M2R/DEA Informatique
Université Montpellier II

Modélisation de systèmes dans un contexte spatialisé

Quelques éléments de réflexion

Philippe REITZ (reitz@lirmm.fr)
LIRMM – INFO – DOC

Explication de texte…

Modélisation de systèmes dans un contexte spatialisé

Système = Processus =
espace d'états
+ dynamique

Spatialisé =
des contraintes sur l'interaction

Guide de lecture

Modéliser des systèmes dans un contexte spatialisé

en sciences physiques :
• équations différentielles
→ Modelica

en informatique :
• modèles de calcul parallèle
→ ChAM
8 ½ - MGS

Plan

- Présentation du contexte
  - CIRAD
  - Modélisation de systèmes dans un espace
- Modélisation de systèmes hybride
  - Les automates hybrides
  - Un langage de modélisation : Modelica
- Programmation de machines massivement //
  - Quels liens avec la modélisation de systèmes
  - Quelques travaux exemplaires
- Conclusion

Contexte : le Cirad

- Modélisation – simulation au Cirad
  - Cirad : recherche en agronomie
  - logiciel Sarra-H
    - Collaboration avec Ch. Baron, de l'équipe EcoTrop – département AMIS – Cirad Montpellier.
    - Simulateur du comportement de plantes
  - projet Safe
    - Collaboration avec Ch. Dupraz, Inra Montpellier.
    - comportement conjoint de plantes

Contexte : le Cirad

- Logiciel Sarra-H
  - Simulation du comportement d'une culture (riz, sorgho, mil, maïs, palmier à huile, …) traçant l'évolution de divers paramètres (ex : biomasse).
  - Composition de divers processus :
    - Eau : transpiration, évaporation, écoulement, assimilation (découpage atmosphère, sol, sous-sol)
    - Chaleur, Lumière
    - Biomasse
    - Flux de matières diverses : carbone, azote, …
  - Le nombre et la nature des processus varient dans le temps : des organes naissent puis meurent.

Contexte : la modélisation de systèmes

- Modélisation dans un cadre continu
  - Cadre classique de modélisation en agronomie.
    - E = espace des états (phases) *[continu]*
    - T = temps *[continu]*
  - Pas d'espace explicite
    - Modèle = équation différentielle ordinaire (EDO [*ODE*])
      - Théorie des systèmes ⇒ fonctions de transfert
    - Modèle = équation différentielle et algébrique (EDA [*DAE*])
      - EDA = EDO + contraintes algébriques
  - Espace explicite *[continu]*
    - Modèle = équation aux dérivées partielles (EDP [*PDE*]) ⇒ différentiation par rapport à toutes les dimensions : temps, espace.

Contexte : la modélisation de systèmes

- Modélisation dans un cadre discret
  - Dimensions toutes discrétisées
    - E = espace des états (phases) *[discret]*
    - T = temps *[discret]*
    - S = espace *[discret]*
  - Modèle = automate calculant l'état suivant :

$$x_{n+1} = f(x_n)$$

    - processus – automates de Markov

Analyse des principales difficultés

Liste non exhaustive (euphémisme ☺)…

- Modélisation
  - Couplage de modèles
    - modèles homogènes (tous continus, tous discrets)
    - modèles hétérogènes
  - Différences d'échelles (temps – espace – état)
  - Changement de nature des entités dans le temps
    - chaque phase de vie = un modèle

Analyse des principales difficultés

- Simulation
  - Langage de représentation
    - Equations différentielles ⇒ solveurs
    - Représentation par objets, agents
  - Outils d'analyse
    - Etudes de sensibilité
    - Argumentations des résultats
    - Optimisations de paramètres
    - Gestion des données de simulation

Plan (avancement)

- Présentation du contexte
  - CIRAD
  - Modélisation de systèmes dans un espace
- Modélisation de systèmes hybride
  - Les automates hybrides
  - Un langage de modélisation : Modelica
- Programmation de machines massivement parallèles
- Conclusion

Quelles pistes suivre ?

- Couplage de modèles hétérogènes
  - Les automates hybrides
    - modèle hybride = automate intégrant un état continu et un état discret, le temps étant continu.
  - Les équations différentielles et algébriques (EDA [*DAE*]) hybrides
    - autre façon de décrire un modèle hybride.

Principal défaut : pas de prise en compte explicite de la dimension spatiale.

Quelles pistes suivre ?

- Intégration explicite de la dimension spatiale dans les modèles discrets
  - Le projet MGS (équipe J.L. Giavitto, LaMI)
    - cadre purement discret (langage de programmation pour machines massivement parallèles).
    - vers une géométrisation de la programmation

Modélisation de systèmes hybride

- Modèle hybride :
  - modèle intégrant à la fois
    - une dimension continue
    - une dimension discrète
- Communautés scientifiques impliquées
  - Automatique
    - contrôle optimal de processus
  - Toute communauté ayant une activité modélisation / simulation de systèmes
    - tous les domaines de l'industrie : transport, énergie, information, …

Les automates hybrides

- Origine
  - Communauté de l'Automatique, début des années 90
    - contrôle d'un processus réel (modèle continu) par un calculateur (modèle discret)
  - problèmes classiques en automatique :
    - vérification (atteignabilité d'un état, contrôlabilité)
    - identification (optimisation de paramètres)
  - Article de référence : Thomas A. HENZINGER (1995)

Les automates hybrides

- Présentation rapide

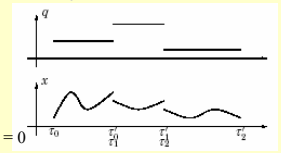
  (présentation détaillée juste après)
  - Automate hybride = automate caractérisé par :
    - son état initial
      - état discret + état continu
    - sa loi de transition
  - L'état évolue selon une loi s'appuyant sur un temps continu
    - alternance de phases continues et de transitions d'état instantanées.

Les automates hybrides

- Le modèle = automate, avec :
  - état = (q, x, τ) avec :
    - état discret q (pris dans un ensemble discret fini Q)
    - état continu x (pris dans un ensemble continu E)
    - temps τ (pris dans un ensemble continu T)
  - loi de transition, décomposée en 4 fonctions :
    - $f(q, x, x', \tau) = 0$: équation différentielle régissant l'évolution de l'état continu en phase stable
    - $g(q, x, x', \tau) \in \{0, 1\}$: prédicat détectant une transition d'état
    - $\delta(q, x, x', \tau) \in$ Q: fonction régissant la transition de l'état discret
    - $i(q, x, x', \tau) \in$ E : fonction régissant la transition de l'état continu (réinitialisation)

Les automates hybrides

- Sémantique
  - état courant = (q, x, τ)
  - phase stable
    - phase stable durant (τ, τ') $\Leftrightarrow \forall t \in [\tau, \tau'[, g(q, x, x', t) = 0$
    - durant toute cette phase, à chaque instant t,
      - l'état discret de l'automate reste à q
      - l'état continu x évolue selon l'équation différentielle f
  - phase de transition
    - transition de phase à l'instant t $\Leftrightarrow g(q, x, x', t) = 1$
    - lorsqu'une transition de phase est détectée, alors :
      - l'état discret de l'automate passe à $\delta(q, x, x', t)$
      - l'état continu x est réinitialisé à $i(q, x, x', t)$



Les automates hybrides

Une balle rebondissante

Position de la balle : $x \geq 0$

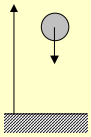
$\Rightarrow$ état continu = $(x, v = \dot{x})$

Accélération : $g > 0$

Coefficient d'amortissement : $c \in [0, 1]$

équation différentielle en q :

$$\begin{cases} \dot{x} = v \\ \dot{v} = -g \end{cases}$$

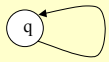


condition de la transition :

$x = 0$ et $v \leq 0$

réinitialisation :

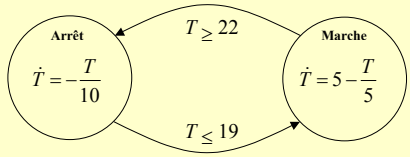
$v := -cv$



Les automates hybrides

Un thermostat

Température de la pièce = T



Les automates hybrides

- Principaux résultats
  - dans le cas général, ils sont tous dramatiques (questions indécidables, au mieux NP).
  - dans quelques cas particuliers (automates hybrides linéaires), les automaticiens ont réussi à trouver des algorithmes raisonnables pour résoudre leurs problèmes (cf. vérification).
    - système HYTECH d'Henzinger par exemple.

Les EDA hybrides

EDA = Equations Différentielles et Algébriques

- Description succincte
  - formalisme permettant d'exprimer les mêmes modèles hybrides que le précédent (automates hybrides), mais dans un style équationnel.
- Support formel du langage Modelica

Les EDA hybrides

EDA

Une EDA est de la forme :

$$F(x, \dot{x}, \ddot{x}, ..., \overset{(n)}{x}, t) = 0$$

Cette forme générale peut être ramenée, modulo l'introduction de variables et une récriture de l'équation, à une forme canonique :

$$F(x, \dot{x}, t) = 0$$

La résolution d'une EDA est rarement faisable analytiquement $\Rightarrow$ solutions approchées via des **solveurs** (ex : algorithme de Petzold – 1982).

Les EDA hybrides

EDA hybride

C'est une EDA à laquelle est ajoutée une variable discrète q, et qui se ramène à la forme canonique :

$$\begin{cases} \dot{x} = F(x, q, t) \\ 0 = G(x, q, t) \end{cases}$$

Le langage Modelica (1/2)

- Un langage déclaratif, orienté objet
- Un langage spécialement adapté à la modélisation/simulation de systèmes
  - applicable à tous les domaines
  - Modelica = UML pour les modélisateurs ?
- Un langage pour décrire des modèles hybrides

Le langage Modelica (2/2)

- Vers un langage de modélisation normalisé
  - Les spécifications sont normalisées
  - Les outils informatiques (compilateurs) ont pour seule obligation de se conformer à ces spécifications
- Domaines d'application à ce jour
  - Mécatronique, Electronique, Systèmes de puissance, Hydraulique, Aérodynamique, …

Pourquoi Modelica (1/2)

- Tout modèle ou toute simulation sur une plate-forme doit pouvoir être exprimé dans un langage, à la fois :
  - compréhensible par le modélisateur
  - traitable informatiquement
- La définition d'un langage impose de caractériser proprement tous les objets manipulés (sémantique)

Pourquoi Modelica (2/2)

- Modelica est issu d'une communauté de modélisateurs, familière de l'approche objet (école scandinave)

Remarque : ce n'est pas Modelica en soit qui est intéressant, mais **la démarche** des modélisateurs qui le développent : simuler, c'est programmer ⇒ recherche d'un langage de programmation adapté à leurs besoins

Modéliser / simuler avec Modelica : processus général (1/2)

1. définition d'un modèle du système
   - orienté composant
     - variables d'état discrètes et/ou continues
     - dynamique (temps continu) décrite par
       - équations différentielles
       - programmes
     - héritage entre composants
       - composants réutilisables (bibliothèques de composants)
   - modèle = assemblage de composants
     - éditeurs graphiques

Modéliser / simuler avec Modelica : processus général (2/2)

2. définition des paramètres d'une simulation
3. compilation du modèle paramétré
   - mise à plat du modèle objet
   - production d'un système d'EDA hybride
4. exécution du programme de simulation
   - exploitation de solveurs d'EDA hybrides adaptés au problème

Modelica : le langage

- Concepts clés
  - une liste de constructions de types de base
    - `Real`, `Integer`, `String`, `Boolean`
    - `Array`, `Enumeration`
  - une notion générale de classe (**`class`**)
  - des formes de classes particulières
    `model`, **`record`**, **`block`**, **`connector`**, **`type`**, **`package`**, **`function`**
  - dynamique : le temps évolue continûment. Il est partagé par tous les composants d'un modèle

Modelica : les classes

Une classe définit les propriétés communes aux composants qui y sont rattachés

- reprise de propriétés de classes existantes (héritage multiple)
  - telles quelles
  - avec spécialisation (types plus spécifiques, valuation de paramètres)

- variables d'état
  - variabilité
    - **`constant`** : valeur imposée dans le modèle (immuable pour toute simulation)
    - **`parameter`** : valeur libre devant être spécifiée en début de simulation (immuable pour cette simulation)
    - **`discrete`** : valeur réelle pouvant changer lors d'événements discrets
  - causalité
    - **`input`** : valeur devant provenir d'une variable output
    - **`output`** : valeur produite
  - flux
    - **`flow`** : variable du genre flux (voir *connecteurs*)
  - partage (modélisation de champs)
    - **`inner`** : variable à exporter vers les composants
    - **`outer`** : variable importée

- équations
  - liste d'équations liant les variables d'état
    - définissent la dynamique des variables continues
    - opérateur **`der`** de dérivation par rapport au temps
  - les équations peuvent être conditionnées
    - notion d'évènement
- algorithme
  - description d'un programme
    - permet de calculer les valeurs de variables
    - le temps du calcul n'est pas le temps de la simulation
      » simulation stoppée (survenue d'un événement) quant un algorithme est exécuté

Modelica : classes particulières

- **`type`** : définit un type, par exemple par extension d'un type prédéfini

Exemple :

```
type Voltage = Real(quantity="Voltage", unit="V");
```


- **`connector`** : définit un connecteur
  - des variables d'état sont des potentiels
    - lorsque deux connecteurs sont reliés, leur connexion impose que leurs variables potentiels soient égales
  - des variables d'état sont des flux (**`flow`**)
    - lorsque deux connecteurs sont reliés, leur connexion impose que la somme de leurs variables flux soit égale à 0
  - pas d'équation associée

Exemple :

```
connector Pin
  Voltage      v;
  flow Current i;
end Pin;
```

- **`model`** : classe ne pouvant pas être exploitée comme connecteur

Exemples :

un modèle abstrait :

```
partial model TwoPins
     Pin     p, n;
     Voltage u;
     Current i;
equation
     0 = p.i + n.i;
     u = p.v - n.v;
     i = p.i;
end TwoPins;
```

des modèles concrets :

```
model Resistor
     extends TwoPins;
     parameter Resistance R;
equation
     u = R*i;
end Resistor;

model Capacitor
     extends TwoPins;
     parameter Capacitance C;
equation
     C*der(u) = i;
end Capacitor;
```


Exemples (suite) :

```
model SinusSource
  extends   TwoPins;
  parameter Frequency F;
  parameter Voltage   A;
protected
  constant Real PI = 3.14159265358979;
equation
  u = A * sin(time * 2*PI*F);
end SinusSource;
```


- **`block`** : classe devant contenir au moins une variable causale (**`input`** ou **`output`**), non exploitable comme connecteur
- **`function`** : classe **`block`** ne contenant pas d'équation et contenant au plus un algorithme

```
function Distance
  parameter Integer N;
  input     Real[N] p1;
  input     Real[N] p2;
  output    Real    result;
algorithm
  result := 0;
  for i loop
    result := result +(p1[i]-p2[i])^2;
  end for;
  result := sqrt(result);
end Distance;
```

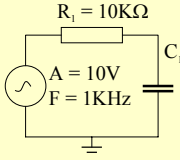

- **`record`** : classe sans équation, non exploitable comme connecteur

```
record Complex
  Real re, im;
end Complex;
```

- **`package`** : classe ne devant contenir que des définitions de classes ou de constantes

Un exemple de modèle

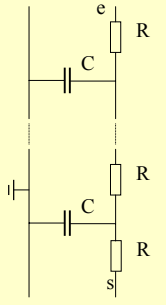
modèle de type EDO




```
model monCircuit
  Resistor    R1(R=1e4);
  Capacitor   C1(C=1e-6);
  SinusSource S(A=10, F=1000);
  Ground      G;
equation
  connect(S.p, R1.p);
  connect(R1.n, C1.p);
  connect(S.n, G);
  connect(C1.n, G);
end monCircuit;
```


Extension du même exemple

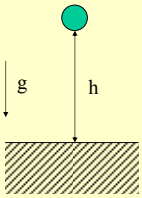
un nombre arbitraire de composants




```
model unEtageAtténuateur
  parameter Integer N;
  Pin               e, s;
  Resistor[N+1]     R;
  Capacitor[N]      C;
  Ground            G;
equation
  connect(e, R[1].p);
  for i in 1:N loop
    connect(R[i].n, R[i+1].p);
    connect(R[i].n, C[i].p);
    connect(C[i].p, G);
  end for;
  connect(R[N+1].n, s);
end unEtageAtténuateur;
```

Un autre exemple

modèle de type EDA hybride




```
model BalleRebondissante
  parameter Real c = 0.7;
  parameter Real g = 9.81;
            Real h (start=1);
            Real v;
equation
  der(h) = v;
  der(v) = -g;
  when h <= 0 then
    reinit(v, -c*pre(v));
  end when;
end BalleRebondissante;
```


Dernier exemple

modélisation d'un champ (partage de variable)

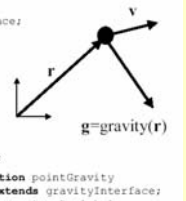
```
model unObjet
  outer Real T0;
        Real T;
equation
  T = T0;
end unObjet;
```

```
model unEnvironnement
  inner      Real T0;
  parameter Real a=1;
equation
  T0 = sin(a*time);
end unEnvironnement;
```

```
model monEnvironnement
  extends unEnvironnement(a=5);
  unObjet   o1, o2, o3;
    // o1.T = o2.T = o3.T = T0 = sin(5*time)
end monEnvironnement;
```



```
model Particle
   parameter Real m = 1;
   outer function gravity = gravityInterface;
   Real r[3](start = {1,1,0}) "position";
   Real v[3](start = {0,1,0}) "velocity";
equation
     der(r) = v;
   m*der(v) = m*gravity(r);
end Particle;

partial function gravityInterface
  input  Real r[3] "position";
  output Real g[3] "gravity acceleration";
end gravityInterface;

function uniformGravity
  extends gravityInterface;
algorithm
  g := {0, -9.81, 0};
end uniformGravity;

function pointGravity
  extends gravityInterface;
  parameter Real k=1;
protected
  Real n[3]
algorithm
  n := -r/sqrt(r*r);
  g := k/(r*r) * n;
end pointGravity;

model Composite1
  inner function gravity =
                pointGravity(k=1);
  Particle p1, p2(r(start={1,0,0}));
end Composite1;

model Composite2
  inner function gravity =
                uniformGravity;
  Particle p1, p2(v(start={0,0.9,0}));
end Composite2;
```



Outils existants

- Editeurs graphiques libres
- Bibliothèques de composants dans divers domaines
  - aucun en agronomie (Cirad : ☹)
- Compilateurs
  - commerciaux pour l'instant
  - un projet *open-source* en cours
    - OpenModelica

Informations utiles

- Source principale

  `http://www.Modelica.org`
- Abréviations
  - EDA = Equation Différentielle et Algébrique

    DAE = Differential and Algebraic Equation
  - EDO = Equation Différentielle Ordinaire

    ODE = Ordinary Differential Equation
  - EDP = Equation aux Dérivées Partielles

    PDE = Partial Differential Equation

Travaux similaires

(non exhaustif...)

- Simulation
  - Scilab - Scicos (Inria)
    - libre et gratuit
  - gPROMS (Imperial College)
- Vérification
  - HyTech (Berkeley)

Conclusions sur Modelica (1/2)

- Nombreuses extensions en cours
  - Etendre le langage aux EDP (PDEModelica)
    - solveurs type *Eléments Finis*
    - classe pour décrire la géométrie du domaine
      - exploitée pour construire le maillage
    - expression des conditions aux bords du domaine
    - ajout d'opérateurs de dérivation
  - Systèmes à structure variable
- Langage permettant de spécifier des séries de simulations (expériences)
  - Calibration
  - Tests de sensibilité

Conclusions sur Modelica (2/2)

- Avancées sur le plan algorithmique
  - Etude des systèmes hybrides (automates hybrides)
  - Caractériser des classes de modèles décidables
    - Sûreté (prouver qu'un état ne sera jamais atteint)
    - Contrôlabilité (prouver qu'un état pourra toujours être atteint à partir d'un état donné)
  - Définition de solveurs à large spectre
    - outils d'analyse et transformation automatique d'une EDA afin de déterminer de quelle famille d'équations elle relève, et donc du meilleur solveur associé

Plan (avancement)

- Présentation du contexte
- Modélisation de systèmes hybride
  - Les automates hybrides
  - Un langage de modélisation : Modelica
- Programmation de machines massivement parallèles
  - Quels liens avec la modélisation de systèmes
  - Quelques travaux exemplaires
- Conclusion

Programmer des machines massivement parallèles

- Quels liens avec la modélisation de systèmes
  - contrôle de processus concurrents, qui interagissent
  - espace = réseau d'interaction
- Penser un programme comme le contrôle d'un processus physique
  - Fondamentalement, d'un point de vue Physique :
    - un programme est un champ opérant sur la mémoire
    - exécuter un programme, c'est développer la trajectoire d'un point (état mémoire) soumis à ce champ
      - donnée d'entrée = début de la trajectoire
      - résultat = point fixe de la trajectoire (s'il existe)

Programmer des machines massivement parallèles

- Question : comment programmer une machine
  - composée d'un très grand nombre d'unités de calcul (PU)
  - PU connectées par un réseau dont la topologie n'est pas nécessairement homogène, aux comportements asynchrones
  - des connexions peuvent être créées ou supprimées en cours de calcul
  - des PU peuvent être retirées ou ajoutées au réseau en cours de calcul (PU non fiables, altérations du réseau)
- Réponse : il n'y en a aucune définitive à ce jour
  - plusieurs voies de recherche ouvertes
  - aucune n'a pris le pas sur les autres
  - bref, pour l'instant, tout le monde patauge ;-)

Quelques pistes suivies

1. Ignorer toute dimension parallèle
   - Faire abstraction du parallélisme intrinsèque du problème et de celui de la machine
     - Haskell
       - rester dans un cadre de programmation classique (ici cadre fonctionnel) : le compilateur est chargé d'exploiter au mieux les ressources de calcul

Quelques pistes suivies

2. Ignorer les contraintes matérielles
   - le langage permet d'expliciter des propriétés relevant du parallélisme intrinsèque du problème, mais l'architecture de la machine cible reste inconnue
     - tout langage type machine chimique : Gamma, JoCaML, etc.
     - MPI (Message Passing Interface) ou PVM
       - programmer = gérer des échanges de messages (notion de processus explicite ou non)

Quelques pistes suivies

3. Tenir compte de tout, explicitement
   - La machine est parallèle, et son architecture est prise en compte dans le code
     - Occam
       - programmer = recevoir et émettre des données sur les canaux de communication associés à chaque unité de calcul (Transputer).

Quid des abstractions du parallélisme

- Quelles sont les structures adaptées
  - structures de données, structures de contrôle
  - objectif :
    repenser en profondeur la programmation des machines, et donc notre façon de percevoir (modéliser) le monde…
- modèles de calcul classiques
  - CSP de Hoare, CCS ou π-calcul de Milner
- métaphore chimique (L.M. Adleman)
  - la machine chimique abstraite (ChAM) de Berry et Boudol ; le join-calculus ; le calcul bleu
- métaphore biologique (cellulaire)
  - les systèmes membranaires de Paun ; la machine BLOB de Gruau ; le calcul des Ambients de Cardelli

La machine chimique abstraite

The CHemical Abstract Machine (ChAM) - publié en 1990, par G.Berry et G. Boudol.

- Présentation
  - Mémoire = une **solution**, i.e. un multi-ensemble de **molécules** : $S = \{| m_1, m_2, ..., m_k |\}$
    - une **molécule** est un terme algébrique.
  - Programme = des **règles de réaction**
    - **règle de réaction** : consomme des molécules pour en produire d'autres
      - Règles structurales (réversibles)
        - ➢ $S \leftrightarrow S'$ (où $\rightarrow$ = échauffement, $\leftarrow$ = refroidissement)
      - Règles de réduction (irréversibles)
        - ➢ $S \rightarrow S'$

La machine chimique abstraite

- Sémantique

  soient S, S' et S'' trois solutions, et $C[T]$ une molécule paramétrée par un terme T.
  - loi chimique
    - si $S \Rightarrow S'$, alors $S \oplus S'' \Rightarrow S' \oplus S''$
  - loi de membrane
    - si $S \Rightarrow S'$, alors $\{| C[S] |\} \Rightarrow \{| C[S'] |\}$
- Réalisation
  - Seule difficulté : il faut garantir que si une réaction est possible, elle aura bien lieu.

ChAM : le langage Gamma

GAMMA = General Abstract Model for Multiset mAnipulation.

Publié par J.P. Banâtre et D. Le Metayer (1988) ; a fortement inspiré les auteurs de la ChAM.

Exemples

```
max :   replace x, y by y when x<= y

sort :  replace (i, x), (j, y) by (i, y), (j, x)
          when i > j and x <y

iota :  replace (x, y) by (x, (x+y)/2), ((x+y)/2+1, y)
          when x < y
        replace (x, y) by x when x = y
sieve : replace x, y by y when x%y==0
primes(n) = sieve(iota( (2, n) ))
```

Les systèmes de Paun

- Pour résumer
  - Système de Paun = machine chimique + membranes solubles
    - regroupement de molécules via des **membranes**, sachant que :
      - membranes imperméables (une molécule ne peut pas changer de membrane)
      - membranes solubles (réaction spéciale, avec une molécule particulière)
      - il existe une membrane encapsulant toutes les autres, non soluble
- Sémantique
  - A l'initialisation, un système membranaire est donné :
    - une hiérarchie de membranes (toute intersection de deux membranes A et B est soit vide, soit égale à A ou B ⇒ le graphe d'inclusion est un arbre)
    - des molécules sont placées dans chaque membrane
  - Règles de réaction appliquées jusqu'à saturation

Les travaux de l'équipe de J.L. Giavitto (LaMI, Evry)

Crédo : décrire des calculs dans un espace, en n'autorisant que des accès locaux (déplacements spatiaux relatifs)

- Langage 8 ½
  - programmer = définir des équations sur des tissus.
    - un tissu est une collection (agrégat spatial) de flots (agrégats temporels) de valeurs
- MGS, une évolution de 8 ½
  - programmer = définir des équations sur des objets algébriques
    - des opérations algébriques dénotent des relations spatio-temporelles.

Les travaux de l'équipe de J.L. Giavitto (LaMI, Evry)

- 1988 : langage de programmation parallèle 8 ½
  - exploiter de bonnes propriétés pour faciliter sa compilation sur des machines massivement parallèles
- 1991 : [constat] principal demandeur = monde de la simulation ; structures de données fondamentales (DF *[data field]*, GBF *[group based field]*)
  - principaux demandeurs et bailleurs de fonds : biologie, agronomie, physique fondamentale, …
- 1997 : concilier les deux mondes informatique / simulation : MGS
  - slogan : approche formelle capable d'appréhender les deux mondes = topologie algébrique (combinatoire)

Le langage 8 ½

- Les entités manipulées : trois concepts clés
  - flot (stream) = structuration du temps
  - collection = structuration de l'espace
  - tissu (web) = composition de flots et de collections
    - processus = programme 8 ½
- Une approche déclarative
  - programme = système d'équations sur un tissu
    - cadre fonctionnel pur
    - langage typé statiquement, types inférés
- Compilateur
  - place et séquence les processus
- Des contraintes fortes
  - les collections et les flots sont homogènes
  - évolution synchrone (horloge globale)

Les flots (*stream*)

- **Définition**
  - Flot = variable qui change de valeur dans le temps

    ⇒ suite de valeurs

    Les flots partagent une même référence temporelle (tics)

    Une valeur ne dépend que d'un nombre fini de valeurs passées, à horizon temporel borné
  - Autrement dit, un flot F est caractérisé par 2 fonctions :

$$F = <\ 0;\quad 1;\quad 2\quad <$$
$$\text{horloge}(F) = H_F = <\ v;\quad f;\quad v;\quad f;\quad v;\quad f <$$
$$\text{valeur}(F) = V_F = <\ 0;\quad 0;\quad 1;\quad 1;\quad 2;\quad 2 <$$

  - top = tic pour lequel l'horloge est vraie (v)

Les flots

- Flots constants
  - Tout scalaire constant est un flot :

$$12 = <\ 12\quad <$$
$$\text{horloge}(12) = H_{12} = <\ v;\quad f;\quad f;\quad \ldots <$$
$$\text{valeur}(12) = V_{12} = <\ 12;\quad 12;\quad 12;\quad \ldots <$$

  - **`Clock`** : flot de synchronisation :

$$\text{horloge}(\texttt{Clock}) = <\ v;\quad v;\quad v;\quad \ldots <$$
$$\text{valeur}(\texttt{Clock}) = <\ v;\quad v;\quad v;\quad \ldots <$$

Les flots : opérations principales

Illustrées sur des exemples ; soient trois flots

Tic⇒	0	1	2	3	4	5	6	7	8	9	10
X		2	3		2		6		7		8
Y	8		7		5	6	1	1	4		
A	v		f	f	v	v	f		v		f

- Extension des opérations classiques

Tic⇒	0	1	2	3	4	5	6	7	8	9	10
X+Y	⊥	10	10		7	8	7	7	11		12

- Retard

Tic⇒	0	1	2	3	4	5	6	7	8	9	10
$X			2		3		2		6		7

- Echantillonnage

Tic⇒	0	1	2	3	4	5	6	7	8	9	10
X when A					2	2			7		

- etc…

Quelques exemples

- Equations quantifiées

`F@t` : valeur du flot `F` au top n°`t`

```
A   = 5
B   = 6 when Clock
C@0 = 1
C   = A + $B
```

- Fibonacci

```
fib@0 = 1
fib@1 = 1
fib   = $fib+$$fib when Clock
```

- Factorielle

```
n@0 = 0
n   = 1+$n when Clock

fact@0 = 1
fact   = n*$fact
```


Les collections

- Définition
  - Collection = agrégat fini de composants
    - composants tous d'un même type
    - chaque composant est repéré par un index (coordonnée dans l'espace qu'est la collection)
    - un composant est soit simple (scalaire, flot), soit une collection
- Notations

Construction

`{ 3, 7, 5, 9 }` de type `int[4]`

Projection

`{ 3, 7, 5, 9 }.2` vaut `5`

Collections : quelques opérations

- Nombreuses (à la Lisp, inspirées d'APL)…

Alpha-extension (équivalent map)

`+^ {1, 2, 3} {4, 5, 6}` vaut `{5, 7, 9}`

Beta-réduction (équivalent fold)

`+\ {1, 2, 3}` vaut `6`

Balayage

`+\\ {1, 2, 3}` vaut `{1, 3, 5}`

Composition (équivalent concat)

`{1, 2}#{3, 4}` vaut `{1, 2, 3, 4}`

Collections : quelques opérations

Sélection

`{4, 5}({1, 0, 1})` vaut `{5, 4, 5}`

Génération

`'n` vaut `{0, 1, 2, …, n-1}`

Troncature

`{1, 2}:[3]` vaut `{1, 2, 1}`

etc…

Quelques exemples

`C[g]` : spécifie explicitement la géométrie `g` de la collection `C`

- Exemple 1 : iota (APL)

Soit n donné

```
iota[n] = 0 # (iota:[n-1] + 1)
```

Si n=9, alors définit la collection :

```
iota = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }
```

- Exemple 2 : Fibonacci

Soit n donné

```
fib[n] = 1 # fib:[n-1]
       + { 0, 0 } # fib:[n-2]
```

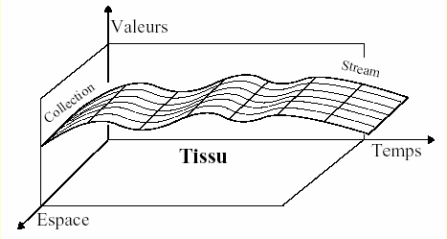
- Exemple 3 : factorielle

Soit n donné

```
fact[n] = 1#((iota+1)*fact:[n-1])
```

Les tissus

- Définition
  - Tissu = flot de collections = collection de flots



Exemple réaliste : diffusion

- Problème
  - une barre de métal fine de longueur L, tenue à ses extrémités ; en tout point x de la barre, sa température à l'instant t est $u(x, t)$.
- Question
  - étudier l'évolution de la température lorsque la barre est laissée à elle-même, connaissant la température à ses extrémités (x=0 ou x=L) et à t=0.
- Solution
  - résoudre l'équation aux dérivées partielles suivante :

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$$

Diffusion (suite)

- Exploitation d'un schéma type **différences finies** :
  - espace discrétisé : $x_i = i \,.\, h$ (h = pas spatial)
  - temps discrétisé : $t_j = j \,.\, k$ (k = pas temporel)
  - donc $u_{i,j} = u(x_i, t_j)$

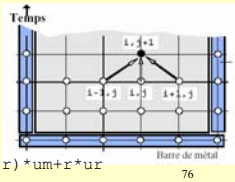
$$\frac{u_{i,j+1} - u_{i,j}}{k} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2}$$

soit $u_{i,j+1} = ru_{i-1,j} + (1-2r)u_{i,j} + ru_{i+1,j}$ en posant $r = \frac{k}{h^2}$

Diffusion (fin)

- En langage 8 ½

```
left[1]  = température en x=0, pour tout t
right[1] = température en x=L, pour tout t
u@0      = distribution température à t=0
u        = (left # V # right)
           when Clock

L        = ...    longueur de la barre
k        = ...    pas temporel
n        = 100    nombre total de points
m        = n-2    nombre total de points sauf
extr.
h        = L/n
r        = k/(h*h)

ul       = ($U)('m)
um       = ($U)('m+1)
ur       = ($U)('m+2)

V[m]     = r*ul+(1-2*r)*um+r*ur
```



Premières leçons

- Constat
  - collection = agrégat spatial homogène
    - ⇒ enlever la contrainte d'homogénéité
- Solution : notion de système
  - Système = collection hétérogène
    - les composants sont repérés par un nom
    - notion classique d'enregistrement
      tableau = produit de types tous homogènes
      enregistrement = tableau de types hétérogènes

Exemple

```
s = 5,
Machine = {
   t@0    = 0,
   t      = $t+1 when Clock,
   pièce  = 0==(t%s)
}
```

- `Machine` = système composé de deux champs :
  - `t` est un flot entier
  - `pièce` est un flot booléen
- flots accessibles via une notation pointée :
  - `Machine.t`
  - `Machine.pièce`

Et alors ?

- Permet une approche objet, via un opérateur de concaténation des systèmes (#).
  - Exemple

```
Mobile = {
     position@0 = initial,
     position   = $position+déplacement
},

TrajectoireUniforme = Mobile # {
     déplacement = vitesse when Clock
},

soleil = TrajectoireUniforme # {
     vitesse = {1, 1},
     initial = {0, 0}
}
```


Leçons à tirer (suite)

- Constat
  - collection = agrégat spatial dont la topologie est fixée, non modifiable
    - ⇒ permettre de définir sa topologie au choix
- Solution : notion de GBF
  - GBF = *Group-Based data Field*
    - généralise la notion de collection : champs de donnée (*data field*)
      - agrégat spatial homogène
      - les index forment un groupe (notion de **déplacement**)
        - » un index est appelé **forme**

GBF et champs de données

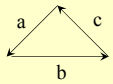
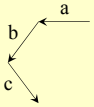
Généralisation de la notion de tableau

- *array* [tableau]
  - tableau = fonction totale : $I_1 \times \ldots \times I_n \rightarrow V$
    associe à tout indice $(i_1, \ldots, i_n)$ une valeur
- DF = *data field* [champ de données]
  - DF = fonction partielle : $\mathbb{Z}^n \rightarrow V$
- GBF = *group-based data field*
  - GBF = fonction partielle : $P \rightarrow V$
    - P ensemble infini énumérable (*formes*)
    - muni d'une topologie (groupe des déplacements)
  - idée générale :
    - à partir d'un point de référence…
    - …atteindre les autres via des déplacements élémentaires

GBF – les formes

- Propriétés du groupe des formes P
  - le groupe $(P, \bullet)$ doit admettre une présentation finie (G, E)
    G = un ensemble fini de **générateurs**
    E = un nombre fini d'**équations** du type $w = w'$
    où w et w' sont des compositions par $\bullet$ d'éléments de G ou de leurs inverses
  - Notations :
    0 = élément neutre (déplacement nul)
    $-x$ = inverse de x
    $x^k = x\bullet\ldots\bullet x$ (k fois)

GBF – quelques topologies

maillage triangulaire	maillage carré	maillage hexagonal
		
$a^2 = b^2 = c^2 = 0$ $(a\bullet b\bullet c)^2 = 0$	$a\bullet b = b\bullet a$	$b = a\bullet c$

GBF - groupe et graphe de Cayley

- si $(P, \bullet)$ est le groupe des formes, alors il est possible de construire un graphe (P, A) :
  avec $A \subseteq P \times B$ tel que :
  - $B \subseteq P$
  - $(x, y) \in A \Leftrightarrow$ il existe z tel que $x\bullet z = y$
- si B = ensemble des générateurs de la présentation finie (appelé aussi *base*), alors ce graphe est appelé *graphe de Cayley* du groupe.

Programmer avec des GBF

- soient F est un GBF et P une forme (un point)
  - hypothèse : la valeur de F en P dépend, selon f, des valeurs voisines à P, quelque soit P ; autrement dit :

    $F(P) = f(F(P \bullet g_1), F(P \bullet g_2), \ldots, F(P \bullet g_n))$

    - où g_i = générateur du groupe
    - et f fonction quelconque d'arité n
- si F opère en tous les points de l'espace E, on note :

  $F[E] = f(F \bullet g_1, F \bullet g_2, \ldots, F \bullet g_n)$
- autrement dit, un programme s'exprime :
  - sans référence absolue aux points
  - seulement en fonction de déplacements locaux (générateurs)

GBF – leurs opérations

- Toutes les opérations définies sur les collections ont été étendues aux GBF
  - constructions de GBF à partir de GBF existants
    - qui modifient les valeurs
    - qui modifient la géométrie
  - opérations algébriques classiques (+, -, …)
  - opérations agrégatives (réductions)

Leçons à tirer (suite)

- Constat
  - système = agrégat spatial hétérogène dont la géométrie est invariable

    ⇒ permettre de modifier la géométrie (dans une certaine mesure)
- Solution : notion d'Amalgame
  - généralise et systématise la notion de système

Les amalgames

- Principales opérations
  - Amalgamation

    `A = { n1 = e1, …, nk = ek }`

    les e_i peuvent être des amalgames
  - Concaténation

    `A = A1 # A2`
  - Sélection

    `A.ni = ei`

Le projet MGS

MGS = un **M**odèle **G**énéral pour la **S**imulation

- Synthèse
  - tout programme 8½ (avec ou sans extension) se ramène au schéma de calcul suivant :

    répéter
    - extraire d'une collection C une sous-collection A
    - calculer à partir de A une nouvelle sous-collection B
    - remplacer dans C la sous-collection A par B

    jusqu'à point fixe
- Outillage théorique : la topologie algébrique
  - définit formellement les notions de voisinage, de bord, de recollage (remplacement)
  - généralise toutes les structures de données connues en informatique, y compris celles de 8½ (complexe simplicial)

MGS en quelques mots

- Reprise des idées de la ChAM
  - dans une réaction chimique

    si x , y alors z

    la notation x , y spécifie que les deux molécules sont distinctes, mais qu'en plus elles sont proches (au sens d'une topologie liée au problème)
- structure de monoïde, avec l'opérateur ,

MGS : développements actuels

- un langage de programmation
  - fonctionnel typé
  - structures de données étudiées pour être traitées efficacement sur une machine massivement parallèle
- structures intégrées :
  - multi-ensembles
    - opérateur , commutatif
  - ensembles
    - opérateur , commutatif et idempotent
  - séquences
    - opérateur , non commutatif

MGS : conclusion

- approche novatrice
  - très en marge des écoles de pensée classiques (graphes, variantes λ-calcul, objets, agents, etc.)
  - outillage mathématique très élaboré : topologie algébrique
- topologie algébrique
  - cadre puissant de formalisation de nombreux concepts en informatique
    - premiers résultats encourageants
      - descriptions multi-échelle (à résolutions multiples)
      - preuves de programmes parallèles
      - nouvelles structures de données (GBF)
    - approche géométrique de la notion de calcul
  - exploitée dans d'autres disciplines
    - Physique fondamentale (cosmologie)

Conclusion générale

Modélisation des systèmes dynamiques

et

Programmation des machines massivement parallèles

Les mêmes fondamentaux :

- Problèmes de représentation
  - du temps
  - de l'espace
  - de l'énergie (états)
  - des processus, donc de leurs interactions
- Problèmes de contrôle
  - exploitation optimale d'une machine ou processus naturel pour satisfaire un but

Conclusion générale

- Trouver de nouvelles voies
  - modèles de calcul parallèles
    - machine auto-développante (Blob) de Gruau

    ⇒ contrôle absolu du développement spatial de la machine
    - machine amorphe

    ⇒ aucun contrôle du développement spatial
  - langages de programmation adaptés