Spatial Computing in MGS

Lecture III – MGS & Applications

Antoine Spicher[1] & Olivier Michel[1] & Jean-Louis Giavitto[2]

mgs.spatial-computing.org/
www.spatial-computing.org/mgs:tutorial

[1]LACL, University Paris Est Créteil

[2]IRCAM, CNRS – UPMC

UCNC'12 – Sept. 2012

Outline

- MGS Rule Application Strategies
- “Last But Not Least” Example

Rule Application Strategy

- ## MGS Pattern Matching Process

Computation of the set of *all the sub-collections* matching a pattern

$$\frac{\partial\, \mathrm{dir}*}{\partial p}(G,K,\emptyset) = \{[]\} \tag{1}$$

$$\frac{\partial P}{\partial p}(G,K,\emptyset) = \emptyset \quad \text{provided that } P \neq \mathrm{dir}* \tag{2}$$

$$\frac{\partial\, \mathrm{id}/\mathrm{expr}}{\partial p}(G,K,H) = \text{if } \mathrm{eval}(K+[\mathrm{id}\to p], G, \mathrm{expr}) \text{ then } \{[p]\} \text{ else } \emptyset \tag{3}$$

$$\frac{\partial\, \mathrm{dir}*}{\partial p}(G,K,H) = \{[]\} \cup \frac{\partial(\mathrm{id}/\mathrm{true}\ \mathrm{dir}\ \mathrm{dir}*)}{\partial p}(G,K,H) \quad \text{where id is a fresh variable} \tag{4}$$

$$\begin{aligned}\frac{\partial\, \mathrm{id}/\mathrm{expr}\ \mathrm{dir}\ P}{\partial p}(G,K,H) = {} & \text{let } K' = K+[\mathrm{id}\to p] \text{ and } H' = H-p \\ & \text{in if } \mathrm{eval}(K',G,\mathrm{expr}) \\ & \quad \text{then } p \oplus \left(\bigsqcup_{p' \in \mathrm{neighbor}(H',\mathrm{dir},p)} \frac{\partial P}{\partial p'}(G,K',H') \right) \\ & \quad \text{else } \emptyset \end{aligned} \tag{5}$$

$$\begin{aligned}\frac{\partial\, \mathrm{dir}*\ \mathrm{dir}'\ P}{\partial p}(G,K,H) = {} & \bigsqcup_{p' \in \mathrm{neighbor}(H,\mathrm{dir}',p)} \left(\frac{\partial P}{\partial p'}(G,K,H) \right) \quad \text{where id is a fresh variable} \\ & \cup \frac{\partial(\mathrm{id}/\mathrm{true}\ \mathrm{dir}\ \mathrm{dir}*\ \mathrm{dir}'\ P)}{\partial p}(G,K,H) \end{aligned} \tag{6}$$

Inspired by the *Brzozowski's derivation* of rational expressions

Rule Application Strategy

- Rewriting of non-intersecting sub-collections
  - Role of the rule application strategy
  - Hard-coded MGS strategies
    - Maximal-Parallel (no more matched sub-collection in the remaining sub-coll.)
      - `Default`: priority given to the first rules over the last ones
      - `SingleStochastic`: randomly chosen between rules
      - `MultiStochastic`: no priority between rules
    - Sequential strategies (only one rule is applied at each application)
      - `Stochastic`: random choice of the rule w.r.t. a given probability
      - `Gillespie`-based: random choice of the rule w.r.t. a given *kinetics*
        - inspired by the chemical *stochastic simulation algorithm* of Gillespie
        - only allowed for constant patterns on complete graph topology
      - `Sooner` strategy: the sooner rule is chosen w.r.t. a given *date*

Outline

- MGS Rule Application Strategies
- “Last But Not Least” Example

(Unconventional) Computation vs. MGS

- MGS programming of a *model of computation*
  - **Topological collection type** modeling the *used data structure*
  - **Specific kind of transformation rules** specifying the *computation rules*
- Examples
  - *L systems*

    Sequence & MGS rules encoding the grammar productions
  - *Chemical computations (Gamma, CHAM)*

    Bag/set & MGS rules encoding the chemical interactions
  - *P systems*

    Nested bag/set & MGS rules encoding transports and chemical interactions
  - *Cellular automata*

    GBF collection (regular space) & MGS rules encoding the local evolution function
  - *Signal Machines ??*

    Sequence of signals & MGS rules encoding the collision rules

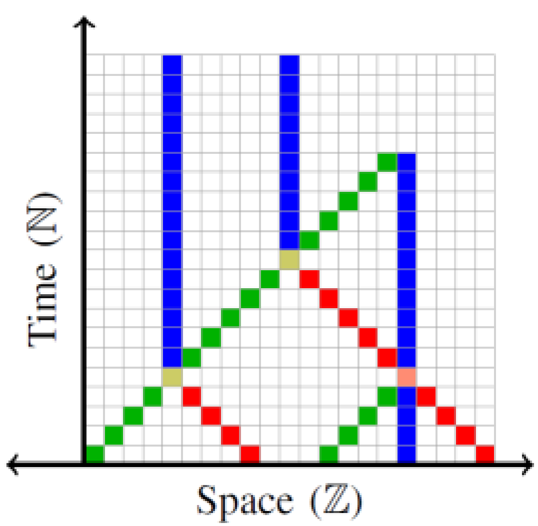
Signal Machines in MGS

- Source used for this example

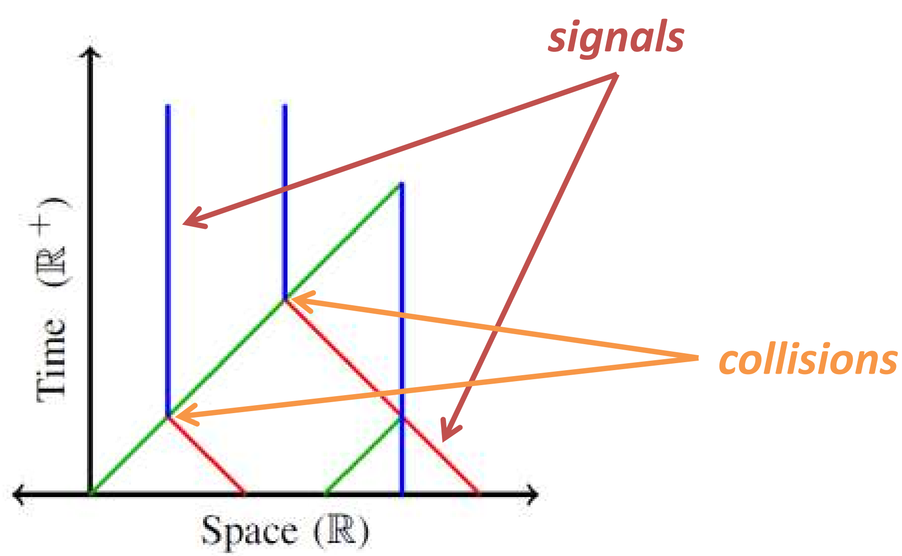
  Massively Parallel Automata in Euclidean Space-Time

  D. Duchier, J. Durant-Lose, M. Senot, SCW'10, Budapest

- Signal Machines
  - Extension of CA into continuous space and time
  - Space/time diagrams, *signals* and *collisions*



=>

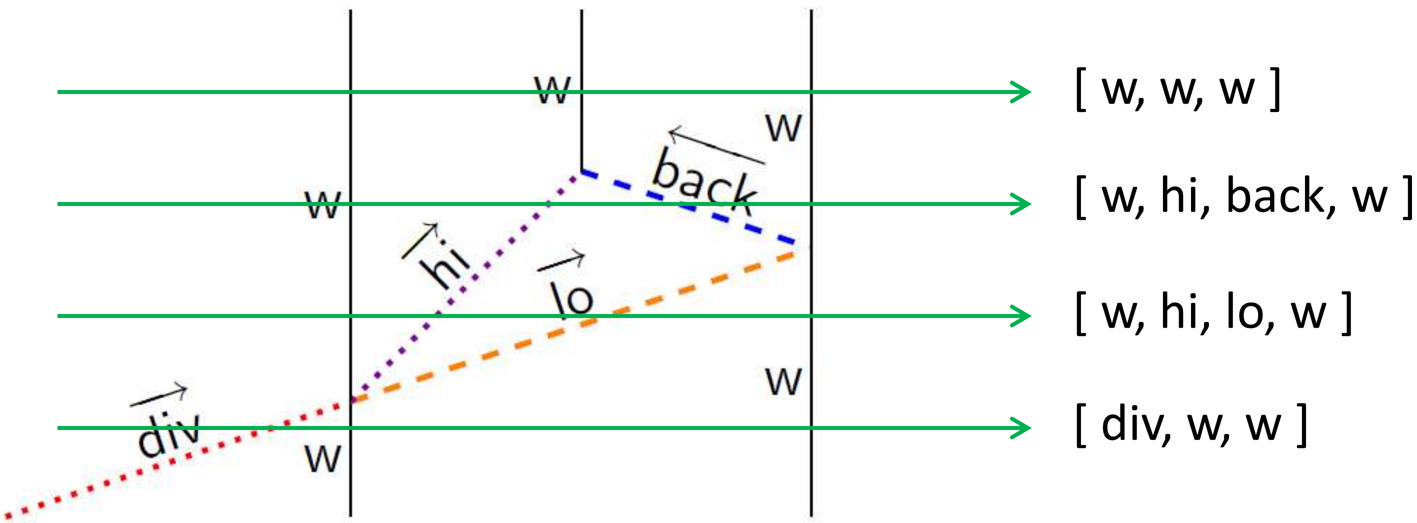


Signal Machines in MGS

- Example of a Signal Machine
  - *“Geometrically computing the middle”*

Meta-Signals	Speed
w	0
$\overrightarrow{\mathrm{div}}$, $\overrightarrow{\mathrm{lo}}$	3
$\overrightarrow{\mathrm{hi}}$	1
$\overleftarrow{\mathrm{back}}$	-3

Collision rules
$\{ \mathrm{w}, \overrightarrow{\mathrm{div}} \} \rightarrow \{ \mathrm{w}, \overrightarrow{\mathrm{hi}}, \overrightarrow{\mathrm{lo}} \}$
$\{ \overrightarrow{\mathrm{lo}}, \mathrm{w} \} \rightarrow \{ \overleftarrow{\mathrm{back}}, \mathrm{w} \}$
$\{ \overrightarrow{\mathrm{hi}}, \overleftarrow{\mathrm{back}} \} \rightarrow \{ \mathrm{w} \}$



Signal Machines in MGS

- Example of a Signal Machine
  - MGS Collection Type (a sequence of signal)

```
record     metasignal    = { name:symbol, speed:float }   and
record     location      = { position:float, date:float } and
record     signal        = metasignal + location          and
collection machine_state = [signal]seq ;;
```

  - Signal Machine Collision Specification (a transformation rule)

```
s1:signal, s2:signal / (s1.speed > s2.speed)
           ={ D = signal_intersection(s1,s2).date }=>
                    let loc = signal_intersection(s1,s2) in
                      map( make_signal(loc), collision(s1,s2) )
```

  - Middle Computation Specification

```
w    := { name = `w,    speed =  0 }
div  := { name = `div,  speed =  3 }
hi   := { name = `hi,   speed =  1 }
lo   := { name = `lo,   speed =  3 }
back := { name = `back, speed = -3 }
```

```
fun collisions(s1,s2) =
  switch (s1.name,s2.name)
    case (`div,`w):     (w,hi,lo)
    case (`lo, `w):     (back,w)
    case (`hi, `back):  (w)
    default:            (s2,s1)
```