caractères slash-inverse et `EOF` ne sont pas indiqués pour ne pas alourdir le schéma. Chaque état attend un caractère servant d'événement. Si ce caractère permet une transition, alors l'état courant change. Sinon, l'état reste inchangé. Une flèche simple, sans caractère de transition est utilisée lorsqu'aucun caractère ne permet de transition.

Cet exemple utilise un automate local, c'est-à-dire que les événements sont chargés par l'automate et non fournis par l'appelant. Pour cela, le source déclare le `#define FSM_BEFORE` pour valoriser la variable `fsm_event`. Les états correspondent aux états possibles de l'analyse.

Le traitement par défaut d'`escape` permet de traiter le cas du caractère slash inverse pour ne pas analyser le caractère suivant. Quel que soit l'état en cours, il ne faut pas traiter le caractère suivant un slash inverse.

Pour que `fsmg` utilise le traitement par défaut des événements en priorité, la commande «`%priority event`» est rajoutée dans la zone déclarative.

Si le caractère slash arrive, plusieurs cas peuvent se produire, soit il s'agit de l'opérateur de division. Soit c'est le début d'un commentaire du format C++, soit c'est le début d'un commentaire C classique. Un état est donc créé pour attendre un deuxième caractère permettant de discerner le cas présent, il s'agit de l'état `$IN_QCOMMENT`.

S'il s'agit de l'opérateur division, le traitement remet le dernier caractère analysé dans le fichier pour que celui-ci soit traité par l'état `$IN_C`.

S'il s'agit d'un commentaire C classique, il faut détecter la fin de ce commentaire. L'état `$IN_COMMENT` attend une étoile pour passer à l'état `$IN_FCOMMENT`.

Dans ce cas, trois situations peuvent se produire.

- Soit le caractère suivant est un slash, dans ce cas il s'agit de la fin du commentaire ;
- soit c'est un autre caractère étoile, il faut donc rester dans cet état ;
- soit enfin, il s'agit d'un autre caractère, dans ce cas il faut retourner à l'état `$IN_COMMENT`.

Le reste du source n'amène pas de commentaire particulier, dans l'état `$IN_C` vous pouvez rajouter des événements pour éventuellement modifier le source C comme par exemple, un traitement particulier lors de la détection du caractère dollar.

5.2 Date

Ce programme se trouve dans le répertoire «`samples`». Il se compile par «`fsmg idate`» suivi de «`make idate`» ou «`nmake idate.exe`». Il utilise le même principe que les exemples `time`. Il demande la date courante au format américain, le mois sur deux chiffres, un caractère de séparation, le jour, une séparation et l'année sur deux chiffres. Le programme teste la validité des jours par rapport au mois, il est donc impossible d'entrer le `4/31/91` ou le `2/30/92`.

Pour des raisons de clarté du source, le traitement des années bissextiles n'est pas effectué.

5.3 StatFsm

Ce programme se trouve dans le répertoire «`samples/statfsm`». Il s'agit d'un exemple un peu particulier. Ce n'est pas un source `.fsm`, mais un squelette. Celui-ci génère un source C affichant les différents paramètres d'un automate.

Pour tester cet exemple il faut donc posséder un source `.fsm`, le traduire avec ce squelette, compiler le fichier généré, et exécuter le programme.

Un fichier batch est présent dans le répertoire pour faciliter ces démarches. Il s'appelle `stat.sh` ou `stat.bat`. Il faut lui fournir les paramètres classiques de `fsmg` avec en dernier le fichier à traiter. Attention, il ne faut pas déclarer de fichier de sortie.

Voici un exemple de lancement : `stat -e test`. Le batch crée le fichier `tmp.c`, le compile et obtient le programme `tmp` ; puis, il le lance pour afficher différentes informations sur l'automate et enfin efface toute trace des différents fichiers intermédiaires nécessaires.

Cette façon de procéder est assez cavalière mais elle a le mérite de montrer la souplesse et la puissance que les squelettes peuvent apporter.

6. Conclusion

J'espère que `fsmg` facilitera vos développements. Des évolutions sont d'ores et déjà à l'étude : un nouvel algorithme spécialisé dans les grands automates ou l'évolution de la syntaxe vers une plus grande puissance. Le source étant ouvert, vous avez la possibilité de l'améliorer comme il vous semble.

ANNEXES

Annexes

Vous trouverez dans ces annexes : la syntaxe du langage au format BNF ; des informations concernant la portabilité du programme ainsi que ses limites ; une suite de questions/réponses types sur les éventuels problèmes que vous pouvez rencontrer ; la liste des messages d'erreurs ; un lexique rappelant la signification de certains termes ; et enfin, un index vous permettant de retrouver rapidement les informations qui vous intéressent.

6.1 Syntaxe type BNF

Les tokens:

<state> : `$[A-Za-z0-9_]+`
<event> : (`[A-Za-z_][A-Za-z0-9_]*` | `[0-9]+` | `'`{*lettre*}`'`)

La syntaxe:

prg : *def** `%%` *rule+ tail*
def : `%{` (C classique) `%}`
`%start` *<event>*
`%extern` *nlist*
`%priority event`
`%priority state`
`%max` *<int>*
nlist : *<event>*
nlist virgule <event>
rule : *states* `:` *events exec rrule*
rrule : `;`
`|` *events exec rrule*
states :
states virgule <state>
events :
events virgule <event>
exec :
`{` (C classique avec conversion des `$`...) `}`
tail :
`%%` (C classique)
virgule :
`,`

Les commentaires classiques du C peuvent être ajoutés. Les chaînes encadrées par «`/*`» et «`*/`» ne sont pas prises en compte. Les chaînes commençant par un double slash («`//`») et finies par un retour chariot sont également ignorées.

6.2 Portabilité

Le source de `fsmg` est conforme à la norme ANSI C et peut être compilé avec les compilateurs GNU. Les fichiers C générés sont parfaitement portables sous d'autres systèmes d'exploitation.

6.3 Limitations

- Seuls les quinze premiers caractères des états et des événements sont significatifs.
- Le nombre d'événements et d'états par règle est limité par la mémoire disponible.
- Le nombre de règles est limité par la mémoire disponible.

6.4 Question/Réponse

Question : Puis-je utiliser `fsmg` seul, sans les fichiers squelettes ?
Réponse : Oui, `fsmg` possède une version non commentée de chaque squelette dans le programme. Utilisez pour cela le paramètre `-m` qui demande à `fsmg` d'utiliser une de ces versions internes. Vous ne devez donc posséder au minimum que le programme `fsmg`.

Question : Si je dévermine l'automate avec un dévermineur, celui-ci pointe sur des lignes incorrectes lorsque j'entre dans l'automate.
Réponse : `fsmg` déclare des `#line`s pour indiquer d'où les lignes suivantes sont extraites. Si vous n'utilisez pas de fichier squelette, `fsmg` ne déclare pas de `#line`s pour le squelette interne qu'il utilise. Le compilateur ne déclare donc pas de source pour l'automate. Mettez des points d'arrêt dans les traitements de votre automate, ou utilisez la version du squelette en fichier fournie avec le programme. Dans ce cas, `fsmg` déclare un `#line` à chaque changement de fichier, le débuggeur passe du squelette à votre source.

Attention, dans les fichiers squelettes, certains `#line`s sont rajoutés manuellement, ils permettent de re-synchroniser le source lors de `#if` ayant échoué. Si vous modifiez l'un de ces sources, n'oubliez pas d'ajuster les valeurs de ces `#line`s, dans le cas contraire, le débuggage pointera sur des lignes incorrectes.

Vous pouvez également utiliser le paramètre «`-l`» sur la ligne de commande pour supprimer tous les `#line`s du fichier.

Question : Comment dans un traitement exécuter un autre traitement déjà déclaré ?
Réponse : Vous pouvez rappeler l'automate dans un traitement en lui fournissant l'état et l'événement déclenchant le traitement que vous voulez exécuter.
Exemple, dans un traitement vous pouvez coder «`return(fsm($DEBUT,CLAVIER));`». L'utilisation récursive d'un automate permet beaucoup de choses, l'état n'étant pas statique à l'automate vous avez toute liberté pour l'utiliser judicieusement.

Question: Mon automate fonctionne avec un squelette mais pas avec un autre.
Réponse: Vous avez certainement utilisé des spécificités d'un squelette, comme des `goto`s entre traitements, des `continue`s ou directement les variables `fsm_state` ou `fsm_event` plutôt que `$state$` et `$event$`. Il ne faut pas procéder ainsi si vous désirez une portabilité. Modifiez votre source en utilisant par exemple la récursivité à la place de `goto`.

6.5 Messages d'erreurs

6.5.1 Erreur

File.ext`(`*line*`) : error FSM0001 : not enought memory !`
Indique que le programme ne dispose pas d'assez de mémoire pour convertir votre source. Rajoutez de la mémoire à votre système, enlevez des résidents ou découpez votre automate en plusieurs plus petits.

File.ext`(`*line*`) : error FSM0002 : syntax error : parse error`
Indique qu'une erreur de syntaxe est détectée à la ligne indiquée du fichier. Vérifiez votre source à la ligne signalée ainsi que sur les lignes précédentes. Vérifiez qu'il ne manque pas de point-virgule, d'accolade fermante ou que vous n'avez pas utilisé le caractère deux-points à la place du caractère «pipe».

File.ext`(`*line*`) : error FSM0003 : parser stack overflow !`
Indique que l'analyse syntaxique ne peut aboutir, vérifiez votre source, la syntaxe ne doit pas être correcte.

`File.ext(line) : error FSM0004 : state/event defined more than one !`
Indique qu'un couple *state/event* est défini plus d'une fois dans le fichier. Modifiez votre source.

`File.ext(line) : error FSM0005 : "str" isn't a state !`
Indique que la chaîne *str* n'est pas un état et ne peut donc pas être convertie. Modifiez votre source.

`File.ext(line) : error FSM0006 : "str" unknow !`
Indique que le mot clef *str* dans un squelette est inconnu. Vérifiez votre source ou la version de `fsmg` que vous utilisez.

`File.ext(line) : error FSM0007 : no state defined !`
Indique que vous n'avez pas déclaré d'état pour l'automate. Modifiez votre source.

`File.ext(line) : error FSM0008 : no event defined !`
Indique que vous n'avez pas déclaré d'événements pour l'automate. Modifiez votre source.

`File.ext(line) : error FSM0009 : %start to no state !`
Indique que vous avez déclaré un `%start` sur un état inconnu. Modifiez votre source.

`File.ext(line) : error FSM0010 : %start defined more than one!`
Indique que vous avez déclaré plus d'un `%start`. Modifiez votre source pour ne laisser qu'un seul `%start`.

`File.ext(line) : error FSM0011 : scanner read file failled !`
Indique que le programme n'arrive pas à lire votre source. Vérifiez le, ainsi que ses droits d'accès.

`File.ext(line) : error FSM0012 : scanner push-back overflow !`
Indique que le programme n'arrive pas à mémoriser suffisamment de caractères pour l'analyse. Vérifiez qu'il n'existe pas de lignes trop longues dans le programme ainsi que des états ou des événements trop longs.

6.5.2 Warning

`File.ext(line) : warning FSM0001 : choose "state" and not "event"`
Indique qu'un conflit est possible entre deux traitements par défaut. Le programme a choisi le traitement concernant l'état plutôt que celui concernant évènement. Vous pouvez modifier la priorité des traitements par le paramètre «`-e`» sur la ligne de commande (voir page 22), ou utilisez la commande «`%priority`». Ce message n'empêche pas l'exécution de votre source, il est là pour vous signaler le choix pris par `fsmg`. Vous pouvez demander de ne plus le voir apparaître en utilisant le paramètre «`-w`» sur la ligne de commande.

`File.ext(line) : warning FSM0002 : %extern "event" not use !`
Indique que vous avez déclaré un événement `%extern` qui n'est pas utilisé. Cet événement n'est pas pris en compte.

6.6 Lexique

Automate	Routine effectuant un traitement suivant des événements extérieurs. Un automate possède un état courant qui le caractérise à un moment donné. Un même événement peut entraîner plusieurs traitements différents suivant les événements antérieurs.

État	Situation courante d'un automate. Peut être assimilé à une variable global ou à un attribut. Il caractérise un automate à un moment donné.
Événement	Informations fournies à un automate pour lui demander de réagir. Celui-ci, suivant son état courant effectue un traitement modifiant éventuellement son état. Un événement peut être n'importe quelle information intéressant un automate, comme un caractère tapé au clavier, le déplacement de la sourie, une alarme déclenchée, l'explosion d'une centrale nucléaire... pour toutes ces situations, l'automate doit réagir.

6.7 Index