Multi-agent Systems with Virtual Stigmergy

Rocco De Nicola[1] **Luca Di Stefano**[2] Omar Inverso[2]

[1] IMT School of Advanced Studies, Lucca, Italy
[2] Gran Sasso Science Institute (GSSI), L'Aquila, Italy

Multi-Agent Systems

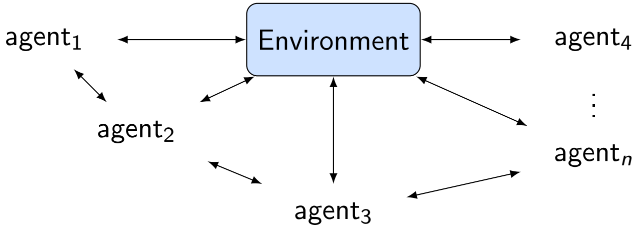
Composed of *agents* situated in an *environment*

Agents

- simple local rules
- limited awareness

Interaction

- among agents
- between each agent and the environment



Multi-Agent Systems

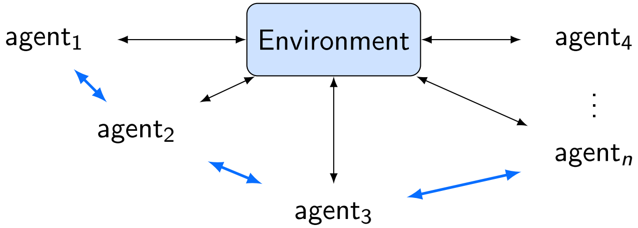
Composed of *agents* situated in an *environment*

Agents

- simple local rules
- limited awareness

Interaction

- among agents
- between each agent and the environment



Multi-Agent Systems

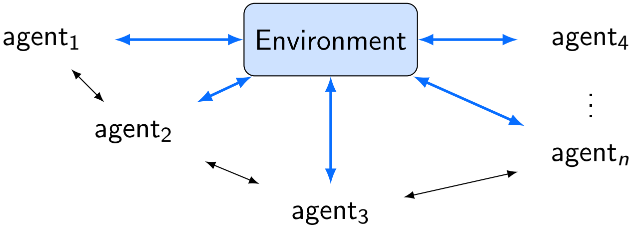
Composed of *agents* situated in an *environment*

Agents

- simple local rules
- limited awareness

Interaction

- among agents
- between each agent and the environment



Emerging Behaviour

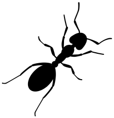
Local behaviour $\leftrightsquigarrow$ Complex emerging behaviour

- No apparent planning
- No centralized control

Emerging Behaviour

Local behaviour $\leftrightsquigarrow$ Complex emerging behaviour

- No apparent planning
- No centralized control



$\overset{?}{\rightsquigarrow}$

Emerging Behaviour

Local behaviour $\leftrightsquigarrow$ Complex emerging behaviour

- No apparent planning
- No centralized control

$\overset{?}{\leftsquigarrow}$

Stigmergy

1. Agents drop messages in the environment
2. Other agents read the messages and use them to decide future behaviour

Wikipedia[1]

- A single user creates a new article (with errors)
- Users visit the article
- Some of them notice errors
  - Grammar/ortography
  - Incorrect information
- Some of those who notice errors also fix them!

[1] F. Heylighen, "Stigmergy as a universal coordination mechanism I: Definition and components," Cognitive Systems Research 38, 2016.

Stigmergy

1. Agents drop messages in the environment
2. Other agents read the messages and use them to decide future behaviour

Wikipedia[1]

- A single user creates a new article (with errors)
- Users visit the article
- Some of them notice errors
  - Grammar/ortography
  - Incorrect information
- Some of those who notice errors also fix them!

[1] F. Heylighen, "Stigmergy as a universal coordination mechanism I: Definition and components," Cognitive Systems Research 38, 2016.

Stigmergy in MASs

Cognitive Stigmergy[2]

- Engineer stigmergic interaction in a MAS setting
- A set of artifacts to manage stigmergic messages (*annotations*)

Virtual Stigmergy[3] (Buzz language)

- Practical implementation
- Multi-Robot Systems

[2] A. Ricci, A. Omicini, M. Viroli, L. Gardelli, and E. Oliva, "Cognitive Stigmergy: Towards a Framework Based on Agents and Artifacts," E4MAS, 2006.

[3] C. Pinciroli, A. Lee-Brown, and G. Beltrame, "A Tuple Space for Data Sharing in Robot Swarms," BICT, 2015.

Stigmergy in MASs

Cognitive Stigmergy[2]

- Engineer stigmergic interaction in a MAS setting
- A set of artifacts to manage stigmergic messages (*annotations*)

Virtual Stigmergy[3] (Buzz language)

- Practical implementation
- Multi-Robot Systems

[2] A. Ricci, A. Omicini, M. Viroli, L. Gardelli, and E. Oliva, "Cognitive Stigmergy: Towards a Framework Based on Agents and Artifacts," E4MAS, 2006.

[3] C. Pinciroli, A. Lee-Brown, and G. Beltrame, "A Tuple Space for Data Sharing in Robot Swarms," BICT, 2015.

Generalizing stigmergic interaction

To pick up a message, you have to...

- visit a *location* containing messages (Cognitive stigmergy)
- be sufficiently *close* to other agents (Virtual stigmergy)

Is spatial neighbourhood general/intuitive enough?

Wikipedia

To fix an error you have to...

- Visit a page
- Know (English/the topic) well enough to notice the error
- The more pages you visit, the better you know English

Generalizing stigmergic interaction

To pick up a message, you have to...

- visit a *location* containing messages (Cognitive stigmergy)
- be sufficiently *close* to other agents (Virtual stigmergy)

Is spatial neighbourhood general/intuitive enough?

Wikipedia

To fix an error you have to...

- Visit a page
- Know (English/the topic) well enough to notice the error
- The more pages you visit, the better you know English

Introducing LAbS

Language with Attribute-based Stigmergy

A simple process algebra to describe MASs

Interaction:

- Shared memory (*environment*)
- Virtual stigmergy

Goals:

- Generalize the behaviour of stigmergy
- Easily mechanizable encoding for automated verification

Introducing LAbS

Language with Attribute-based Stigmergy

A simple process algebra to describe MASs

Interaction:

- Shared memory (*environment*)
- Virtual stigmergy

Goals:

- Generalize the behaviour of stigmergy
- Easily mechanizable encoding for automated verification

Introducing LAbS

Language with Attribute-based Stigmergy

A simple process algebra to describe MASs

Interaction:

- Shared memory (*environment*)
- Virtual stigmergy

Goals:

- Generalize the behaviour of stigmergy
- Easily mechanizable encoding for automated verification

Attribute-based communication

- Systems are represented as sets of parallel components
- Components are equipped with a set of attributes (key-value pairs)
- Attribute values can be modified by internal actions
- Components are not aware of the existence of each other

AbC calculus[4]

- Senders send to all those that satisfy a predicate over attributes
- Receivers only accept messages if the sender satisfying a predicate
- Receive is blocking (synchronizes with available sent messages)

LAbS

- Focus on the indirect nature of stigmergic interaction
- No send, receive in individual behaviours

[4] Y. Abd Alrahman, R. De Nicola, and M. Loreti, "On the Power of Attribute-Based Communication," FORTE, 2016.

Attribute-based communication

- Systems are represented as sets of parallel components
- Components are equipped with a set of attributes (key-value pairs)
- Attribute values can be modified by internal actions
- Components are not aware of the existence of each other

AbC calculus[4]

- Senders send to all those that satisfy a predicate over attributes
- Receivers only accept messages if the sender satisfying a predicate
- Receive is blocking (synchronizes with available sent messages)

LAbS

- Focus on the indirect nature of stigmergic interaction
- No send, receive in individual behaviours

[4] Y. Abd Alrahman, R. De Nicola, and M. Loreti, "On the Power of Attribute-Based Communication," FORTE, 2016.

Attribute-based communication

- Systems are represented as sets of parallel components
- Components are equipped with a set of attributes (key-value pairs)
- Attribute values can be modified by internal actions
- Components are not aware of the existence of each other

AbC calculus[4]

- Senders send to all those that satisfy a predicate over attributes
- Receivers only accept messages if the sender satisfying a predicate
- Receive is blocking (synchronizes with available sent messages)

LAbS

- Focus on the indirect nature of stigmergic interaction
- No send, receive in individual behaviours

[4] Y. Abd Alrahman, R. De Nicola, and M. Loreti, "On the Power of Attribute-Based Communication," FORTE, 2016.

Link Predicates

Generalize stigmergic interaction

- We can define a **link predicate** φ over attributes
- If two components satisfy φ they can communicate
- Each component stores its attributes in an **interface** $\mathtt{I} : \mathcal{K} \hookrightarrow \mathcal{V}$
- The value of attributes can change $\Rightarrow$ links are *dynamic*

Examples (1 = sender, 2 = receiver)

$true$	Broadcast
$\lVert \mathtt{I}_1(pos) - \mathtt{I}_2(pos) \rVert \leq \delta$	Ranged broadcast
$\mathtt{I}_1(pub\_id) = \mathtt{I}_2(sub\_id)$	Publish-subscribe

Link Predicates

Generalize stigmergic interaction

- We can define a **link predicate** φ over attributes
- If two components satisfy φ they can communicate
- Each component stores its attributes in an **interface** $\mathtt{I} : \mathcal{K} \hookrightarrow \mathcal{V}$
- The value of attributes can change $\Rightarrow$ links are *dynamic*

Examples (1 = sender, 2 = receiver)

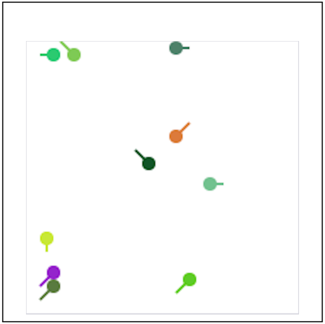
true	Broadcast
$\lVert \mathtt{I}_1(\mathit{pos}) - \mathtt{I}_2(\mathit{pos}) \rVert \leq \delta$	Ranged broadcast
$\mathtt{I}_1(\mathit{pub\_id}) = \mathtt{I}_2(\mathit{sub\_id})$	Publish-subscribe

Example: flocking

Each component:

1. Picks an arbitrary direction (dx, dy) from a given set D
2. Stores (dx, dy) in the virtual stigmergy
3. (Recursively) Moves in the direction stored in the v. stigmergy

The arena wraps round (agents can cross an edge and reach the opposite side)



A possible execution

Components

A LAbS component is a 5-ple

$$\langle\, \mathtt{I}, L, P, Z_c, Z_p \rangle$$

Components

A LAbS component is a 5-ple

$$\langle \mathtt{I}, L, P, Z_c, Z_p \rangle$$

Interface

Components

A LAbS component is a 5-ple

$$\langle \mathtt{I}, L, P, Z_c, Z_p \rangle$$

Interface

Local stigmergy

Components

Behaviour

A LAbS component is a 5-ple

$$\langle \mathtt{I}, L, P, Z_c, Z_p \rangle$$

Interface

Local stigmergy

Components

Behaviour

A LAbS component is a 5-ple

$$\langle \mathtt{I}, L, P, Z_c, Z_p \rangle$$

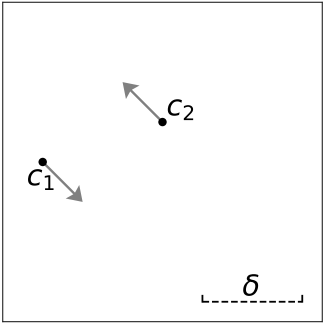
Interface

Local stigmergy

For asynchronous interaction

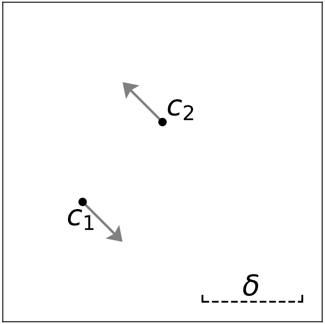
Virtual Stigmergy: Example

Two agents c_1, c_2 moving in different directions
When c_1, c_2 are within a given distance δ, they exchange messages to agree on a common direction



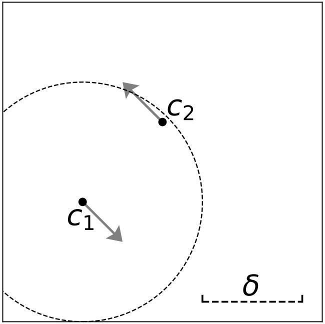
Virtual Stigmergy: Example

Two agents c_1, c_2 moving in different directions
When c_1, c_2 are within a given distance δ, they exchange messages to agree on a common direction



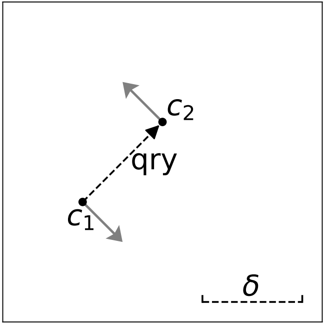
Virtual Stigmergy: Example

Two agents c_1, c_2 moving in different directions
When c_1, c_2 are within a given distance δ, they exchange messages to agree on a common direction



Virtual Stigmergy: Example

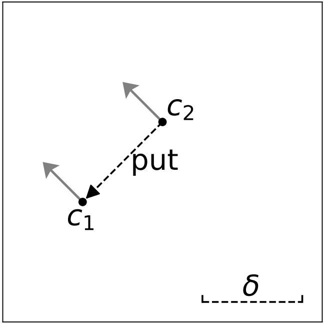
Two agents c_1, c_2 moving in different directions
When c_1, c_2 are within a given distance δ, they exchange messages to agree on a common direction



qry: agent asks if neighbour agrees with a value

Virtual Stigmergy: Example

Two agents c_1, c_2 moving in different directions
When c_1, c_2 are within a given distance δ, they exchange messages to agree on a common direction



put: agent proposes a value to neighbour

Virtual Stigmergy in LAbS

- Distributed and decentralized data store
- Key-value-timestamp triples
- Local: each component maintains a (possibly incomplete) copy
- Global: rules to allow information spread
  - After reading: send query (*confirm*)
  - After writing: propose new value (*propagate*)
  - After receiving an outdated query: propagate up-to-date triple
  - After receiving a new triple: propagate it

A triple is *new* for a component c **unless** c already stores a triple with the same key and a more recent timestamp.

LAbS syntax: basic processes

Actions:

$\mathtt{I}(x) := E(x)$	Read from the environment
$\mathtt{I}(x) := e$	Update a component's *attribute*
$E(x) := e$	Write into the *environment*
$L(x) := e$	Write into the (local) *stigmergy*

Other basic processes:

0	Idle/deadlocked process
$\surd$	Successfully terminated process

LAbS syntax: processes

$P ::=$	0	Idle process
	$\mid \surd$	Successful termination
	$\mid \alpha$	Basic actions
	$\mid b \rightarrow P$	Guarded process
	$\mid P; P$	Sequentialization
	$\mid P + P$	Nondeterministic choice
	$\mid P \mid P$	Parallel composition
	$\mid K$	Process constant

Guards

$$b ::= \mathit{true} \mid e \bowtie e \mid \neg b \mid b \wedge b$$

$\bowtie$ binary comparison operator

Expressions

$$e ::= v \mid x \mid e \diamond e$$

$\diamond$ binary arithmetic operator

Component semantics

Assign result of an expression to a stigmergy key

1. Evaluate $1 + a$ (call the result v)
2. Get timestamp t from global clock
3. Store *(b, 2, 1)* into local stigmergy
4. (async) Propagate *(b, 2, 1)* to neighbours
5. (async) Ask confirmation for all stigmergy keys in $1 + L(a)$

L	$(a, 1, 0)$
P	$L(b) := 1 + L(a)$

Component semantics

Assign result of an expression to a stigmergy key

1. Evaluate $1 + a$ (call the result v)
2. Get timestamp t from global clock
3. Store *(b, 2, 1)* into local stigmergy
4. (async) Propagate *(b, 2, 1)* to neighbours
5. (async) Ask confirmation for all stigmergy keys in $1 + L(a)$

$$\begin{array}{c|l} \mathbf{L} & (a, 1, 0) \\ \mathbf{P} & L(b) := 1 + L(a) \end{array}$$

$v = 2$

Component semantics

Assign result of an expression to a stigmergy key

1. Evaluate $1 + a$ (call the result v)
2. Get timestamp t from global clock
3. Store *(b, 2, 1)* into local stigmergy
4. (async) Propagate *(b, 2, 1)* to neighbours
5. (async) Ask confirmation for all stigmergy keys in $1 + L(a)$

$$\begin{array}{c|l} \mathbf{L} & (a, 1, 0) \\ \mathbf{P} & L(b) := 1 + L(a) \end{array}$$

$v = 2$ $t = 1$

Component semantics

Assign result of an expression to a stigmergy key

1. Evaluate $1 + a$ (call the result v)
2. Get timestamp t from global clock
3. Store *(b, 2, 1)* into local stigmergy
4. (async) Propagate *(b, 2, 1)* to neighbours
5. (async) Ask confirmation for all stigmergy keys in $1 + L(a)$

L	$(a, 1, 0)$
P	$L(b) := 1 + L(a)$

$v = 2 \qquad t = 1$

L	$(a, 1, 0), (b, 2, 1)$
P	$\surd$

Component semantics

Assign result of an expression to a stigmergy key

1. Evaluate $1 + a$ (call the result v)
2. Get timestamp t from global clock
3. Store *(b, 2, 1)* into local stigmergy
4. (async) Propagate *(b, 2, 1)* to neighbours
5. (async) Ask confirmation for all stigmergy keys in $1 + L(a)$

L	$(a, 1, 0)$
P	$L(b) := 1 + L(a)$

$v = 2$ $t = 1$

L	$(a, 1, 0), (b, 2, 1)$
P	$\surd$
$\mathbf{Z_p}$	**b**

Component semantics

Assign result of an expression to a stigmergy key

1. Evaluate $1 + a$ (call the result v)
2. Get timestamp t from global clock
3. Store *(b, 2, 1)* into local stigmergy
4. (async) Propagate *(b, 2, 1)* to neighbours
5. (async) Ask confirmation for all stigmergy keys in $1 + L(a)$

L	$(a, 1, 0)$
P	$L(b) := 1 + L(a)$

$v = 2 \qquad t = 1$

L	$(a, 1, 0), (b, 2, 1)$
P	$\surd$
$\mathbf{Z_p}$	b
$\mathbf{Z_c}$	**a**

Propagation

If $b \in Z_p$ then a component can:

1. perform a put-transition (denoting its willingness to propagate)
2. remove b from Z_p after the transition

Propagation

If $b \in Z_p$ then a component can:

1. perform a put-transition (denoting its willingness to propagate)
2. remove b from Z_p after the transition

L	$(a, 1, 0), (b, 2, 1)$
P	$\surd$
$\mathbf{Z_p}$	b
$\mathbf{Z_c}$	a

Propagation

If $b \in Z_p$ then a component can:

1. perform a put-transition (denoting its willingness to propagate)
2. remove b from Z_p after the transition

L	$(a, 1, 0), (b, 2, 1)$
P	$\surd$
$\mathbf{Z_p}$	b
$\mathbf{Z_c}$	a

$$\xrightarrow{\text{put}\,(\mathtt{I}, b, 2, 1)}$$

Propagation

If $b \in Z_p$ then a component can:

1. perform a put-transition (denoting its willingness to propagate)
2. remove b from Z_p after the transition

$$\begin{array}{r|l} \mathbf{L} & (a,1,0),(b,2,1) \\ \mathbf{P} & \surd \\ \mathbf{Z_p} & b \\ \mathbf{Z_c} & a \end{array} \xrightarrow{\text{put}\,(\mathtt{I}, b, 2, 1)} \begin{array}{r|l} \mathbf{L} & (a,1,0),(b,2,1) \\ \mathbf{P} & \surd \\ \mathbf{Z_p} & \emptyset \\ \mathbf{Z_c} & a \end{array}$$

Systems

Stigmergic interaction (put)

If
1. S can do a put-transition and become S'
2. Another component c satisfies the link predicate
3. The propagated triple is "new" for c

Then
4. $S \| c$ performs the same put-transition as S
5. c stores the new triple
6. (async) c propagates the new triple

S		
	L	$(a, 1, 0), (b, 2, 1)$
	$\mathbf{Z_p}$	b

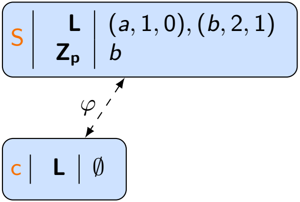
Systems

Stigmergic interaction (put)

If
1. S can do a put-transition and become S'
2. Another component c satisfies the link predicate
3. The propagated triple is “new” for c

Then

4. $S \| c$ performs the same put-transition as S
5. c stores the new triple
6. (async) c propagates the new triple



Systems

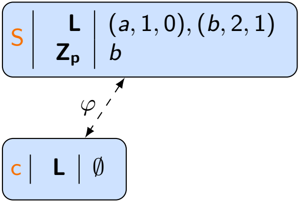
Stigmergic interaction (put)

If

1. S can do a put-transition and become S'
2. Another component c satisfies the link predicate
3. The propagated triple is "new" for c

Then

4. $S \| c$ performs the same put-transition as S
5. c stores the new triple
6. (async) c propagates the new triple

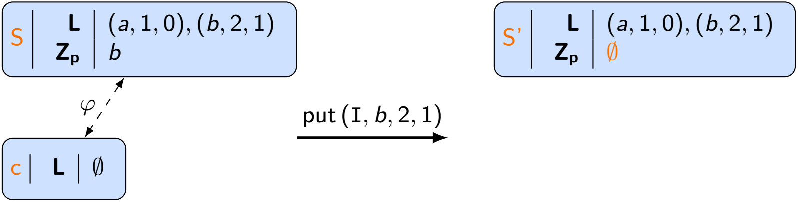


Systems

Stigmergic interaction (put)

If
1. S can do a put-transition and become S'
2. Another component c satisfies the link predicate
3. The propagated triple is “new” for c

Then
4. $S \| c$ performs the same put-transition as S
5. c stores the new triple
6. (async) c propagates the new triple

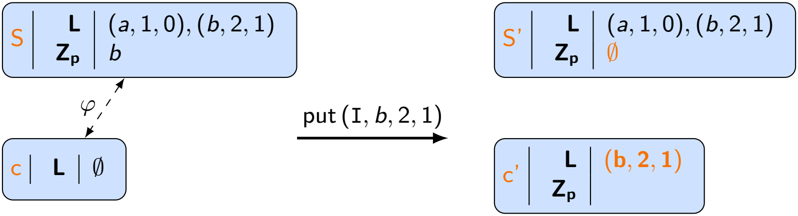


Systems

Stigmergic interaction (put)

If
1. S can do a put-transition and become S'
2. Another component c satisfies the link predicate
3. The propagated triple is "new" for c

Then
4. $S \| c$ performs the same put-transition as S
5. c stores the new triple
6. (async) c propagates the new triple

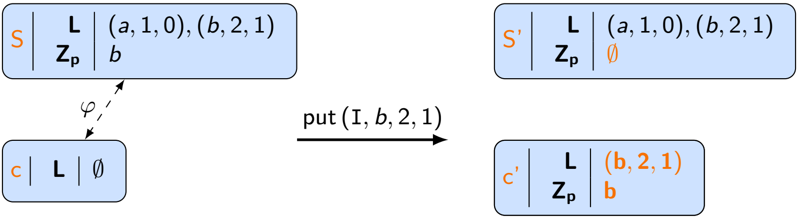


Systems

Stigmergic interaction (put)

If
1. S can do a put-transition and become S'
2. Another component c satisfies the link predicate
3. The propagated triple is "new" for c

Then
4. $S \| c$ performs the same put-transition as S
5. c stores the new triple
6. (async) c propagates the new triple



Environment

Asynchronous primitives are not always well-suited

- Agent a_1 drops some material in a specific location
- Agents a_2 and a_3 have to pick up and move it
- Only one agent can pick it

Situated systems

A situated system is formed by

$$E : \mathcal{K} \hookrightarrow \mathcal{V} \ (\textit{environment})$$

$$S : \text{a LAbS system}$$

The environment has the role of a shared memory
Actions on E are synchronous and atomic
(We rely on interleaving semantics)

Environment

Asynchronous primitives are not always well-suited

- Agent a_1 drops some material in a specific location
- Agents a_2 and a_3 have to pick up and move it
- Only one agent can pick it

Situated systems

A situated system is formed by

$$E : \mathcal{K} \hookrightarrow \mathcal{V} \ (\textit{environment})$$
$$S : \text{a LAbS system}$$

The environment has the role of a shared memory
Actions on E are synchronous and atomic
(We rely on interleaving semantics)

Environment

Asynchronous primitives are not always well-suited

- Agent a_1 drops some material in a specific location
- Agents a_2 and a_3 have to pick up and move it
- Only one agent can pick it

Situated systems

A situated system is formed by

$$E : \mathcal{K} \hookrightarrow \mathcal{V} \text{ (\textit{environment})}$$

$$S : \text{a LAbS system}$$

The environment has the role of a shared memory

Actions on E are synchronous and atomic

(We rely on interleaving semantics)

Environment

Asynchronous primitives are not always well-suited

- Agent a_1 drops some material in a specific location
- Agents a_2 and a_3 have to pick up and move it
- Only one agent can pick it

Situated systems

A situated system is formed by

$$E : \mathcal{K} \hookrightarrow \mathcal{V} \ (\textit{environment})$$

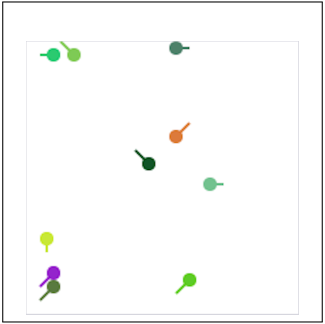
$$S : \text{a LAbS system}$$

The environment has the role of a shared memory
Actions on E are synchronous and atomic
(We rely on interleaving semantics)

Example: flocking (again)

$$\begin{aligned}
\textsc{Flock} &\triangleq \textstyle\sum_{(i,j)\in D} L(dir) := (i,j); \textsc{Move} \\
\textsc{Move} &\triangleq \mathtt{I}(pos) := \mathtt{I}(pos) + L(dir) \mod G; \textsc{Move} \\
\varphi &\triangleq \| \mathtt{I}_1(pos) - \mathtt{I}_2(pos) \| \leq \delta
\end{aligned}$$

G = size of the arena



A possible execution

Future work on LAbS

- From global to local clocks
  - Lamport timestamps[5]?
- Additional primitives
  - Send/receive between neighbours
  - Different link predicates for different keys (or different stigmergies?)

[5] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," Comm. ACM Vol. 21, 1978.

SLiVER

Symbolic LAbS Verifier

How to use SLiVER:

1. Write a specification file (system + properties)
2. Invoke SLiVER with arguments:
   - The name of the file
   - A bound on the number of transitions
   - Other optional arguments
3. Possible outputs:
   - Success
   - Property violated + counterexample
   - Out of time/memory/etc.

SLiVER

Symbolic LAbS Verifier

How to use SLiVER:

1. Write a specification file (system + properties)
2. Invoke SLiVER with arguments:
   - The name of the file
   - A bound on the number of transitions
   - Other optional arguments
3. Possible outputs:
   - Success
   - Property violated + counterexample
   - Out of time/memory/etc.

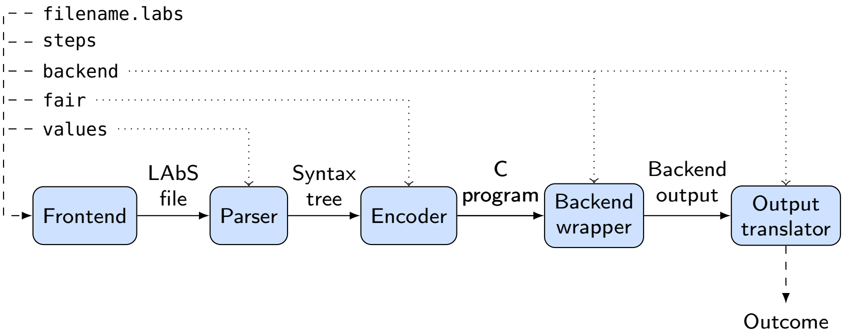
SLiVER

Symbolic LAbS Verifier

How to use SLiVER:

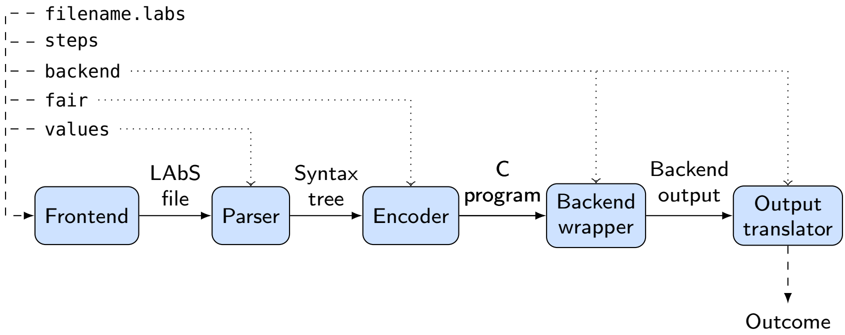
1. Write a specification file (system + properties)
2. Invoke SLiVER with arguments:
   - The name of the file
   - A bound on the number of transitions
   - Other optional arguments
3. Possible outputs:
   - Success
   - Property violated + counterexample
   - Out of time/memory/etc.

SLiVER Architecture



steps	Number of system transitions to bound the verification
backend	Currently: either cbmc, cseq or esbmc (partial support)
fair	Enforce round-robin interleaving of components
values	Key-value pairs for parametric systems

SLiVER Architecture



`steps`	Number of system transitions to bound the verification
`backend`	Currently: either `cbmc`, `cseq` or `esbmc` (partial support)
`fair`	Enforce round-robin interleaving of components
`values`	Key-value pairs for parametric systems

Example: flocking

```
system {
    extern = _birds, _grid, _delta
    spawn = Bird: _birds
    link = ((x of c1 - x of c2) * (x of c1 - x of c2)) +
        ((y of c1 - y of c2) * (y of c1 - y of c2)) <= _delta * _delta

    # "Global" processes
    Movex = x <- (x + dirx) % _grid
    Movey = y <- (y + diry) % _grid

}

comp Bird {
    interface = x: 0.._grid, y: 0.._grid
    stigmergy = <dirx: [-1, 1], diry: [-1,1]>
    behavior = Flock

    # "Local" processes (can only be used by component Bird)
    Flock = (Movex & Movey); Flock
}

check {
    Px = finally exists Bird b1, forall Bird b2, dirx of b1 = dirx of b2
    Py = finally exists Bird b1, forall Bird b2, diry of b1 = diry of b2
}
```

Example: flocking

External parameters

```
system {
    extern = _birds, _grid, _delta
    spawn = Bird: _birds
    link = ((x of c1 - x of c2) * (x of c1 - x of c2)) +
        ((y of c1 - y of c2) * (y of c1 - y of c2)) <= _delta * _delta

    # "Global" processes
    Movex = x <- (x + dirx) % _grid
    Movey = y <- (y + diry) % _grid

}

comp Bird {
    interface = x: 0.._grid, y: 0.._grid
    stigmergy = <dirx: [-1, 1], diry: [-1,1]>
    behavior = Flock

    # "Local" processes (can only be used by component Bird)
    Flock = (Movex & Movey); Flock
}

check {
    Px = finally exists Bird b1, forall Bird b2, dirx of b1 = dirx of b2
    Py = finally exists Bird b1, forall Bird b2, diry of b1 = diry of b2
}
```

Example: flocking

```
system {
    extern = _birds, _grid, _delta
    spawn = Bird: _birds
    link = ((x of c1 - x of c2) * (x of c1 - x of c2)) +
        ((y of c1 - y of c2) * (y of c1 - y of c2)) <= _delta * _delta

    # "Global" processes
    Movex = x <- (x + dirx) % _grid
    Movey = y <- (y + diry) % _grid

}

comp Bird {
    interface = x: 0.._grid, y: 0.._grid
    stigmergy = <dirx: [-1, 1], diry: [-1,1]>
    behavior = Flock

    # "Local" processes (can only be used by component Bird)
    Flock = (Movex & Movey); Flock
}

check {
    Px = finally exists Bird b1, forall Bird b2, dirx of b1 = dirx of b2
    Py = finally exists Bird b1, forall Bird b2, diry of b1 = diry of b2
}
```

System composition

Example: flocking

```
system {
    extern = _birds, _grid, _delta
    spawn = Bird: _birds
    link = ((x of c1 - x of c2) * (x of c1 - x of c2)) +
        ((y of c1 - y of c2) * (y of c1 - y of c2)) <= _delta * _delta

    # "Global" processes
    Movex = x <- (x + dirx) % _grid
    Movey = y <- (y + diry) % _grid

}

comp Bird {
    interface = x: 0.._grid, y: 0.._grid
    stigmergy = <dirx: [-1, 1], diry: [-1,1]>
    behavior = Flock

    # "Local" processes (can only be used by component Bird)
    Flock = (Movex & Movey); Flock
}

check {
    Px = finally exists Bird b1, forall Bird b2, dirx of b1 = dirx of b2
    Py = finally exists Bird b1, forall Bird b2, diry of b1 = diry of b2
}
```

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \leq \delta$$

Example: flocking

```
system {
    extern = _birds, _grid, _delta
    spawn = Bird: _birds
    link = ((x of c1 - x of c2) * (x of c1 - x of c2)) +
        ((y of c1 - y of c2) * (y of c1 - y of c2)) <= _delta * _delta

    # "Global" processes
    Movex = x <- (x + dirx) % _grid
    Movey = y <- (y + diry) % _grid

}

comp Bird {
    interface = x: 0.._grid, y: 0.._grid
    stigmergy = <dirx: [-1, 1], diry: [-1,1]>
    behavior = Flock

    # "Local" processes (can only be used by component Bird)
    Flock = (Movex & Movey); Flock
}

check {
    Px = finally exists Bird b1, forall Bird b2, dirx of b1 = dirx of b2
    Py = finally exists Bird b1, forall Bird b2, diry of b1 = diry of b2
}
```

Nondet. initial state

Example: flocking

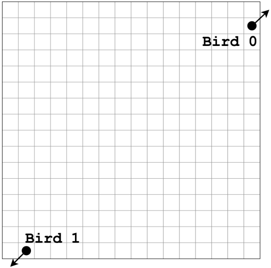
```
system {
    extern = _birds, _grid, _delta
    spawn = Bird: _birds
    link = ((x of c1 - x of c2) * (x of c1 - x of c2)) +
        ((y of c1 - y of c2) * (y of c1 - y of c2)) <= _delta * _delta

    # "Global" processes
    Movex = x <- (x + dirx) % _grid
    Movey = y <- (y + diry) % _grid

}

comp Bird {
    interface = x: 0.._grid, y: 0.._grid
    stigmergy = <dirx: [-1, 1], diry: [-1,1]>
    behavior = Flock

    # "Local" processes (can only be used by component Bird)
    Flock = (Movex & Movey); Flock
}

check {
    Px = finally exists Bird b1, forall Bird b2, dirx of b1 = dirx of b2
    Py = finally exists Bird b1, forall Bird b2, diry of b1 = diry of b2
}
```

Tuple (will be treated as a single stigmergy entry)

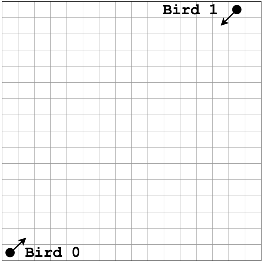
Flocking: counterexample

grid=16, birds=2, delta=21

```
Bird 0: x <- 15
Bird 0: y <- 14
Bird 0: dirx <~ 1 (0)
Bird 0: diry <~ 1 (1)
Bird 1: x <- 1
Bird 1: y <- 0
Bird 1: dirx <~ -1 (2)
Bird 1: diry <~ -1 (3)
--step 0--
Bird 1: x <- 0
--step 1--
Bird 0: y <- 15
--step 6--
Bird 1: x <- 15
--step 7--
Bird 0: x <- 0
--step 11--
Bird 1: y <- 15
--step 13--
Bird 0: y <- 0
--step 19--
Bird 1: x <- 14
--step 20--
Violated property: Px
```

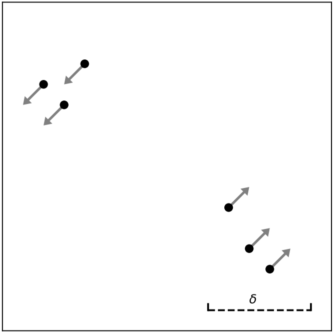


Initial state



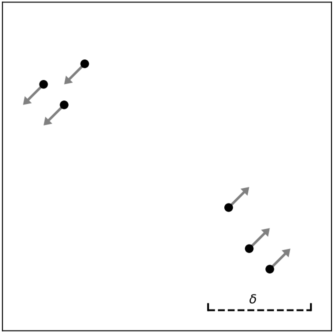
Final state

Flocking: another counterexample



- Difficult to find through simulation
- Actually falsifies the desired property for *any* number of transitions

Flocking: another counterexample



- Difficult to find through simulation
- Actually falsifies the desired property for *any* number of transitions

Parallel analysis with CSeq

- SAT does not always benefit from parallelism
- In our encoding some SAT variables have special meaning
- Array `choice`: scheduling of components and system transitions
- CSeq[6] *splits* the analysis based on the possible values of `choice`
- System specs:
  - 64-bit GNU/Linux (4.9.95 kernel)
  - 128GB RAM
  - Dual 3.10GHz Xeon E5-2687W 8-core CPU

Agents	Steps	Result	CBMC	CSeq - Parallel
2	12	Pass	2' 22"	30"
3	18	Pass	26' 43"	7' 27"
3	20	Pass	55' 49"	19' 19"
4	22	Fail	3h 31' 29"	31' 28"

[6] O. Inverso, T. L. Nguyen, B. Fischer, S. L. Torre, and G. Parlato, "Lazy-CSeq: A Context-Bounded Model Checking Tool for Multi-threaded C-Programs," ASE, 2015.

Conclusions

- Stigmergy can capture relations between individual and collective behaviour of a system
- A simple process algebra for MASs with virtual stigmergy
  - Intuitive design of individual behaviour
  - Generalize interaction by means of attribute-based predicates
  - Investigate the power of stigmergic interaction
  - Amenable to automatic verification

- Mechanize LAbS-to-C encoding
- Verify the C translation with off-the-shelf tools (*backends*)
- Preliminary results show relevant speedup from parallelization

Conclusions

- Stigmergy can capture relations between individual and collective behaviour of a system
- A simple process algebra for MASs with virtual stigmergy
  - Intuitive design of individual behaviour
  - Generalize interaction by means of attribute-based predicates
  - Investigate the power of stigmergic interaction
  - Amenable to automatic verification

- Mechanize LAbS-to-C encoding
- Verify the C translation with off-the-shelf tools (*backends*)
- Preliminary results show relevant speedup from parallelization

Future work on SLiVER

- Improve analysis efficiency
  - Extended backend support
  - Exploit structure (State space reduction)
- Extend analysis tractability
  - Unbounded steps (Completeness threshold, k-induction)
  - Unbounded agents (Cutoff techniques)
- Feature improvements
  - Simulation and statistical model checking
  - Integration with ROS platform

Future work on SLiVER

- Improve analysis efficiency
  - Extended backend support
  - Exploit structure (State space reduction)
- Extend analysis tractability
  - Unbounded steps (Completeness threshold, k-induction)
  - Unbounded agents (Cutoff techniques)
- Feature improvements
  - Simulation and statistical model checking
  - Integration with ROS platform

Thank you!