Proposition de présentation au G.D.R. de Programmation

8,5 un environnement de développement pour le langage 81/2

Olivier Michel, Dominique De Vito

LRI, URA 410 du CNRS
Bâtiment 490, Université de Paris-Sud
F-91405 Orsay Cedex FRANCE
email : {michel, devito}@lri.lri.fr

I. Introduction

Dans ce papier, nous présentons **8,5** un environnement pour le langage **81/2**. Nous décrirons rapidement les fonctionnalitées du langage 81/2 à travers l'exemple d'un automate cellulaire connu : le jeu de la vie (Cf. [GIA 91a], [GIA 93], [GIA 94] pour une présentation plus systématique du langage). Nous décrirons ensuite les phases de la compilation implémentées dans 8,5. Nous donnerons à titre indicatif quelques analyses de performance sur la génération de code issue d'expressions 81/2.

II. Le jeu de la vie comme système dynamique

L'exemple du jeu de la vie va nous servir à illustrer les caractéristiques principales de 81/2. Nous décrirons le principe du jeu de la vie, puis une possible implémentation en 81/2 et enfin le résultat de la simulation.

Le monde du jeu de la vie est constitué d'un tableau de cellules organisées en grille rebouclée (un tore). Chaque cellule a huit voisins possibles et est soit dans l'état vivant soit dans l'état mort. L'état des cellules évolue au cours du temps suivant une loi précise. Ici, nous adoptons la loi suivante : une cellule vivante reste vivante à la génération suivante si le nombre de ses cellules voisines est égale à deux ou à trois et elle meurt sinon ; une cellule morte devient vivante si le nombre de ses voisins est égal à 2 ou 3.

A un instant donné, l'état du jeu est représenté par l'état des cellules. Cet état, qui correspond à l'*ensemble* des états des cellules est appelé une *collection* en 81/2. Une collection [SIP 91] est un ensemble de donnée que l'on peut manipuler comme un tout. Dans l'exemple du jeu de la vie, la transition entre un état et l'état suivant se fait simultanément pour toutes les cellules en même temps et avec la même loi.

La suite des états du jeu constitue la *trajectoire* du système dynamique. Une succession d'états se nomme en 81/2 un *stream*. L'évolution du jeu de la vie est donc une suite d'états, soit encore un stream de collection. Cette structure de données est appelée **tissu** dans le cadre de 81/2.

Un programme 81/2 est un système d'équations permettant de définir un tissu. Voici le système d'équations correspondant à notre version du jeu de la vie :

```
System Life = {
        t@0 = t0;
        t   = if total < 2 then 0 else if total > 3 then 0 else ($t + 1) fi fi;

        past_t = if ($t > 0) then 1 else 0 fi when Clock

        ligne = if ('20 % 3) == 0 then 0 else 1 fi;
        t0 = 0:[9, 20] # { ligne, if ligne then 0 else 1 fi, ligne } # 0:[8, 20];

        north = 't + {{{1, 0}}}:[20, 20, 2];
        North = if north > 19 then 0 else north fi;

        south = 't + {{{-1, 0}}}:[20, 20, 2];
        South = if south < 0 then 10 else south fi;

        west = 't + {{{0, 1}}}:[20, 20, 2];
        West = if west > 19 then 0 else west fi;

        east = 't + {{{0, -1}}}:[20, 20, 2];
        East = if east < 0 then 10 else east fi;

        total =  past_t.(North) + past_t.(South) + past_t.(West) + past_t.(East);
}
```

Le tissu `t` correspond au tissu du jeu de la vie. Les tissus `past_t`, `North`, `South`, … sont des tissus auxiliaires. `t` est un tableau de 20×20 éléments, chacun représentant une cellule. L'état d'une cellule est un entier dont la valeur représente l'age de la cellule. La forme du tableau et le type de ses éléments constituent la *géométrie* d'une collection. Le calcul des géométries est effectué implicitement par le compilateur 8,5 (Cf III.c.1.).

La définition de `t` est composée de deux équations :

- « `t@0 = t0;` » spécifie la première valeur de la suite `t`. Le qualificatif "`@0`" est une quantification temporelle qui garde la définition.
- « `t = if total < 2 then 0 else if total > 3 then 0 else ($t + 1) fi fi;` » est une équation non quantifiée qui définit `t` pour tous les autres éléments de la suite `t`.

La valeur initiale de `t` est celle du tissu `t0`. Ce tissus est définit par la concaténation (`#`) de trois tableaux de taille respective 9×20, 3×20 et 8×20. Le tableau 3×20 du milieu est constitué par le regroupement de trois vecteurs de taille 20. L'opérateur "`:[]`" permet de coercer explicitement une constante scalaire en une collection de géométrie quelconque. Cette coercion explicite est ici nécessaire afin de forcer la géométrie du tissu `t0` (sinon le compilateur inférerait un tissu `t0` de géométrie 5×20).

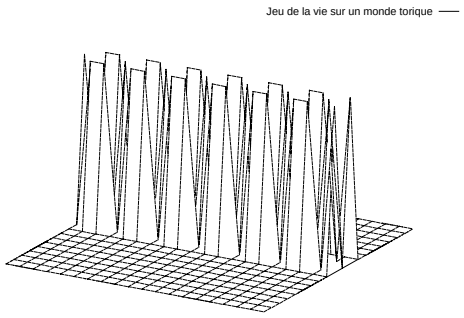
Le tissu total comptabilise pour chaque cellule le nombre de ses voisins. Pour cela, on utilise le tissu auxiliaire `past_t`. Un élément de `past_t` a pour valeur 1 si la cellule était vivante à l'instant précédent, 0 sinon. C'est l'opérateur "`$`" qui permet de "retarder" un stream de valeurs.

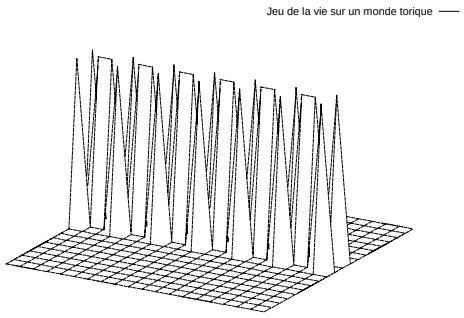
Afin de référencer une cellule voisine, on utilise une opération généralisée de décalage sur les collections. Cette opération généralise les opérations de gather/scatter bien connues en Fortran. L'expression "`a.(b)`" représente une collection constituée des éléments de `a` adressés par les éléments de `b`. L'adresse d'un élément correspond au vecteur des indices nécessaires pour l'atteindre dans un tableau. L'opérateur "`'`" (iota en APL) appliqué à un tissu `c` permet de générer la collection des adresses de ses éléments. Ainsi le tableau `north` correspond au décalage vers le sud d'un tableau bi-dimensionnel : on ajoute en effet 1 à la

première coordonnée. Le décalage d'un tableau vers le sud permet d'atteindre en un point courant le voisin nord. La collection `North` rectifie les adresses spécifiées par le tableau `north` afin de réaliser le rebouclage des bords. Il en va de même pour les quatre autres directions.

Dans ce qui précède, nous avons parlé parfois en terme de collection et parfois en terme de suite. En fait, les équations précédentes définissent bien des tissus, c'est à dire des streams de collections. Mais la combinaison de ces deux structures de données dans un tissu est orthogonale : les opérations qui s'appliquent à l'aspect collection d'un tissu s'appliquent pour chacune des valeurs du stream et les opérations sur les streams s'appliquent à chaque élément d'une collection. L'opérateur d'échantillonnage "`when Clock`" permet de spécifier la "vitesse temporelle" du stream `past_t`. En fait, dans ce programme, toutes les suites évoluent à la même vitesse mais de manière générale un programme 81/2 permet de définir des tissus allant à des "vitesses dans le temps" différentes.

Les deux figures suivantes correspondent à la visualisation du tissu `t` pour les deux premiers instants du stream :





III. L'environnement 8,5

L'environnement 8,5 est actuellement en cours de développement au sein de l'équipe Architectures Parallèles du L.R.I. et contient actuellement environ 28000 lignes de caml-light (pour l'environnement) et 5600 lignes de C (pour la machine virtuelle utilisée lors de l'exécution au vol des expressions 81/2). Nous allons décrire dans cette section l'environnement 8,5 ainsi que ses caractéristiques principales :

- exécution au vol d'expression 81/2,
- les possibilités de tracé graphique des tissus 81/2,
- le processus de compilation d'une expression 81/2,
- la génération de code C.

Nous donnerons à titre indicatif une comparaison des performances entre le code C généré par le processus de compilation et un code C ad-hoc écrit manuellement.

III.a. L'évaluation au vol d'équations 81/2

L'environnement 8,5, permet d'entrer les expressions définissant un système d'équations les unes après les autres et d'évaluer le tissu ainsi défini. Par exemple, un compteur temporel se définit de la façon suivante : (=> est le signe d'invite du système)

```
=> t@0=0;
t@0 = 0 ;
=> t=$t+1 when Clock;
t = ($t + 1) when Clock; t@0 = 0 ;
```

L'environnement 8,5 permet d'évaluer des équations 81/2 de deux façons différentes :

- en évaluant au vol dans l'environnement à tout instant les équations,
- en générant un source C compilable puis exécutable.

La commande `!eval` permet de lancer la compilation et la génération d'un code pour une machine virtuelle SIMD qui est évalué au vol. Ainsi, avec l'exemple précédent :

```
=> !eval 1 t;
tic 0:
         14:  val:  int t = ($t + 1) when Clock; t@0 = 0        => { 0 }
tic 1:
         14:  val:  int t = ($t + 1) when Clock; t@0 = 0        => { 1 }
```

Cette commande partage avec la commande `!compile` les premières étapes du processus de compilation (détaillées en section III.c.) :

- liaisons des identificateurs, macro-expansion des fonctions…
- inférence de la géométrie,
- synthèse de l'ordonnancement des calculs.

Suivant la commande, le processus de compilation se poursuit par la génération de code pour la machine virtuelle et son exécution ou bien par la génération d'un source C. La machine virtuelle 8,5 est une machine virtuelle SIMD qui peut être vue comme un "co-processeur" de tableau écrite en C.

III.b. Les sorties graphiques sous 8,5

Il est possible de visualiser un tissu 81/2 sous sa forme numérique (un scalaire pour une collection de dimension 1, un tableau pour une collection de dimension 2 ou bien une hiérarchie de vecteurs pour une collection de dimension supérieure à 2) ou bien sous une forme graphique, tracée par GnuPlot [WIL 93]. 8,5 dispose de fonctionnalités issues de GnuPlot ainsi que de fonctionnalitées propres, telles que :

- tracé en 2D, 3D et en paramétrique 2D et 3D ;
- tracé en temps-réel d'un tissu : le mode normal de tracé d'un tissu est celui qui trace l'état du tissu à l'issue des `n` itérations demandées lors de son évaluation. Il est cependant possible de voir l'évolution du tissus à chaque instant ou uniquement aux instants de variation. On a ainsi une vision *temps-réelle* de la dynamique du tissu ;
- tracé de tous les tics ou bien uniquement des tops ;
- tracé dans plusieurs fenêtres de plusieurs tissus simultanément ;

- avoir des attributs différents pour chaque fenêtre (couleur, légende, échelle, type graphique de tracé…).

La sélection de toutes les caractéristiques du tracé se fait à l'aide de contextes graphiques dont on modifie les attributs et que l'on attache à un ou plusieurs tissus 81/2. Par exemple, la spécification du contexte graphique utilisés pour la visualisation du jeu de la vie est le suivant :

```
!gc u =   {nomemory; realtime=1; linestyle=lines; color = 0; tick; window = 0;
noparametric;         refresh; title ="Jeu de la vie sur un monde torique"};;
```

III.c. Le processus de compilation d'une équation 81/2

Le processus de compilation d'une expression 81/2 est décomposé en quatre phases principales :

III.c.1. Synthèse de la géométrie

Il est nécessaire de synthétiser les types des programmes 81/2 afin d'assurer une exécution correcte et prévisibles à la compilation de ceux ci (on rappelle que 81/2 à été conçu pour supporter un modèle d'exécution qui soit totalement statique [GIA 91b]). En effet, le programmeur ne fournit aucune information de type lors de l'entrée des équations dans 8,5. Si le programme est incorrectement typé, alors il ne sera pas possible de décider à la compilation de certains paramètres de son exécution (espace mémoire utilisé…).

La phase de synthèse de type est automatiquement effectuée au moment de la compilation par 8,5 et rejette les programmes incorrects. A l'issue de cette phase, les programmes sont correctement typés et la phase de calcul des horloges des équations peut commencer.

III.c.2. Calcul des horloges

On associe à tout tissu T un couple de prédicats nommés (d_T, h_T) définissant le domaine et l'horloge du tissu T. Le domaine de T est vrai à partir du momment où T a une valeur définie. L'horloge de T est vraie quand le tissu T à une nouvelle valeur à l'instant considéré. En effet, un tissu est une suite (de collections) "à trous" qui ne dispose pas de valeur à chaque instant. L'horloge d'un tissu désigne les instants du temps pour lesquels la suite est valuée. Plus précisément :

- $d_T(t)$ est une fonction totale qui a pour valeur vraie si la valeur du tissu T est définie à l'instant t.
- $h_T(t)$ est définie si $d_T(t)$ est définie et a pour valeur vraie si la suite associée à T a une valeur à l'instant t.
- On désigne par $v_T(t)$ la valeur courante de T à l'instant t. Si $h_T(t)$ est à faux alors $v_T(t)$ à pour valeur $v_T(t_0)$ où $h_T(t_0)$ est vrai et $\nexists$ t', $t_0 < t' \leq t \ \wedge \ h_T(t')$.

Les prédicats d_T et h_T sont inférés à l'aide d'un système de typage. Ces types sont représentés dans le compilateur par des arbres de décisions binaires réduits et ordonnés (ROBDD Cf. [BRY 86]). On se référera à [DEV 94] pour une descriptions complète du système de typage.

III.c.3. Ordonnancement

La résolution des équations 81/2 nécessite l'ordonnancement des calculs correspondants aux expressions apparaissant en partie droite des équations. Le séquencement des expressions est réalisé par un tri topologique sur les relations de dépendances entre les expressions. Les équations récursives correspondent à des cycles qui ne peuvent être traités par cet algorithme. Des techniques spécifiques développées à partir de la sémantique dénotationelle du langage sont utilisées pour la résolution de ces cycles. Une fois le séquencement des instructions effectué, le compilateur calcule l'espace mémoire requis par le programme pour son exécution.

III.d. La génération de code C

III.d.1. La génération de code C

Un programme C généré à partir d'un programme 81/2 correspond à une évaluation séquentielle et cyclique. En effet, le programme calcule la valeur des différents tissus 81/2 dans l'ordre des instants croissants et finit le calcul des valeurs correspondant à un instant donné avant de commencer les calculs correspondants à un instant suivant. Dans le schéma de compilation adopté, on mémorise les résultats élaborés à l'instant t qui sont nécessaires aux calculs des valeurs d'un instant t'>t. De ce fait le temps d'exécution du programme C obtenu est linéaire avec le nombre d'instants de la simulation. Un programme C généré par le compilateur 8,5 est composé de trois parties :

- la première partie correspond à la déclaration des variables C (zones mémoires) nécessaires ;
- dans la seconde partie, le programme calcule pour chaque instant le domaine de définition et l'horloge des expressions et sous-expressions 81/2 ainsi que la valeur des différents tissus ;
- dans la troisième partie le programme sauvegarde à chaque instant la valeur des différentes expressions référencées par un opérateur de délai afin de pouvoir utiliser ces valeurs lors des instants suivants.

En première approximation, le code généré à l'allure suivante :

```
pour chaque instant t
      pour chaque tissu T dans l'ordre du séquencement
            calculer dT(t)
            si dT(t) alors calculer hT(t)
                        si hT(t) alors calculer vT(t)
```

III.d.2. Utilisation du code généré

Le compilateur génère à partir d'un programme 81/2 une fonction C appelée `compute()`. Cette fonction prends pour argument l'instant de fin de la simulation. Pour chaque instant, la fonction `compute()` appelle la fonction `ticlude()` avec pour arguments l'instant courant, la valeur courante de l'horloge du tissu T qui à été compilé et un pointeur sur la valeur courante de T. Un utilisateur est libre d'utiliser la fonction `compute()` dans n'importe quel programme en fournissant sa propre fonction `ticlude()`.

Un programmeur à aussi la possibilité d'utiliser une des deux fonctions `ticlude()` standard présente dans la librairie `lib8,5.a`. La première version ne fait rien de particulier ; la seconde signale sur la sortie standard les valeurs du tissu T. La libraire standard `lib8,5.a` comporte en outre une fonction `main()` minimale qui réalise le programme suivant : `prelude(); compute(); postlude()`. Les fonctions `prelude()`, `postlude()` et `ticlude()` sont des fonctions dont le programmeur peut fournir une version afin de paramétrer la simulation standard. Finalement, le programmeur peut agir à plusieurs niveaux pour paramétrer une simulation 81/2 :

- embarquer la fonction `compute()` issue de la compilation d'un programme 81/2 dans un programme ad-hoc. Il doit alors fournir une fonction `ticlude()` qui a le contrôle à chaque instant de la simulation (cette fonction joue un rôle analogue aux callbacks des interface graphiques).
- utiliser le programme standard fourni sous la forme d'une libraire `lib8,5.a`. Cette librairie fournie des fonctions `main()`, `prelude()`, `postlude()` et `ticlude()` réduites au minimum. Chacune de ces fonctions peut être remplacée par une version propre à l'utilisateur.

III.d.3. Optimisation de la recopie des délais

On associe à chaque délai une zone mémoire pour mémoriser la valeur courante d'un tissu d'un instant à l'autre. Ces mémorisations sont réalisées par des recopies qui peuvent être coûteuses si on a à faire à des tableaux de grandes tailles. Toutefois, il est souvent possible de se passer de la recopie de T si à chaque instant, il est possible d'utiliser à la place de $T la valeur du tissu T avant sa mise à jour (une optimisation analogue est présentée dans [HAL 91]). Par exemple le programme :

```
for (t = a; t < MaxDate; t++)
{
        a = delai_$x + 1;
        x = 3 + a;
        …
        /* recopie des délais */
        delai_$x = x;
}
```

peut se réécrire dans le programme équivalent plus performant suivant :

```
for (t = a; t < MaxDate; t++)
{
        a = x + 1;
        x = 3 + a;
        …
}
```

D'autres optimisations ont été implémentées comme par exemple le partage des expressions.

III.d.4. Évaluation de la performances de la génération de code C.

Afin de déterminer les performances du compilateur 8,5, nous avons choisis de comparer l'implémentation d'un programme écrit en 81/2 sous 8,5 et l'implémentation directe du problème en langage C. Les équations 81/2 sont compilés et génèrent un source C. Ce sont les temps d'exécution de ces deux exécutables que nous comparons. Le programme de test est basé sur la résolution d'une équation aux dérivées partielles par une méthode aux différences finies [SMI 85].

Sur une station Sun Sparc10 bi-processeurs sous Solaris2.1, les ratios des temps d'exécutions des deux programmes sont les suivants :

Nombre d'itérations Dimensions du problème	100	1000	10000
100	1 1,5	1,3 1,2	1,3 1,3
1000	1,5 1,2	1,6 1,2	1,6 1,2
10000	1,6 1,3	1,6 1,3	1,6 1,3

Cette table présente les ratios entre les performances du code généré par 8,5 et du code écrit en C manuellement. Les deux codes sont compilés par `gcc -O`. La première valeur correspond au compilateur actuel. Celui ci est désavantagé par l'utilisation d'une opération générale de gather/scatter alors que le code C manuel se contente d'un décalage d'indice. Il serait facile de détecter que l'opération de gather/scatter peut dans ce cas là être optimisée. Le second chiffre correspond à la comparaison du code généré par 8,5 avec un code C manuel utilisant lui aussi une opération générale de gather/scatter. On se reportera à [DEV 94] pour des comparaisons plus approfondies sur la génération de code C dans le compilateur 8,5.

IV. Conclusion

Les premiers tests effectués valident l'approche adoptée lors de chaque phase de la compilation. Ceci montre en particulier que l'expressivité ne se paye pas au détriment de l'efficacité. Néanmoins, les performances du code généré sont moins bonnes qu'un code C natif dès que l'on ne manipule que des tableaux de petites tailles (moins de 20 éléments). Cette dégradation des performances est due au surcoût de calcul des horloges qui devient sensible avec les tableaux de petites tailles. Une voie prometteuse consiste à étudier les optimisations des calculs des horloges en terme d'interprétation abstraite à partir des types d'horloges.

Nous étudions actuellement des extensions dynamiques de 81/2. En effet, le compilateur actuel n'admet que le sous ensemble des équations spécifiant des tissus dont le comportement est prévisible à la compilation. Cette restriction interdit la modélisation de tableaux à dimensions dynamiques ou la simulation de phénomènes de croissance. L'exemple suivant est un programme 81/2 rejeté actuellement par le compilateur et qui permettrait le calcul des lignes du triangle de Pascal : à l'instant n, le tissu `pascal` est la collection des C_n^p.

```
pascal = (($pascal # 0) + (0 # $pascal)) when Clock;
pascal@0 = 1
```

l'exécution produit le résultat suivant :

```
Top:0        : { 1 } : int[1]
Top:1        : { 1, 1 } : int[2]
Top:2        : { 1, 2, 1 } : int[3]
Top:3        : { 1, 3, 3, 1 } : int[4]
Top:4        : { 1, 4, 6, 4, 1 } : int[5]
Top:5        : { 1, 5, 10, 10, 5, 1 } : int[6]
…
```

V. Références

[BRY 86] R.E. Bryant, *Graph based algorithms for boolean function manipulation*, IEEE Trans. Comp., Vol C-35, N°8, August 1986, Page 677-691.

[DEV 94] D. De Vito, Compilation portable d'un langage déclaratif à collections et flots de donneés synchrones, Rapport de D.E.A., Université Paris XI, Juin 1994.

[GIA 91a] J.-L. Giavitto, *A synchronous data-flow language for massively parallel computers*, Proc. of Parallel Computing'91, 3-6 September 1991, London.

[GIA 91b] J.-L. Giavitto, *81/2 : Un modèle MSIMD pour la simulation massivement parallèle*, mémoire de thèse, Décembre 1991 à Orsay. Numéro d'ordre : 1872.

[GIA 93] J.-L. Giavitto, J.-P. Sansonnet, *81/2 : data-parallélisme et data-flow*, Technique et Science Informatique, Octobre 1993.

[GIA 94] J.-L. Giavitto, J.-P. Sansonnet, *Introduction à 81/2*, Rapport de recherche, L.R.I. N°901, Mai 1994.

[HAL 91] N. Halbwachs, P. Raymond, C. Ratel, *Generating efficient code from data-flow programs*, 3RD, Int. Symp. P.L.I.L.P. 91, August 91, Passau, LNCS-528.

[SIP 91] J. M. Sipelstein, G. E. Blelloch, *Collection-oriented languages*, proc. of the IEEE, Vol. 79, N° 4, april 1991, pp. 633-649.

[SMI 85] G. D. Smith, Numerical solution of partial differential equations: finite difference methods, third edition, 1985, Oxford Applied Mathematics and Computing science series, Oxford University Press.

[WIL 93] T. Williams, C. Kelley, (C) GNUPLOT An Interactive Plotting Program, reference manual, Free Software Foundation, 1993