LIRMM – projet D'OC

Phil. REITZ (reitz@lirmm.fr)

27 octobre 2003

Modélisation de processus dans un contexte spatialisé

ou

variations sur quelques problèmes de modélisation à des fins de simulation au Cirad

Plan

Contexte : Cirad, outils de modélisation
Synthèse de quelques difficultés à surmonter
Quelles pistes suivre
  Modéliser des processus…
    ⇒ les automates hybrides
    ⇒ les équations différentielles et algébriques hybrides
  …dans un contexte spatialisé
    ⇒ le projet MGS
Conclusion

Contexte : agronomie et Cirad

Modélisation – simulation au Cirad

- 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

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 processus

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 = une équation différentielle ordinaire (EDO [*ODE*])

$$\dot{x} = f(x,t)$$

- Espace explicite *[continu]*

Modèle = équation aux dérivées partielles (EDP [*PDE*]) $\Rightarrow$ différentiation par rapport à toutes les dimensions : temps, espace.

Modélisation dans un cadre discret

E = espace des états (phases) *[discret]*

T = temps *[discret]*

S = espace *[discret]*

Modèle = automate calculant l'état suivant :

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

$\Rightarrow$ processus – automates de Markov

Analyse des principales difficultés

Liste non exhaustive (doux euphémisme ☺)

Modélisation

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

Simulation

Langage de représentation

- Equations différentielles ⇒ solveurs
- Représentation par objets

Outils d'analyse

- Etude de sensibilité
- Argumentation des résultats
- Optimisation de paramètres
- Gestion des données de simulation

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.

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

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)

Présentation rapide

(présentation plus 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 de phases instantanées.

Les automates hybrides

Le modèle

état = (q, x, t) avec :

- q = état discret (pris dans un ensemble fini discret Q)
- x = état continu (pris dans un ensemble infini continu E)
- t = temps (pris dans un ensemble infini continu T)

loi de transition, décomposée en 4 fonctions :

- $f(q, x, \dot{x}, t) = 0$: équation différentielle régissant l'évolution de l'état continu en phase stable
- $g(q, x, \dot{x}, t) = 1$ ou 0 : prédicat détectant une transition d'état
- $\delta(q, x, \dot{x}, t) = q'$: fonction régissant la transition de l'état discret
- $i(q, x, \dot{x}, t) = x'$: fonction régissant la transition de l'état continu (réinitialisation)

Les automates hybrides

Sémantique

état courant = (q, x, τ)

phase stable

Phase stable durant $(\tau, \tau') \Leftrightarrow \forall t \in [\tau, \tau'[, g(q, x, \dot{x}, t) = 0$

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

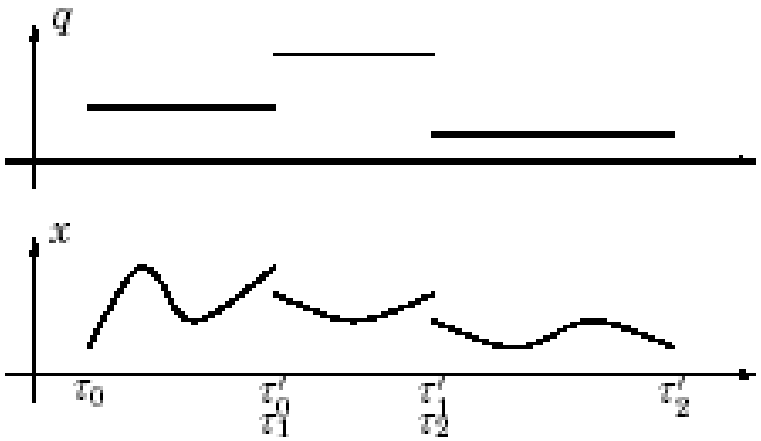
phase de transition

Transition de phase à l'instant $t \Leftrightarrow g(q, x, \dot{x}, t) = 1$

Lorsqu'une transition de phase est détectée, alors :

- l'état discret de l'automate passe à $\delta(q, x, \dot{x}, t)$
- l'état continu x est réinitialisé à $i(q, x, \dot{x}, t)$

soit en image…



Les automates hybrides

Exemples

Une balle rebondissante

Position de la balle : $x > 0$

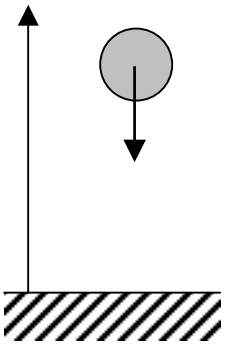
$\Rightarrow$ état continu = $(x, v = \dot{x})$

Accélération : $g > 0$

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

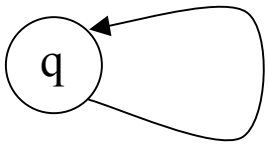
équation différentielle en q :

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



condition de la transition :

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

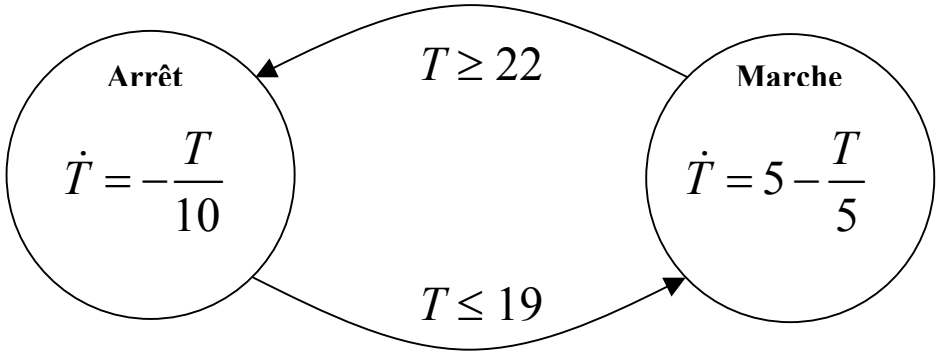


réinitialisation :

$v := -cv$

Un thermostat

Température de la pièce = T



Les automates hybrides

Principaux résultats

Dans le cas général, ils sont tous dramatiques (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.

EDA

Une EDA est de la forme :

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

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).

EDA hybride

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

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

- un langage de modélisation s'appuyant sur des EDA hybrides : Modelica

Le projet Modelica

Références

http://www.Modelica.org

Présentation

MODELICA = définition d'un langage de modélisation normalisé (spécifications de la syntaxe et de la sémantique) :

- langage à objets
- modélisation des processus via des EDA hybrides

Concrètement

- initiative internationale, très nord-européenne jusqu'ici, commencée en 2000
- aucun outil n'est disponible sous forme libre (à ce jour, mais projets en cours).

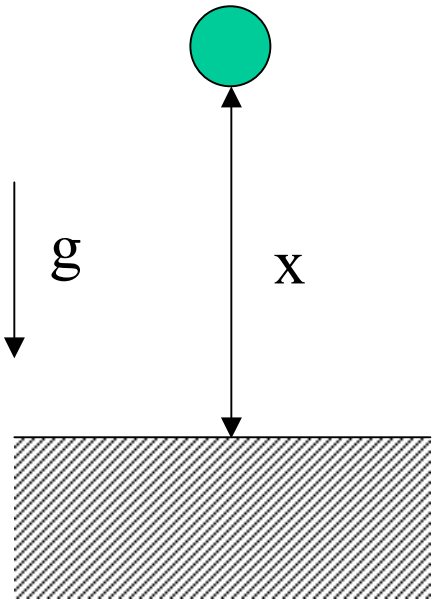
Principe général de compilation

- mise à plat du modèle objet $\Rightarrow$ transformation en une EDA hybride
- génération de l'automate hybride associé
- l'exécution s'appuie sur un solveur d'EDA idoine.

Modelica

Exemple 1 : balle rebondissante

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



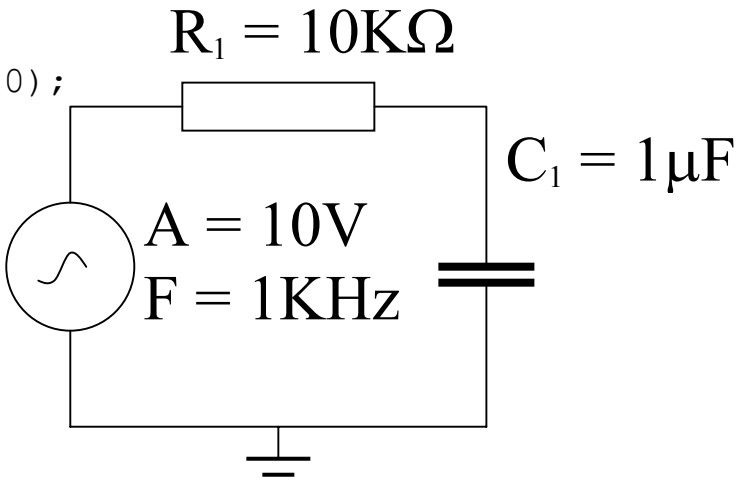
Modelica

Exemple 2 : les circuits électroniques

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

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

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;

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;

model SinusSource
    extends   TwoPins;
    parameter Frequency F;
    parameter Voltage   A;
protected
    constant Real PI = 3.1415927;
equation
    u = A * sin(time * 2*PI*F);
end SinusSource;
```

```
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;
```



Premières leçons à tirer

Modélisation de processus

- modèles hybrides : une synthèse réussie des approches continues et discrètes
- défaut principal : aucune prise en compte explicite de l'espace
  - ➢ premiers papiers de la communauté Modelica sur l'expression de contraintes spatiales : 2002

Intégrer la dimension spatiale

- EDP (ex : éléments finis) $\Rightarrow$ ex : $\frac{\partial T}{\partial t} = k \frac{\partial^2 T}{\partial x^2}$
- SIG – graphes multi-échelle – complexes simpliciaux
- langages de programmation parallèle
  - ➢ mémoire = espace ayant une certaine topologie
  - ➢ programme = composition de processus ; chaque processus agit localement sur la mémoire

Programmation des machines parallèles

- Les automates cellulaires et affiliés
- Les langages historiques à règles de récriture (ChAM, P-Systems, L-Systems)
- Introduction explicite de la topologie des structures de données dans les langages : MGS

Le projet MGS

Références : J.L. Giavitto (LaMI)

Historique

Thèse sur le langage 8,5 en 1991

Programme = spécification d'une contrainte sur une collection (espace) de flots (temps).

Pas de référence à des positions absolues (temps, espace) : accès aux voisins d'un point via des compositions de déplacements élémentaires.

Période 1992 à 2000

- Généralisation des concepts de collection et de flot à celui de champ de données (DF)
  - Tableau = fonction totale de $I_1 \times ... \times I_n \to V$,

    où I_k intervalle de $\mathbb{Z}$

    $\Rightarrow$ *accès à une composante via un indice absolu*
  - Champ de donnée = fonction partielle de $\mathbb{Z}^n \to V$
- Focalisation sur les champs de données basés sur des groupes (GBF)
  - GBF = fonction partielle de $G \to V$, où G muni d'une structure de groupe (déplacements)

$\Rightarrow$ *accès à une composante via un déplacement par rapport à une composante de référence, jamais désignée explicitement*

A partir de 2001 : MGS

MGS

Intérêt majeur

cadre unificateur de nombreuses approches connues :

- Automates cellulaires
- Systèmes parallèles de récriture : ChAM, P et L-Systems, flots de données, etc.

Principe général

- une **collection** est un espace ayant une certaine topologie (paramétrable)
  - il doit être possible d'identifier des **sous-collections**
  - chaque sous-collection est délimitée par un **bord** dans la collection
- une **transformation** remplace une sous-collection par une autre dont le bord est **compatible** avec celui de la première
- une transformation est décrite par un ensemble de **règles**
  - si *motif sous-collection*
    tel que *condition*
    alors *sous-collection de substitution*
- un **calcul** sur une collection est une **application itérée** de transformations sur cette collection

MGS

Collections de base (monoïdales)

`a,b` relation binaire associative (`a` *voisin de* `b`)

- sans propriété additionnelle ⇒ **séquence**
- si `.,.` commutative ⇒ **multi-ensemble**
- si `.,.` commutative et idempotente ⇒ **ensemble**

Autres collections

- tableau
  - une dimension ⇒ groupe ayant 1 déplacement *g* (gauche) ; `a,b` si `a` voisin gauche de `b`
  - *n* dimensions ⇒ 1 déplacement g_i (gauche) par dimension ; `a,b` si `a` voisin gauche de `b` selon n'importe quelle dimension ;
    `a|`g_i`>b` si `a` voisin gauche de `b` selon la dimension *i*
- GBF
  - groupe des déplacements décrit par une présentation finie ; `a,b` si `b` accessible depuis `a` par un déplacement élémentaire
- graphe
  - `a,b` si les nœuds `a` et `b` sont liés par un arc

MGS

Quelques exemples

Programmation type machine chimique

```
T = {
   (x, y / x<=y) => x
}

X = Multiset(4, 3, 3, 2, 5, 1, 4, 2, 5)

T['fixpoint](X)
```

Spécification de sous-collections

Langage du type expressions régulières :

```
(_,+ as s / cardinal(s)<10 & fold[+](s)>=5) => …
```

Avancement

- Prototype de compilateur MGS disponible
- Recherche des bons outils algébriques pour analyser ce genre de système de programmation
  - piste suivie : la topologie algébrique (*bords*, *collages*), en particulier les complexes simpliciaux.
  - difficulté principale : bagage théorique hors de la formation standard d'un informaticien ⇒ long chemin d'appropriation d'outils non triviaux ☹

Conclusion

Intégration espace – temps

Prise en compte explicite des relations spatiales et temporelles dans les langages ⇒ nouvelles structures de données à définir

Spécification des processus dans un style déclaratif (type équationnel)

A plus long terme

Nous (informaticiens) sommes encore à la préhistoire de ce type d'analyse :

- en Physique : les tenseurs permettent de s'affranchir des particularités de l'espace-temps sous-jacent (toute l'information est concentrée dans le tenseur de courbure en Relativité Générale, par exemple)
  - calculer sans connaître la topologie sous-jacente : *amorphous computing*
- en Physique (toujours) : trajectoire d'un point dans l'espace-temps (souvent) définie par le champ auquel il est soumis (gravitation, électromagnétique, etc.) ; le concept de champ est semblable à celui de programme en informatique
  - Informatique et Physique partagent de nombreux concepts fondamentaux : espace, temps, énergie (état, complexité)
  - Quels outils pour rendre compte des interactions : une masse qui se déplace modifie le champ de gravitation, lequel influe sur sa trajectoire.

Peut-être nous faut-il développer (adapter ?) ce même type d'outillage mathématique pour l'informatique