Engineering Confluent Computational Fields

from Functions to Rewrite Rules

Mirko Viroli

Dipartimento di Informatica: Scienza e Ingegneria (DISI)
Alma Mater Studiorum—Università di Bologna, Italy
mirko.viroli@unibo.it

Spatial Computing Workshop, St. Paul (USA), 6/5/2013

Computational fields

A way to engineer self-organising applications in different scenarios

- Sensor networks (aggregating/diffusing data)[a]
- Swarm and modular robotics (cooperative team work)[b]
- Pervasive computing (crowd steering, coordinated visualisation)[c]

---

[a] Jacob Beal and Jonathan Bachrach. "Infrastructure for Engineered Emergence on Sensor/Actuator Networks". In: *IEEE Intelligent Systems* 21.2 (2006).

[b] Jonathan Bachrach, Jacob Beal, and James McLurkin. "Composable continuous-space programs for robotic swarms". In: *Neural Computing and Applications* 19.6 (2010).

[c] Mirko Viroli, Matteo Casadei, Sara Montagna, and Franco Zambonelli. "Spatial Coordination of Pervasive Services through Chemical-inspired Tuple Spaces". In: *ACM Transactions on Autonomous and Adaptive Systems* 6.2 (2011).

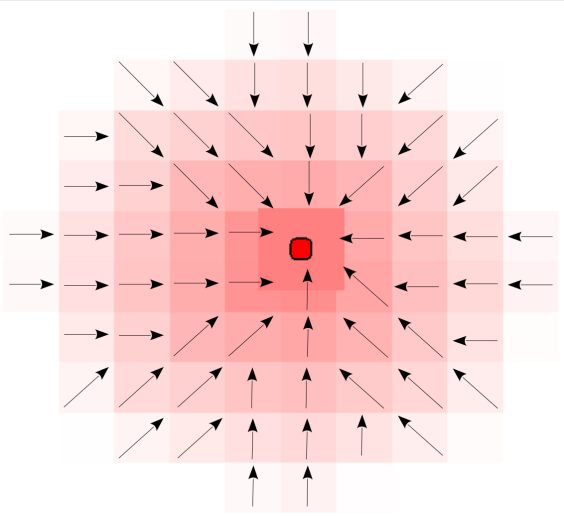
Definition

- A function ϕ mapping nodes of a (dense) network to structured values
- Defines a spatio-temporal structure situated in the (physical) world
- Paradigmatic case: a *gradient* (i.e., `distance-to` field)

$$\textit{Spreading: } x \Rightarrow (x+1)^{\rightsquigarrow} \qquad \textit{Aggregation: } x + y \Rightarrow \textit{min}(x, y)$$

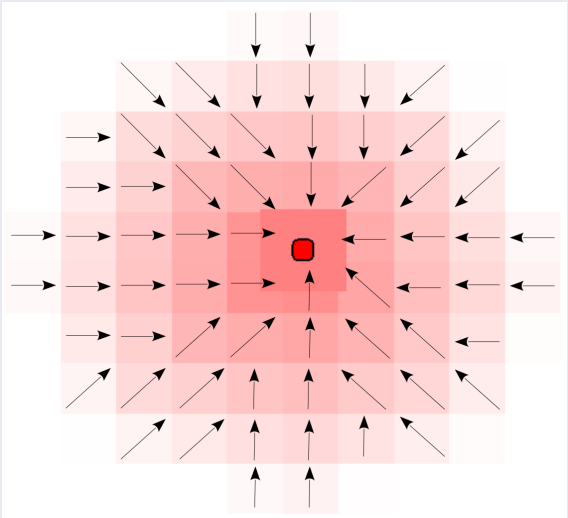
Towards a library of computational fields

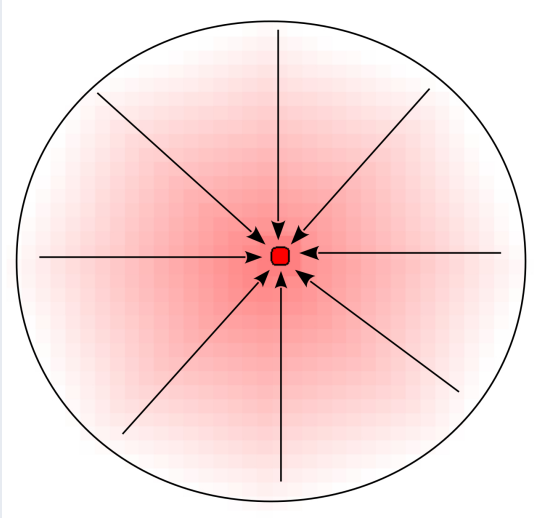
Gradient (discrete, flat continuous, structured continuous)



Towards a library of computational fields

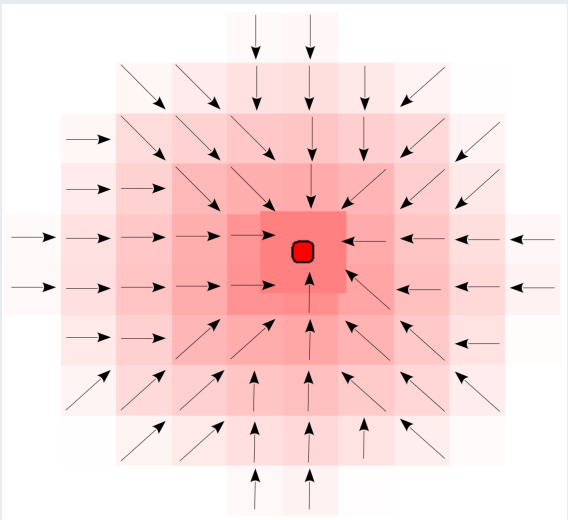
Gradient (discrete, flat continuous, structured continuous)

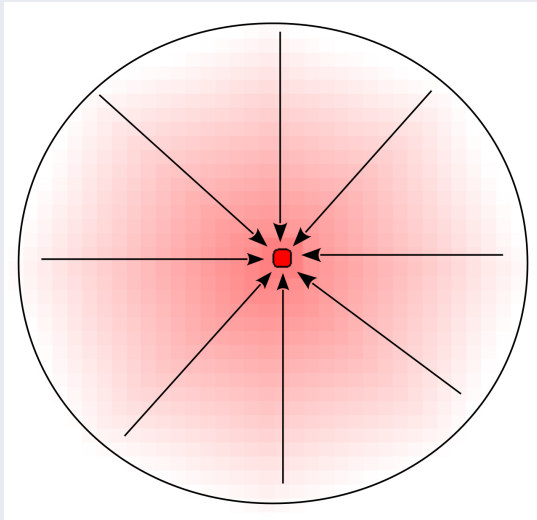


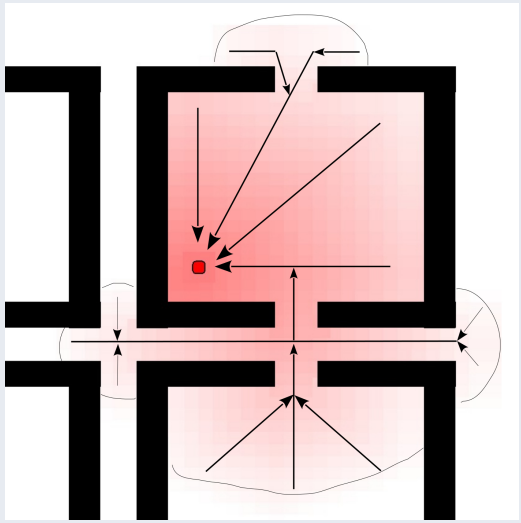


Towards a library of computational fields

Gradient (discrete, flat continuous, structured continuous)

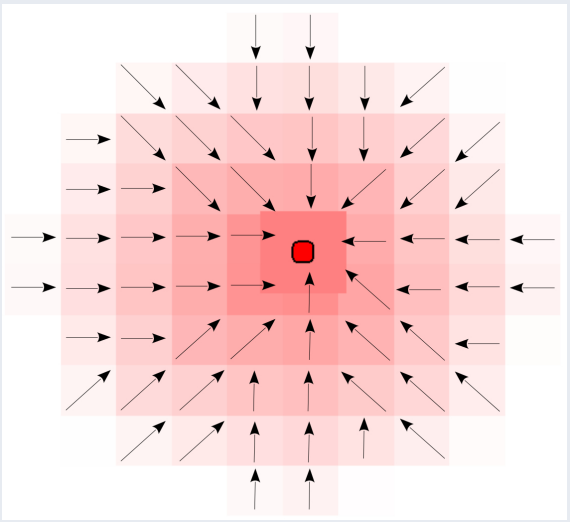
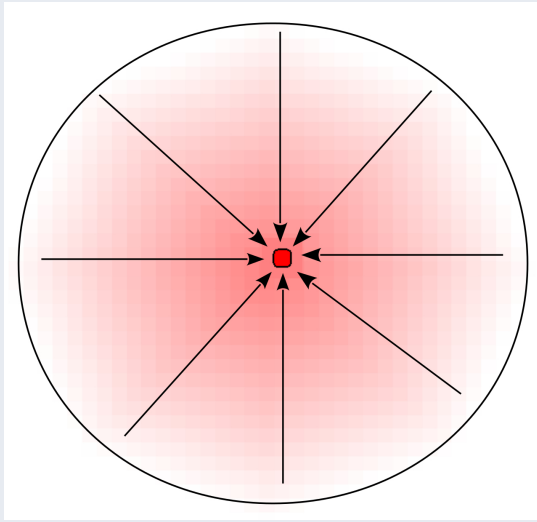
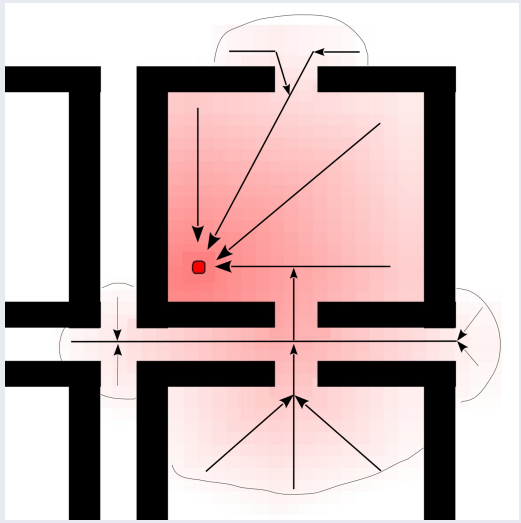




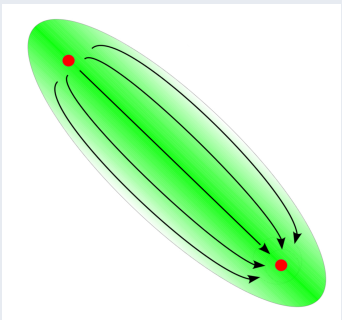
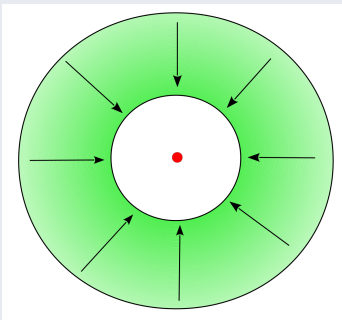
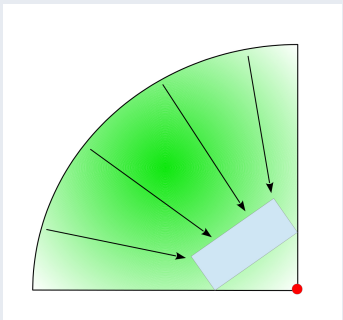
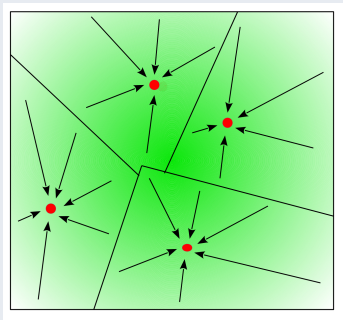
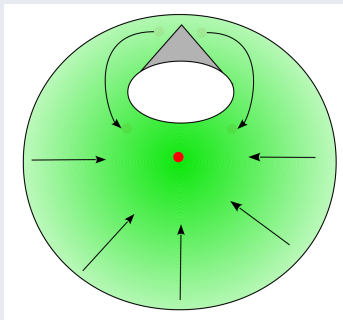


Towards a library of computational fields

Gradient (discrete, flat continuous, structured continuous)

Other structures (channel, shrinking crown, sector, partition, shadow, . . .)

The quest for well-engineered computational fields

General problems we are interested in

- Find good constructs to program spreading-aggregation process
- Predict the final outcome of a field computation
- Find sufficient conditions for certain key properties – stabilisation

Two recent converging research threads

- Proto: the (functional) language of space-time

$\Rightarrow$ Isolating an expressive and tractable core of Proto[a]

- SAPERE: chemical-inspired fields for pervasive computing[b]

$\Rightarrow$ Seeking for "predictable" outcomes, taming nondeterminism

---

[a] Mirko Viroli, Jacob Beal, and Kyle Usbeck. "Operational Semantics of Proto". In: *Science of Computer Programming* 78.6 (2013). Edited by Marjan Mernik, pages 633–656. DOI: j.scico.2012.12.003.

[b] Sara Montagna, Mirko Viroli, Jose Luis Fernandez-Marquez, Giovanna Di Marzo Serugendo, and Franco Zambonelli. "Injecting Self-organisation into Pervasive Service Ecosystems". In: *Mobile Networks and Applications* (2012). Online first.

Two approaches to be possibly reconciled

Functional fields (Proto): a programming language approach

```
(rep d (inf) (mux (sense 1) 0 (min-hood (nbr (+ d 1)))))
```

- High modularity and composability of specifications
- High expressiveness (up to space-time computation completeness)
- Proto programs difficult to analyse in the entire language

Chemical-like rewrite rules (SAPERE): a concurrency model approach

$$x + y \Rightarrow min(x, y) \qquad x \Rightarrow (x + 1)^{\rightsquigarrow}$$

- Not developed in a full-fledged programming language
- More tailored to the "coordination" aspects of spatial systems
- Seemingly more easy to analyse in terms of the structure of rules

Outline

Contributions

- Reconcile the functional and rewrite-based field approaches
- Accordingly give sufficient conditions for stabilisation (via confluence)

Progression

1. A core functional calculus of fields
2. Semantics by rewrite rules
3. Proving the confluence property
4. Implications on behaviour predictability

Outline

Contributions

- Reconcile the functional and rewrite-based field approaches
- Accordingly give sufficient conditions for stabilisation (via confluence)

Progression

1. A core functional calculus of fields
2. Semantics by rewrite rules
3. Proving the confluence property
4. Implications on behaviour predictability

Definition

Core Syntax

Let f, g be computable pointwise-applied function expressions:

$$F \quad ::= \quad \sigma \mid f(F_1, \ldots, F_n) \mid [g(\{F\}, F_1, \ldots, F_n)]$$

A field expression (or process) F is either:

- an environmental field σ (e.g. enacted by distributed sensors);
- functional composition of fields $F_1, \ldots, F_n$ $(n \geq 0)$ by function f;
- a spreading-aggregation field $[g(\{F\}, F_1, \ldots, F_n)]$:
  - originated by F
  - continuosly spread to neighbours
  - locally contextualised by function $g(\cdot, F_1, \ldots, F_n)$
  - reconciling multiplicity of values in a node by taking the minimum

  $\Rightarrow$ rough Proto-equivalent:

```
(rep x NaN (min F (min-hood+ (g (nbr x) F1 .. Fn))))
```

Surface language

On functions

- f, g are denoted by functional expressions
- functions are partial, and can test for absence of a value
- allow non-recursive function definitions (arguments capitalised)
- values ($0-ary$ functions) are literals (numbers, strings) and tuples
- use built-in mathematical and tuple-like functions and operators

  $\Rightarrow$ `+`, `-`, `*`, `0`, `1`, `>=`, `not`, `?:`, `tup`, `1st`, `2nd`, `min`
- minimum function over tuples is custom

Examples of environmental fields

- `#`: constant field undefined everywhere
- `#SRC`,`#DST`: generic sensors (possibly undefined somewhere)
- `#RNG`: 0 on physical nodes, r on virtual nodes representing connections

Mini-tutorial

Basic spreading-aggregation fields

Assume `#SRC` is defined in some *sources*, where it holds 0:

`[{#SRC}]`: creates a plateau of 0 out of sources

`[{#SRC}+1]`: creates a hop-count gradient from sources

`[{#SRC}+#RNG]`: creates a (distance-to) gradient from sources

Restriction (if): can be used to block spreading-aggregation

`[#OBS ? # : {#SRC}+#RNG]`: gradient avoiding the domain of `#OBS`

this is different from expression `#OBS ? # : [{#SRC}+#RNG]`

GradCast (gradient propagating `#VAL` from `#SRC`)

`[{<#SRC,#VAL>}+<#RNG,0>)]`: partitioning out of `#SRC`'s domain nodes

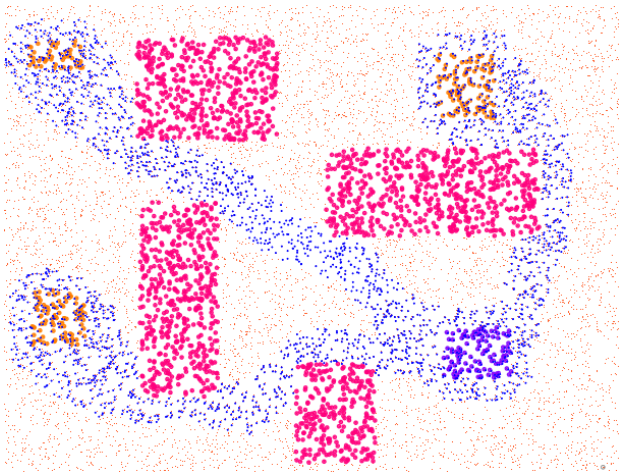
Example 1: Channel

Few definitions

```
def If(FIELD,WHERE) is WHERE ? FIELD : #
def Gossip(VAL) is [{VAL}]
def Grad(SRC) is [{SRC}+#RNG]
def Grad(SRC,OBS) is [If({SRC}+#RNG,not(OBS))]
def Dist(SRC,DST,OBS) is Gossip(If(Grad(SRC,OBS),DST))
```

Channel from #S to #D of width #W, avoiding #O

```
Grad(#S,#O) + Grad(#D,#O) + #W < Dist(#S,#D,#O)
```

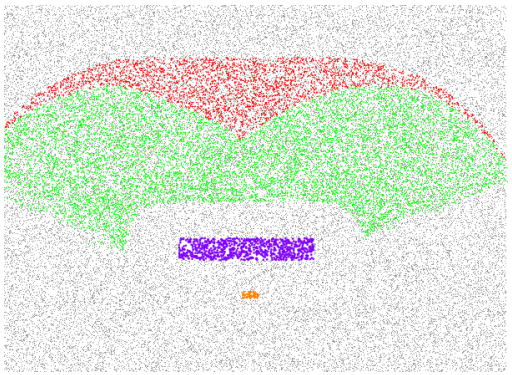


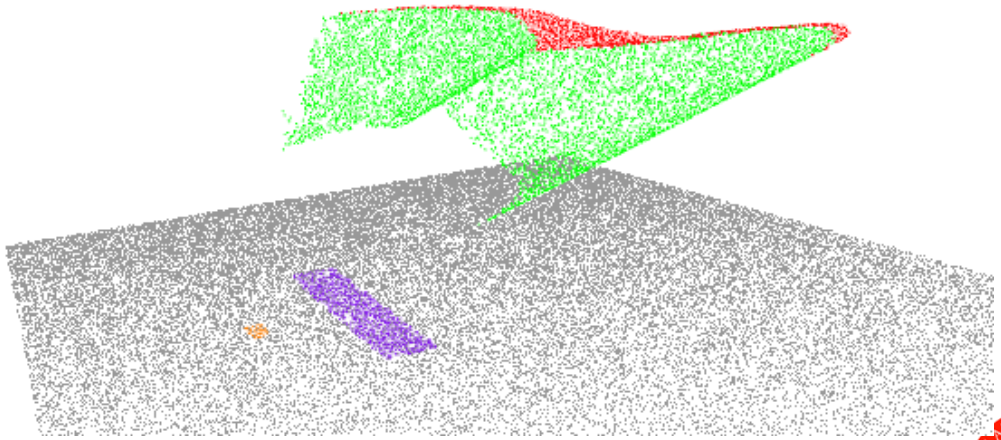
(Simulation of an equivalent model developed in Proto)

Example 2: Anticipative gradient [paper at SASO 2012]

AG from S, with future event F in between distance/time D1 and D2

```
def Sector(S,F) is 2nd( [ {<S,F>}+<#RNG,F> ] )
def Crown(S,D1,D2) is Grad(S)>=D1 and Grad(S)<=D2
def Warn(S,F,D1,D2) is Grad(S,F) min Grad(F)-Grad(S)+D2
def Ag(S,F,D1,D2) is Sector(S,F) and Crown(S,D1,D2) ?  Warn(S,F,D1,D2)
                                                   :  Grad(S)
```





(Simulation of an equivalent model developed in Proto)

Outline

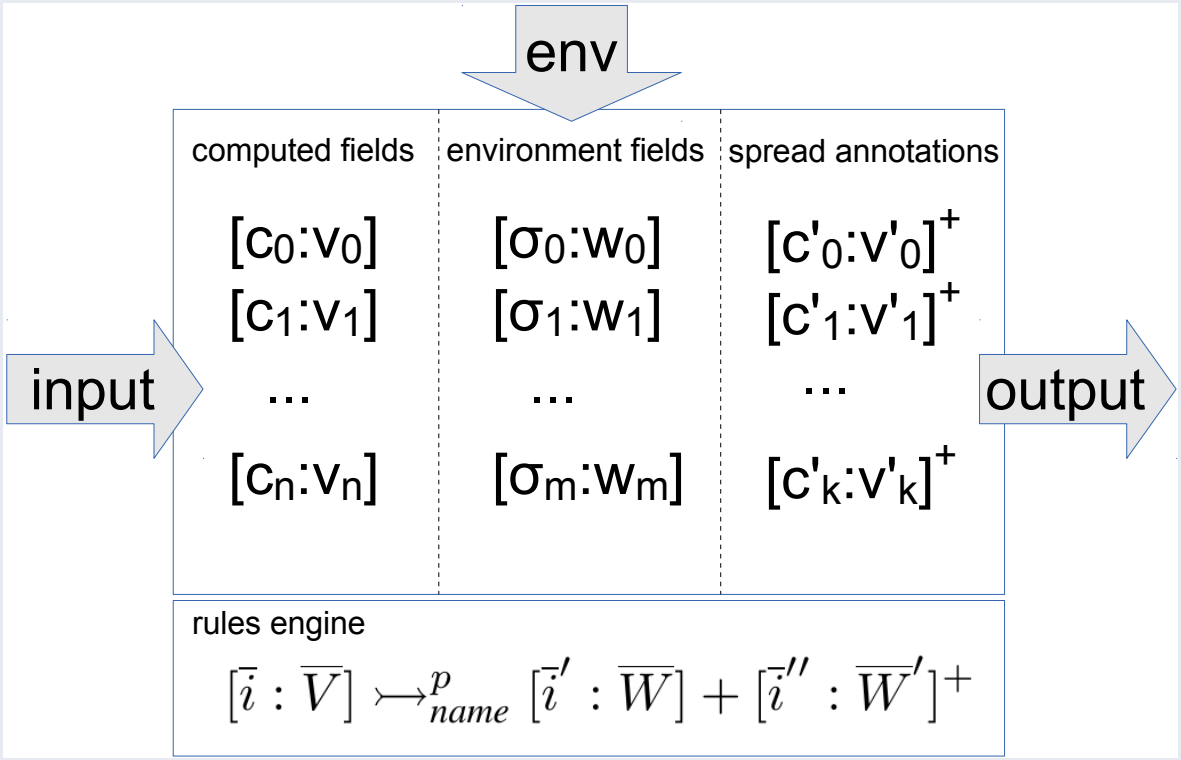
Contributions

- Reconcile the functional and rewrite-based field approaches
- Accordingly give sufficient conditions for stabilisation (via confluence)

Progression

Node architecture and rule model

Architecture



Computation round

Sleep, clean, environment update, input, exec rules, output

Rewrite Semantics of the Field Calculus

Universal rules

$$
\begin{array}{rcll}
\mathcal{F}(i,\bar{i},f) & = & [\bar{i}:\overline{x}] \rightarrowtail_F [\bar{i}:\overline{x}]+[i:f(\overline{x})] & \textit{Functional comp.} \\
\mathcal{C}(i_s,i_c,\bar{i},g) & = & [i_s:x]+[\bar{i}:\overline{x}] \rightarrowtail^*_C [\bar{i}:\overline{x}]+[i_c:g(x,\overline{x})] & \textit{Contextualisation} \\
\mathcal{A}(i_c) & = & [i_c:x]+[i_c:x'] \rightarrowtail^*_A [i_c:x \wedge x'] & \textit{Min-aggregation} \\
\mathcal{O}(i_c,i,i_o) & = & [i_c:x]+[i:x'] \rightarrowtail_O [i:x']+[i_o:x \wedge x'] & \textit{Output} \\
\mathcal{S}(i_o,i_s) & = & [i_o:x] \rightarrowtail_S [i_o:x]+[i_s:x]^+ & \textit{Spreading}
\end{array}
$$

Deriving the rules needed for a given field expression

$$\frac{}{|\sigma| \stackrel{\sigma}{=} \cdot}$$

$$\frac{\textit{fresh}(i) \quad |\overline{F}| \stackrel{\bar{i}}{=} \overline{R} \quad R_F = \mathcal{F}(i,\bar{i},f)}{|f(\overline{F})| \stackrel{i}{=} (\overline{R}\ R_F)}$$

$$\frac{\begin{array}{c}\textit{fresh}(i_s,i_c,i_o) \quad |F| \stackrel{i}{=} \overline{R} \quad |\overline{F}| \stackrel{\bar{i}}{=} \overline{R}' \\ R_C = \mathcal{C}(i_s,i_c,\bar{i},g) \quad R_A = \mathcal{A}(i_c) \\ R_O = \mathcal{O}(i_c,i,i_o) \quad R_S = \mathcal{S}(i_o,i_s)\end{array}}{|[g(\{F\},\overline{F})]| \stackrel{i_o}{=} (\overline{R}\ \overline{R}'\ R_C\ R_A\ R_O\ R_S)}$$

Outline

Contributions

- Reconcile the functional and rewrite-based field approaches
- Accordingly give sufficient conditions for stabilisation (via confluence)

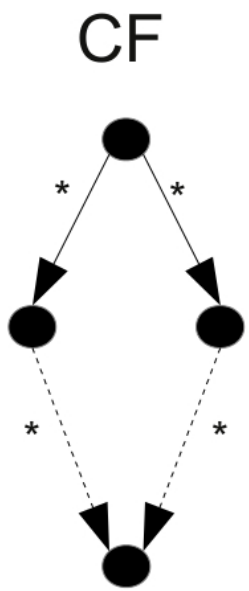
Progression

1. A core functional calculus of fields
2. Semantics by rewrite rules
3. Proving the confluence property
4. Implications on behaviour predictability

Motivations and Conditions for Confluence

Idea and consequence

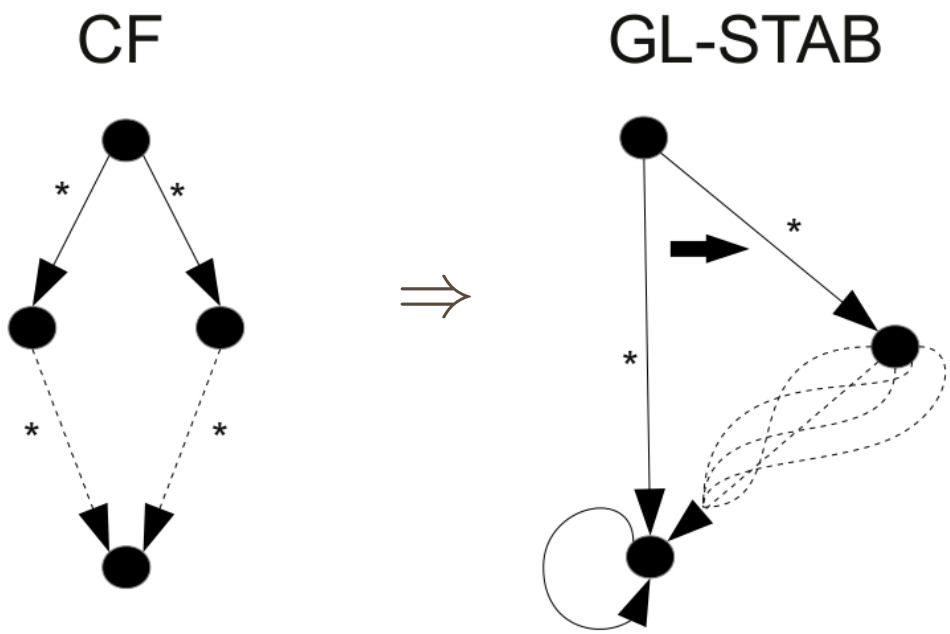
- Idea: prove CF by local CF + absense of infinite transition sequences
- Consequence: it will suffice to show that one sequence of transitions leads to a stable field S, to prove that S will be inevitably reached



Motivations and Conditions for Confluence

Idea and consequence

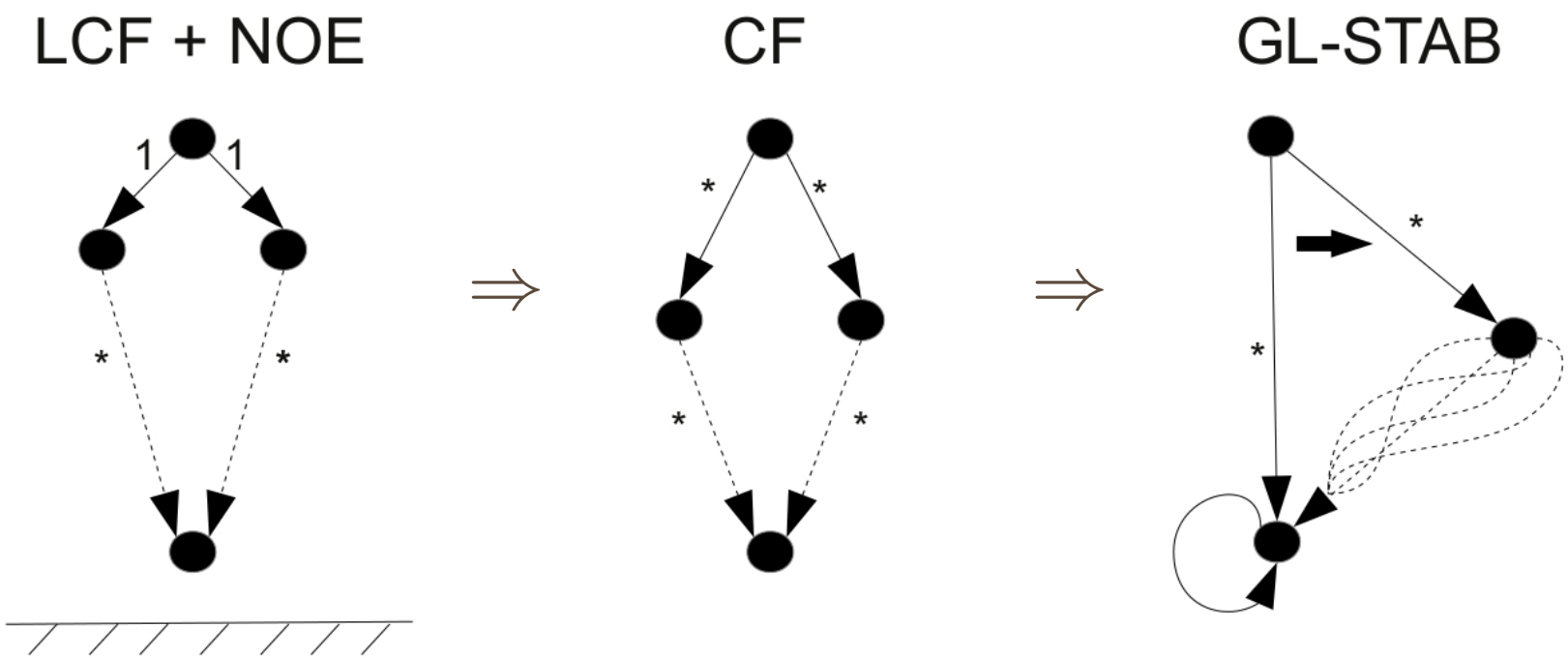
- Idea: prove CF by local CF + absense of infinite transition sequences
- Consequence: it will suffice to show that one sequence of transitions leads to a stable field S, to prove that S will be inevitably reached



Motivations and Conditions for Confluence

Idea and consequence

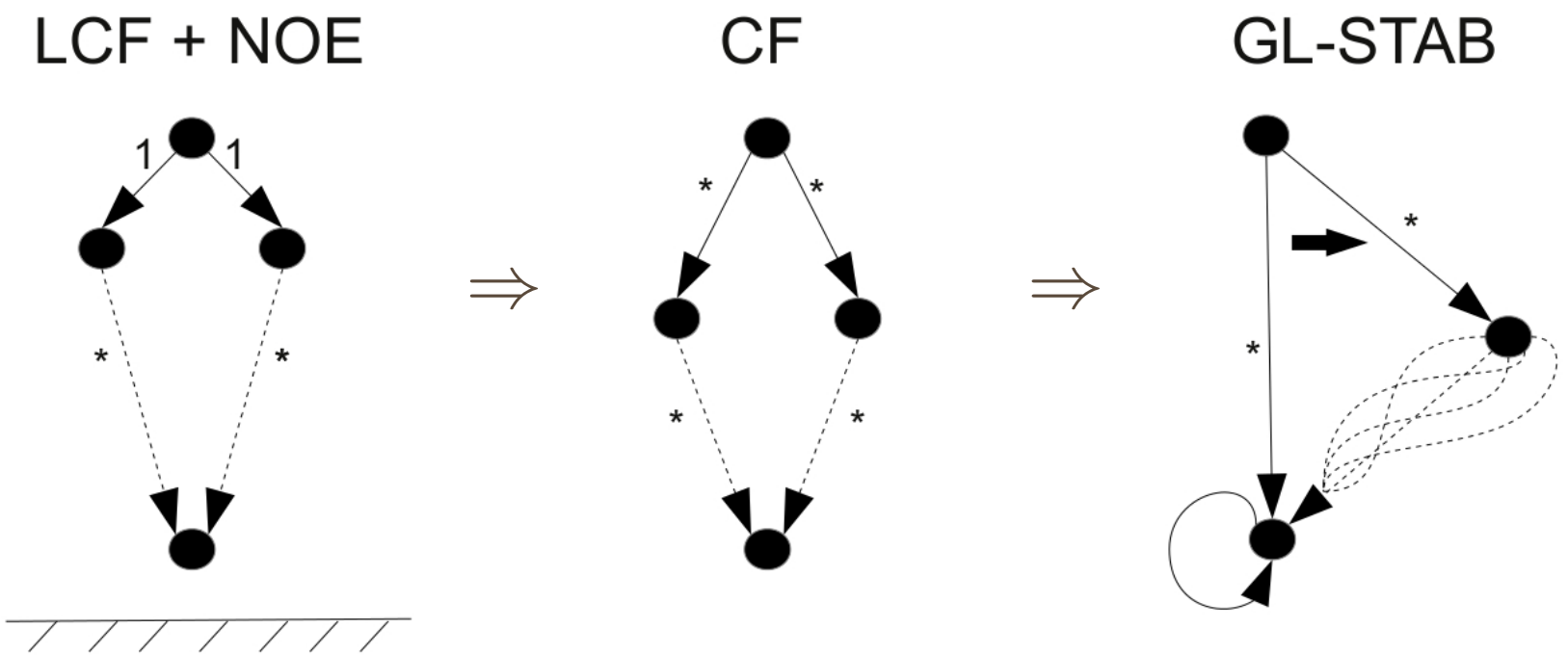
- Idea: prove CF by local CF + absense of infinite transition sequences
- Consequence: it will suffice to show that one sequence of transitions leads to a stable field S, to prove that S will be inevitably reached



Motivations and Conditions for Confluence

Idea and consequence

- Idea: prove CF by local CF + absense of infinite transition sequences
- Consequence: it will suffice to show that one sequence of transitions leads to a stable field S, to prove that S will be inevitably reached



Our proof methodology based on confluence

Sufficient conditions for confluence

- Fair evolution (the whole network computes)
  - $\Rightarrow$ at each step, each node n is given another chance to "fire"
- Transitory evolution to topoloy T (avoiding network partition issues)
  - $\Rightarrow$ start with $T' \subseteq T$, can add/remove links, never escape T, reach T
  - $\Rightarrow$ environmental fields eventually stabilise
- Well-monotonicity (ensuring "termination" of spreading-aggregation)
  - $\Rightarrow$ $g(v, \overline{v}) \geq v$
  - $\Rightarrow$ $g(v, \overline{v}) \geq g(v', \overline{v})$ if $v \geq v'$
  - $\Rightarrow$ no infinite sequence built as $v_i = g(v_{i-1}, \overline{v})$ is strictly ascending

Generality of those conditions

- After ensuring values have an upper-bound..
- `[{#SRC}]`, `[{#SRC}+1]`, `[{#SRC}+#RNG]`,... and all others, are ok!

Outline

Contributions

- Reconcile the functional and rewrite-based field approaches
- Accordingly give sufficient conditions for stabilisation (via confluence)

Progression

1. A core functional calculus of fields
2. Semantics by rewrite rules
3. Proving the confluence property
4. Implications on behaviour predictability

Result 1: Compositionality of stabilised fields

Theorem

- Consider field specifications $F_1, \ldots, F_n$, each stabilising;
- let $\Downarrow F_i$ be the fields ϕ_i that F_i stabilises to;
- then, informally: $\Downarrow f(F_1, \ldots, F_n) = f(\Downarrow F_1, \ldots, \Downarrow F_n)$
- and, $\Downarrow [g(\{F_1\}, F_2, \ldots, F_n)] = [g(\{\Downarrow F_1\}, \Downarrow F_2, \ldots, \Downarrow F_n)]$

Rationale

- Field evolution processes are very nondeterministic
- Though, we can think to field expressions in terms of their (stabilised) outcome, i.e., the resulting field ϕ constructed compositionally

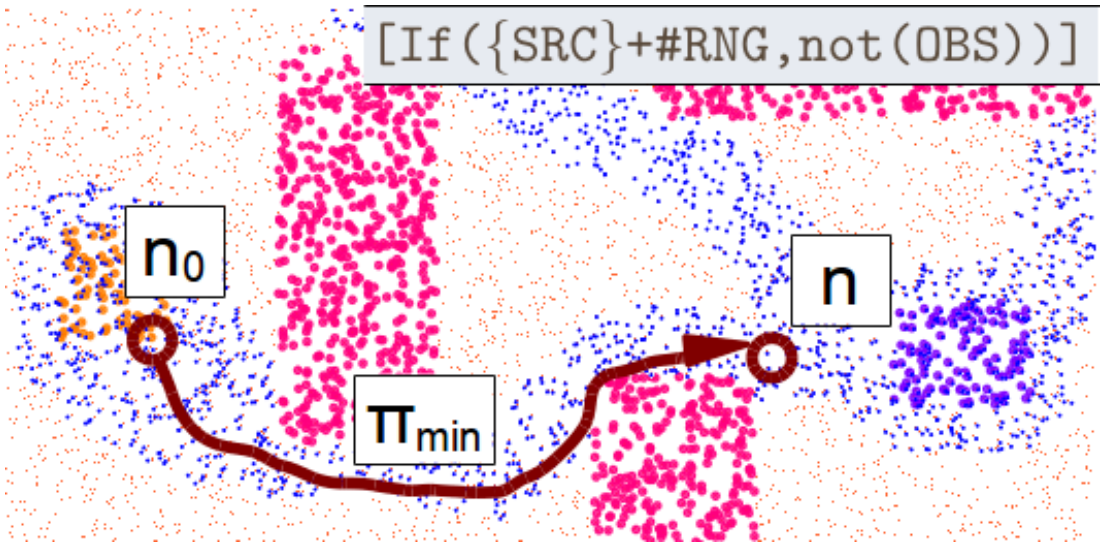
$\Rightarrow$ Namely, we can forget about transient effects

Result 2: Outcome of spreading-aggregation

What $[g(\{F_1\}, F_2, \ldots, F_n)]$ stabilises to?

Computing stable value v in node n

- it is the same you would have considering a chain π_{min} in isolation
- π_{min} is the shortest path according to the metric imposed by g, F_i
- value v is easily iteratively computed out of g
- stabilisation-time is linear in the size of π_{min}



Conclusions

Main result

Proving that a whole set of fields are stabilising
$\Rightarrow$ Fostering productio of a library of well-engineered selforg mechanisms

Future technical works

- enlarging applicability (e.g., releasing boundedness)
- enriching the spreading-aggregation construct
- studying further properties (soundness, convergence to continuum)

Future practical works

- formal mapping to Proto (towards better analysis of Proto programs)
- formal mapping to SAPERE (applying in pervasive computing)

Engineering Confluent Computational Fields

from Functions to Rewrite Rules

Mirko Viroli

Dipartimento di Informatica: Scienza e Ingegneria (DISI)
Alma Mater Studiorum—Università di Bologna, Italy
mirko.viroli@unibo.it

Spatial Computing Workshop, St. Paul (USA), 6/5/2013