IMT SCHOOL FOR ADVANCED STUDIES LUCCA

From KLAIM to AbC and Beyond

Rocco De Nicola

Partially funded by MIUR project
PRIN 2017FTXR7S *IT MATTERS*

Pisa - July 2022

IMT- School for Advanced Studies - Lucca

IMT SCHOOL FOR ADVANCED STUDIES LUCCA

Outline

1. Languages for supporting the engineering of different classes of modern distributed systems
   - network-aware programming
   - service-oriented computing
   - autonomic computing.

Programming collective adaptive systems

2. AbC: A calculus for Attribute based Programming
   - Syntax and Semantics
   - Implementations
   - Verification

Agent Based Modelling

3. LABS: A Language with Attribute-based Stigmergies
   - Syntax
   - Implementations
   - Verification

Why a Languages based Approach

Languages

Languages play a key role in the engineering of systems

- Systems must be specified as naturally as possible
- Distinctive aspects of the domain need to be first-class citizens
- Intuitive/concise specifications are possible and encodings can be avoided

Models

Models strictly related to languages are at least as important for effective analysis

- high-level abstract models guarantee feasible investigations
- the scrutiny of results (e.g., counterexample) based on system features, rather than on their low-level representation, guarantees better feedbacks.

Language-based methodology

Major challenge

The big challenge for language designers is to devise appropriate abstractions and linguistic primitives to deal with the specificities of the systems under consideration while relying on an appropriate semantic model.

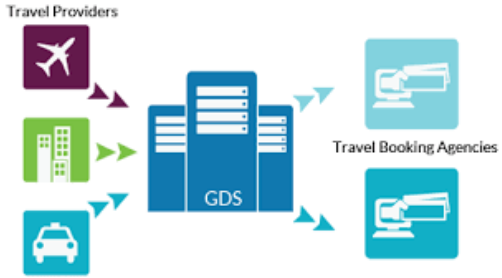
A possible approach

Combined use of formal methods with model-driven software engineering. Key ingredients are

1. A specification language equipped with a formal semantics
2. A programming framework with associated runtime environment
3. A number of verification techniques and associated tools

Our Contributions: A timeline

Service-oriented computing
- services composition
- heterogeneous components
- code reuse
- interoperability



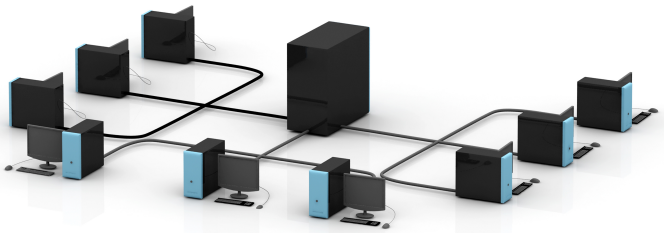
Collective adaptive systems progr.
- large number of components
- decentralised control
- unpredictable environment
- emergent behaviour

2006-2009 SCC - COWS - CASPIS

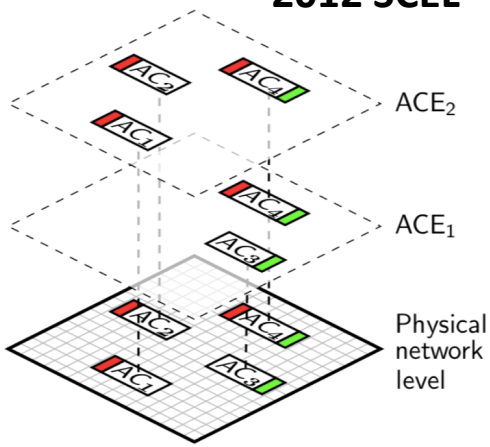
2016 AbC

1998 Klaim

2012 SCEL



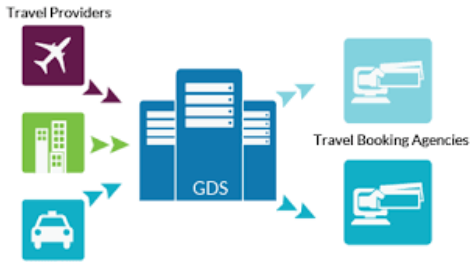
Network-aware programming
- awareness of the network infrastructure
- asynchronous interactions
- open-ended non-determ. environment
- computation mobility



Autonomic computing
- reduced maintenance cost
- no human intervention
- continuous monitoring
- adaptation

Our Contributions: A timeline

Service-oriented computing
- services composition
- heterogeneous components
- code reuse
- interoperability



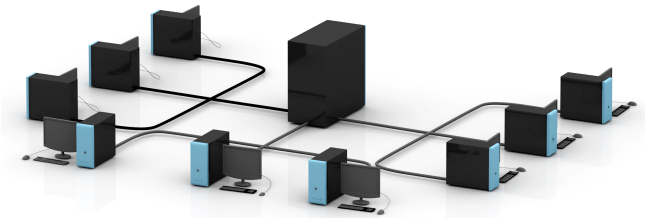
Collective adaptive systems progr.
- large number of components
- decentralised control
- unpredictable environment
- emergent behaviour

2006-2009 SCC - COWS - CASPIS

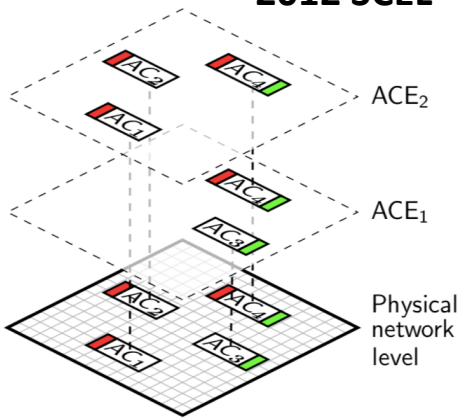
2016 AbC

2020 LABS

1998 Klaim



2012 SCEL



Autonomic Computing with Stigmergic Interaction

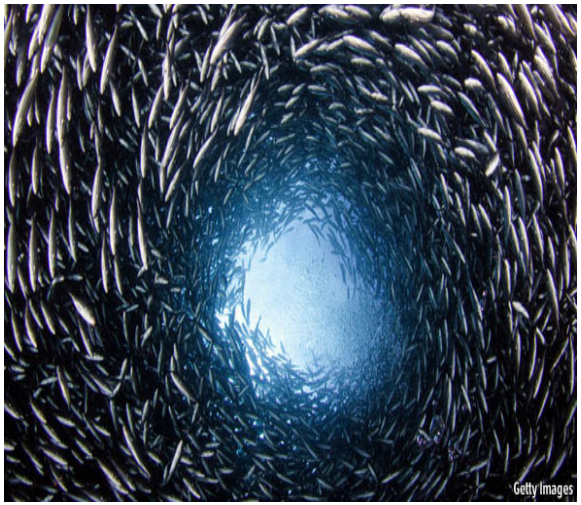
Network-aware programming
- awareness of the network infrastructure
- asynchronous interactions
- open-ended non-determ. environment
- computation mobility

Autonomic computing
- reduced maintenance cost
- no human intervention
- continuous monitoring
- adaptation

Collective Adaptive Systems

We are surrounded by examples of collective systems, in the natural world ....

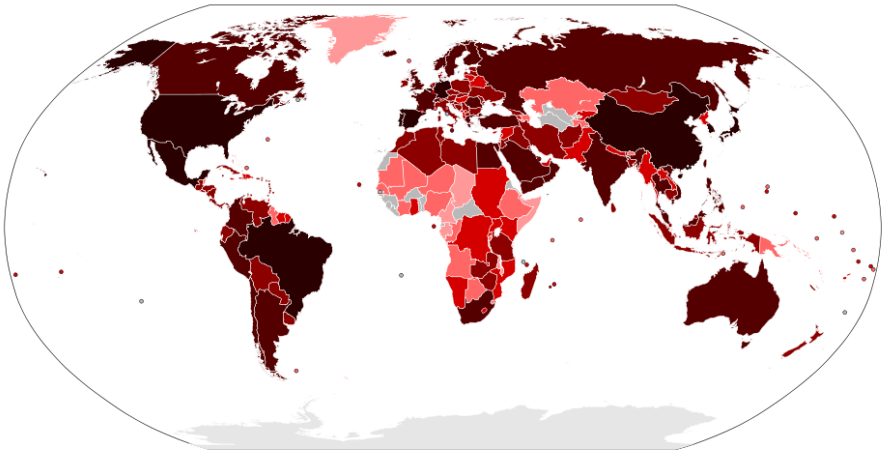
- Bees, Fishes, Birds, ...

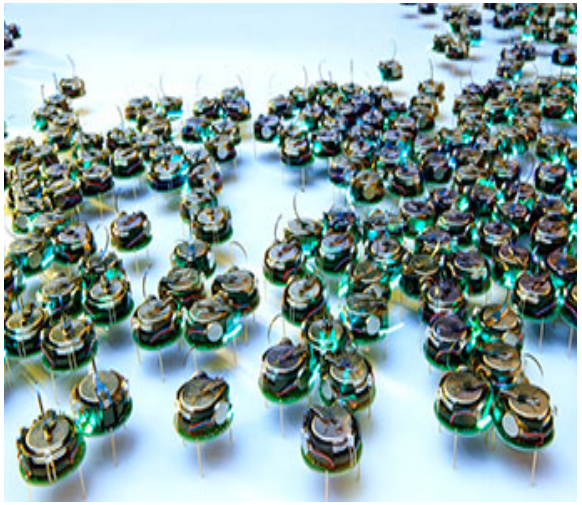


Collective Adaptive Systems

... and in the man-made world

- Traffic, Epidemics, Robots, ...





Collective Adaptive Systems

Many components

From a computer science perspective, collective adaptive systems can be viewed as consisting of a large number of interacting entities.

Local behaviour

Each entity may have its own properties, objectives and actions and at the system level the entities combine to create the collective - emergent - behaviour.

Mutual Influence

The behaviour of the system is dependent on that of the individual entities and the behaviour of the individuals will be influenced by the state of the overall system.

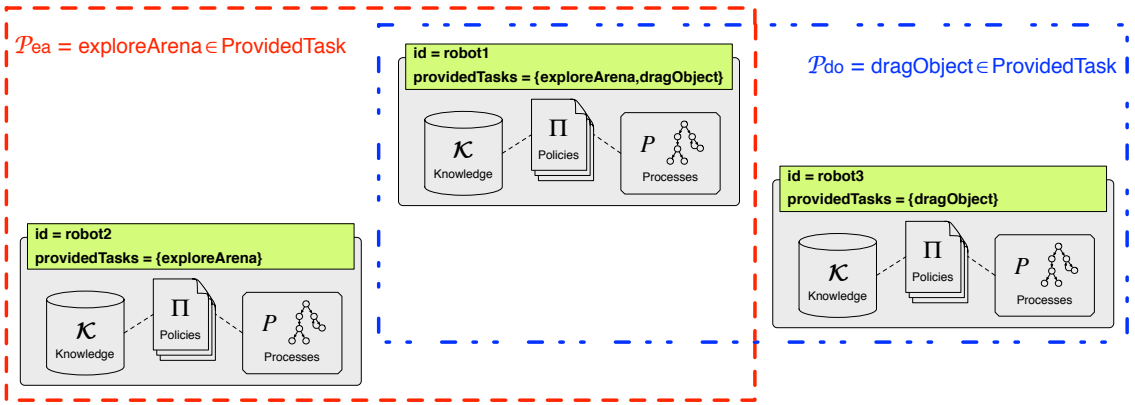
No Central Control

CAS need to operate without centralised control or direction. When conditions within the system change it may not be feasible to have human intervention to adjust behaviour appropriately and systems must autonomously adapt.

The SCEL language: ensembles

Ensembles formation

- Attributes are used by the system components to dynamically organize themselves into ensembles
- Predicates $\mathcal{P}$ over attributes are used by components to specify the targets of communication actions.

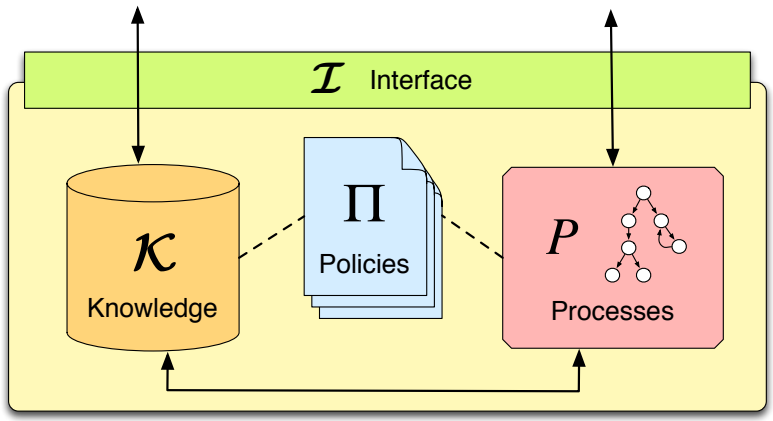


- Ensembles are determined by the predicates validated by each component
- There is no coordinator, hence no bottleneck or critical point of failure
- A component might be part of more than one ensemble

The SCEL language

Introduced to deal with the challenges posed by the design of ensembles of autonomic components

An autonomic component in SCEL:



- Knowledge repositories where components store and retrieve information about their working environment and to use it for determining and adapting their behaviour
- Policies regulating the inter- and intra-components interaction
- Interfaces consisting of a collection of attributes, like provided functionalities, spatial coordinates, group memberships, trust level, response time, ...

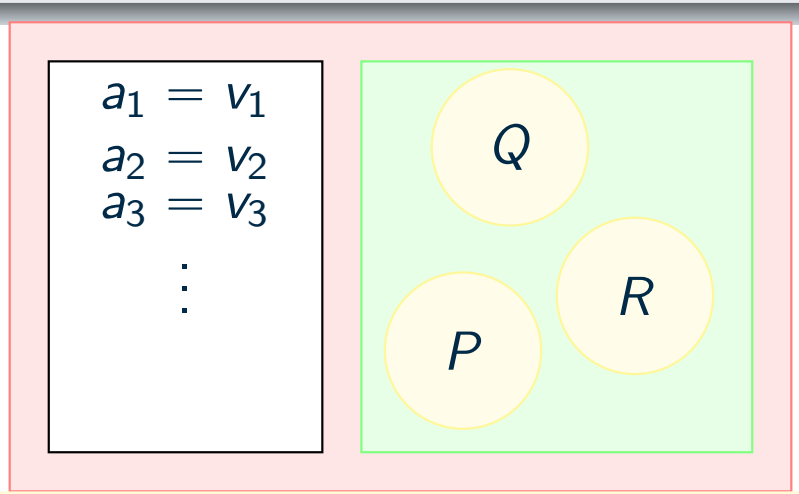
AbC: a calculus distilled from SCEL

- Systems are represented as sets of parallel components, each equipped with a set of attributes whose values can be modified by internal actions.
- Communication actions (send and receive) are decorated with predicates over attributes that partners have to satisfy to make the interaction possible.
- Communication takes place in an implicit multicast fashion: partners are selected via predicates over the attributes exposed in their interfaces.
- Components are unaware of the existence of each other and receive messages only if they satisfy senders requirements.
- Components can offer different views of themselves and can communicate with different partners according to different criteria.
- Semantics for output actions is non-blocking while input actions are blocking: they can take place through synchronization with an available sent message.

AbC: basic ingredients.

An AbC system consists of a set components that contain

- a behaviour - a set of running processes
- an environment - a map from attributes names to values



Processes can:

- send a message to all the components satisfying a given predicate;
- receive a message from a component satisfying a given predicate;
- change the environment;
- wait until a given predicate is locally satisfied.

AbC Syntax

Components	$C ::= \Gamma :_I P \quad \mid \quad C_1 \| C_2 \quad \mid \quad [\ C\ ]^{\lhd f} \quad \mid \quad [\ C\ ]^{\rhd f}$
Processes	$P ::= 0 \quad \mid \quad \Pi(\tilde{x}).U \quad \mid \quad (\tilde{E})@\Pi.U \quad \mid \quad \langle \Pi \rangle P \quad \mid \quad P_1 + P_2 \quad \mid \quad P_1 \mid P_2 \quad \mid \quad K(x_1, \ldots, x_n)$
Updates	$U ::= [a := E]U \quad \mid \quad P$
Predicates	$\Pi ::= \mathsf{tt} \quad \mid \quad \mathsf{ff} \mid p_k(E_1, \ldots, E_k) \mid \Pi_1 \wedge \Pi_2 \mid \Pi_1 \vee \Pi_2 \mid \neg \Pi$
Expressions	$E ::= v \quad \mid \quad x \quad \mid \quad a \quad \mid \quad this.a \quad \mid \quad o_k(E_1, \ldots, E_k)$

AbC: Interfaces

A basic component, $\Gamma :_I P$, is a process P associated with an attribute environment Γ, and an interface I.

- The attribute environment $\Gamma : \mathcal{A} \twoheadrightarrow \mathcal{V}$ is a partial map from attribute identifiers with $a \in \mathcal{A}$ to values $v \in \mathcal{V}$ where $\mathcal{A} \cap \mathcal{V} = \emptyset$. A value could be a number, a name (string), a tuple, etc.
- The interface $I \subseteq \mathcal{A}$ consists of a *finite* set of attributes names that are exposed by a component to control the interactions with other components.
- Attributes in I are public, and to those in $dom(\Gamma) - I$ are private.

AbC: Controlling Interaction

Two operators $[\ C\ ]^{\lhd f}$ and $[\ C\ ]^{\rhd f}$ are introduced to restrict information flow. Function f associates a predicate Π to each tuple of values $\tilde{v} \in \mathcal{V}^*$ and attribute environment Γ.

- $[\ C\ ]^{\rhd f}$ is used to restrict the messages that component C can send.
  - When the message outgoes $[\ C\ ]^{\rhd f}$, the target predicate is updated to consider also predicate $\Pi' = f(\Gamma, \tilde{v})$
  - Only components satisfying $\Pi \wedge \Pi'$ will receive the message.
  - To prevent a specific *secret* s from being spread outside C, one can use $f_s(\Gamma, \tilde{v}) = \mathrm{tt}$ if $s \notin \tilde{v}$ and $f_s(\Gamma, \tilde{v}) = \mathrm{ff}$ otherwise.
- $[\ C\ ]^{\lhd f}$ is used to restrict the messages that component C can receive.
  - If a component with public attribute environment Γ sends a message $\tilde{v}$ to components C satisfying Π, only those components in C that satisfy $\Pi \wedge f(\Gamma, \tilde{v})$ are eligible to receive the message.

AbC: Processes

A process P can be the:

- inactive process 0,
- action-prefixed process, $act.U$, where act is a communication action and U is a process possibly preceded by an attribute update,
- self-aware process $\langle \Pi \rangle P$, blocks the execution of P until predicate Π is satisfied within the attribute environment where the process is executing and triggers execution of P when the environment changes and $\Gamma \models \Pi$
- nondeterministic choice between two processes $P_1 + P_2$,
- interleaving composition of two processes $P_1 | P_2$, processes can only communicate indirectly through the attribute environment they share
- parametrised process call with a unique identifier K and a sequence of formal parameters $(x_1, \ldots, x_n)$ used in the process definition $K(x_1, \ldots, x_n) \triangleq P$.

Predicate based communication

Using attributes

- attribute-based output $(\tilde{E})@\Pi$ is used to send the evaluation of the sequence of expressions $\tilde{E}$ to the components whose attributes satisfy the predicate Π.
- attribute-based input $\Pi(\tilde{x})$ is used to receive messages from any component whose attributes (and possibly transmitted values) satisfy the predicate Π; the sequence $\tilde{x}$ acts as a placeholder for received values.
- attribute update $[a := E]$ is used to assign the result of the evaluation of E to the attribute identifier a. Updates are only possible after communication actions: they can be viewed as side effects of interactions. Execution of a communication action and the following update(s) is atomic.

Predicates can refer to public and private attributes of components.

$$(\text{“Req”}, 1, 3)@(\text{i} \geqslant \texttt{this}.\text{i})$$

can be used to send the message $(\text{“Req”}, 1, 3)$ to all components whose attribute i is not less than `this`.i.

Semantics rules: Potential Communications

$$\frac{[\![\tilde{E}]\!]_\Gamma = \tilde{v} \quad \{\Pi_1\}_\Gamma = \Pi}{\Gamma :_I (\tilde{E})@\Pi_1.U \xrightarrow{\Gamma\downarrow I \rhd \overline{\Pi}(\tilde{v})} \{\!|\Gamma :_I U|\!\}} \textsc{Brd}$$

Expressions in $\tilde{E}$ are evaluated to $\tilde{v}$, and the *closure* Π of predicate Π_1 under Γ is computed then $\tilde{v}$, $\{\Pi_1\}_\Gamma$ and $\Gamma \downarrow I$. Environment updates may be applied.

$$\frac{\Gamma' \models \{\Pi_1[\tilde{v}/\tilde{x}]\}_{\Gamma_1} \quad \Gamma_1 \downarrow I \models \Pi}{\Gamma_1 :_I \Pi_1(\tilde{x}).U \xrightarrow{\Gamma' \rhd \Pi(\tilde{v})} \{\!|\Gamma_1 :_I U[\tilde{v}/\tilde{x}]|\!\}} \textsc{Rcv}$$

A message can be received when $\Gamma_1 \downarrow I$ satisfies sender's predicate Π, and the environment of the sender Γ' satisfies the receiving predicate $\{\Pi_1[\tilde{v}/\tilde{x}]\}_{\Gamma_1}$. Updates U under substitution $[\tilde{v}/\tilde{x}]$ may be applied.

Atomicity of Communications and Updates

$$\{\!|C|\!\} = \begin{cases} \{\!|\Gamma[a \mapsto [\![E]\!]_\Gamma] :_I U|\!\} & C = \Gamma :_I [a := E]U \\ \Gamma :_I P & C = \Gamma :_I P \end{cases}$$

Semantics rules: Actual Interactions

$$\frac{C_1 \xrightarrow{\Gamma \rhd \Pi(\tilde{v})} C_1' \quad C_2 \xrightarrow{\Gamma \rhd \Pi(\tilde{v})} C_2'}{C_1 \parallel C_2 \xrightarrow{\Gamma \rhd \Pi(\tilde{v})} C_1' \parallel C_2'}\ \textsc{Sync} \qquad \frac{C_1 \xrightarrow{\Gamma \rhd \overline{\Pi}(\tilde{v})} C_1' \quad C_2 \xrightarrow{\Gamma \rhd \Pi(\tilde{v})} C_2'}{C_1 \parallel C_2 \xrightarrow{\Gamma \rhd \overline{\Pi}(\tilde{v})} C_1' \parallel C_2'}\ \textsc{ComL}$$

- Sync states that C_1 and C_2 can receive the same message.
- ComL governs communication between components C_1 and C_2.

$$\frac{C \xrightarrow{\Gamma \rhd \overline{\Pi}(\tilde{v})} C' \quad f(\Gamma, \tilde{v}) = \Pi'}{[\ C\ ]^{\rhd f} \xrightarrow{\Gamma \rhd \overline{\Pi \wedge \Pi'}(\tilde{v})} [\ C'\ ]^{\rhd f}}\ \textsc{ResO} \qquad \frac{C \xrightarrow{\Gamma \rhd \Pi \wedge \Pi'(\tilde{v})} C' \quad f(\Gamma, \tilde{v}) = \Pi'}{[\ C\ ]^{\lhd f} \xrightarrow{\Gamma \rhd \Pi(\tilde{v})} [\ C'\ ]^{\lhd f}}\ \textsc{ResI}$$

- ResO: if C evolves to C' via $\Gamma \rhd \overline{\Pi}(\tilde{v})$ and $f(\Gamma, \tilde{v}) = \Pi'$ then $[\ C\ ]^{\rhd f}$ evolves via $\Gamma \rhd \overline{\Pi \wedge \Pi'}(\tilde{v})$ to $[\ C'\ ]^{\rhd f}$.
- ResI: $[\ C\ ]^{\lhd f}$ will receive $\tilde{v}$ and evolve to $[\ C'\ ]^{\lhd f}$ with a label $\Gamma \rhd \Pi(\tilde{v})$ only when $C \xrightarrow{\Gamma \rhd \Pi \wedge \Pi'(\tilde{v})} C'$ where $f(\Gamma, \tilde{v}) = \Pi'$.

Behavioural Theory for AbC

Observable Barbs

Let $C\downarrow_\Pi$ mean that component C can send a message with a predicate $\Pi' \eqsim \Pi$ (i.e., $C \xrightarrow{\nu\tilde{x}\overline{\Pi'}\tilde{v}}$ where $\Pi' \eqsim \Pi$ and $\Pi' \not\eqsim \mathrm{ff}$). We write $C \Downarrow_\Pi$ if $C \rightarrowtriangle^* C' \downarrow_\Pi$.

Barb Preservation

$\mathcal{R}$ is barb-preserving iff for every $(C_1, C_2) \in \mathcal{R}$, $C_1\downarrow_\Pi$ implies $C_2 \Downarrow_\Pi$

Weak Reduction Barbed Congruence Relations

A Weak Reduction Barbed Relation is a symmetric relation $\mathcal{R}$ over the set of AbC-components which is barb-preserving, reduction-closed, and context-closed.

Barbed Bisimilarity

Two components are weakly reduction barbed congruent, written $C_1 \cong C_2$, if $(C_1, C_2) \in \mathcal{R}$ for some weak reduction barbed congruent relation $\mathcal{R}$.

Full Abstraction

Weak Bisimulation

A symmetric binary relation $\mathcal{R}$ over the set of AbC-components is a *weak bisimulation* if and only if for any $(C_1, C_2) \in \mathcal{R}$ and for any λ_1

$$C_1 \xrightarrow{\lambda_1} C_1' \text{ implies } \exists \lambda_2 : \lambda_1 \asymp \lambda_2 \text{ such that } C_2 \stackrel{\hat{\lambda}_2}{\Longrightarrow} C_2' \text{ and } (C_1', C_2') \in \mathcal{R}$$

Two components C_1 and C_2 are weakly bisimilar, written $C_1 \approx C_2$ if there exists a weak bisimulation $\mathcal{R}$ relating them.

Theorem (Soundness)

$C_1 \approx C_2$ implies $C_1 \cong C_2$, for any two components C_1 and C_2.

Theorem (Completeness)

$C_1 \cong C_2$ implies $C_1 \approx C_2$, for any two components C_1 and C_2.

Encoding other paradigms

A number of alternative communication paradigms can be easily modelled by relying on AbC primitives.

Explicit Message Passing

A $b\pi$-calculus process P is rendered as an AbC component $\Gamma{:}P$ where $\Gamma = \varnothing$ and the communication channel is sent as a part of the transmitted values with the receiver checking its compatibility.

Group based Communications

The group name is encoded as an attribute in AbC. The constructs for joining or leaving a given group can be encoded as attribute updates.

Publish-Subscribe

A Publisher sends tagged messages for all subscribers by exposing from his environment only the current topic while subscribers check compatibility of messages according to their subscriptions.

Implementations issues

Many challenges:

- Which kind of Middleware?
  - Centralized?
  - Distributed?
- Whom checks the predicates?
  - the sender?
  - the receiver?
  - a central entities?
- For the moment: four implementations
  - one in Java
  - two in Erlang
  - one in Go

AbC implementations

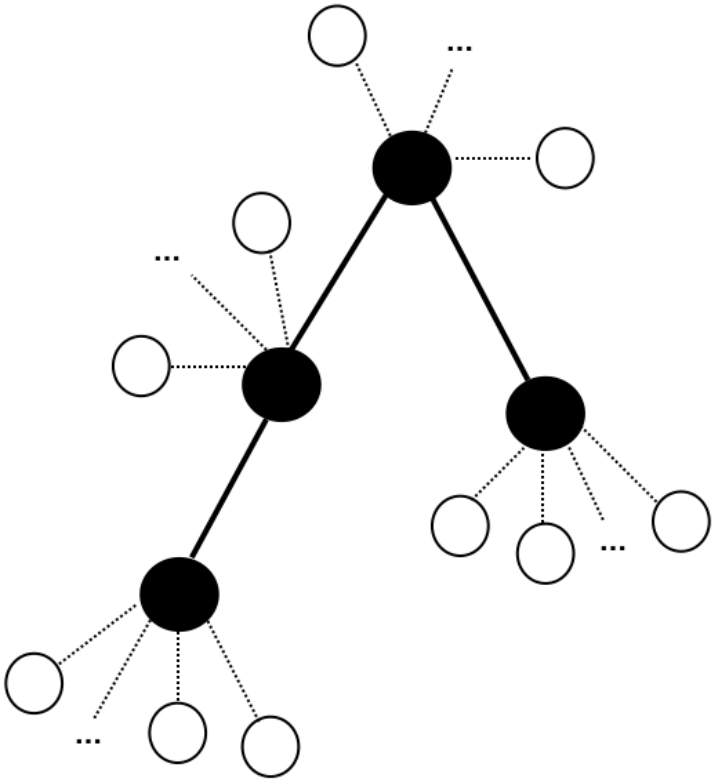
- AbaCus - Java: a centralized broker, broadcast, missed performance evaluation [ISOLA'16]
- AErlang - Erlang: a centralized broker with different dispatching policies [COORD'17]
  - Broadcast: Receivers checks both sending and receiving predicates
  - Push: broker checks sending predicates, receivers check receiving predicates
  - Pull: broker checks receiving predicates, receivers check sending predicates
  - Push-pull: broker checks both sending and receiving predicates

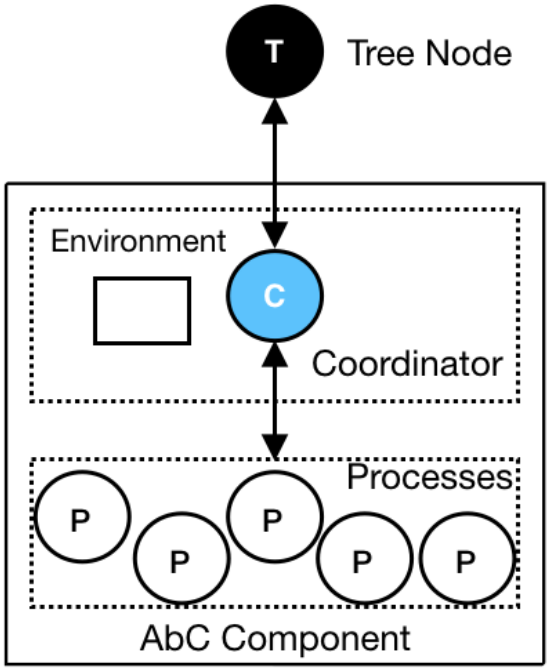
  dynamically handling messages, good performance, **deviated** semantics
- GoAt - Go: a set of broker connected in different shapes [ISOLA'18]
  - Semantics preserving implementation
  - Performance evaluation showed a tree-based structure performs best
  - However, deriving Goat code from AbC code is not immediate

ABEL - A programming framework for AbC

ABEL - Erlang is a recent implementation of AbC

- Providing Inter-coordinators (tree-based) and intra-coordinators interaction
- Supporting total-ordering and relaxed ordering of message delivery



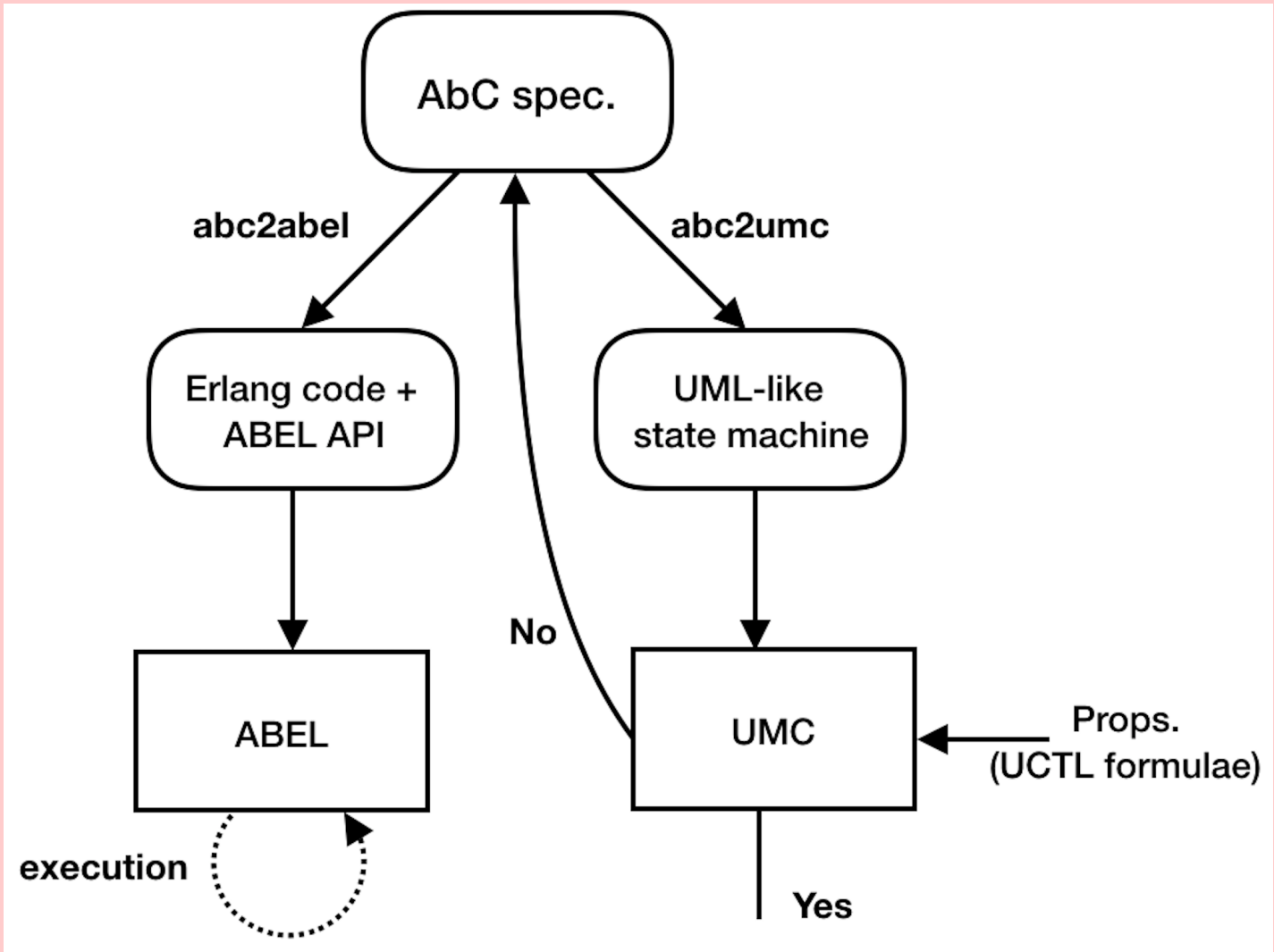


ABEL - A programming framework for AbC

ABEL API offers a one-to-one correspondence with AbC constructs

$C ::=$	**new_component**$(comp\_name, Env, I)$	Create
	start_component$(C, BRef)$	Start
$BDef ::=$	$proc(\mathrm{C}, \langle vars \rangle) \rightarrow Com.$	Definition
$BRef ::=$	**fun**$(\langle vars \rangle) \rightarrow proc(C, \langle vars \rangle)$ **end**	Reference
	| **nil**	
$Act ::=$	$\{\langle g \rangle, m, s, \langle u \rangle\}$	Output
	$\mid \{\langle g \rangle, r, \langle u \rangle\}$	Input
$Com ::=$	**prefix**$(\mathrm{C}, \{Act, BRef\})$	Prefix
	| **choice**$(\mathrm{C}, [\{Act, BRef\}])$	Choice
	| **parallel**$(\mathrm{C}, [BRef])$	Parallel
	| **call**$(\mathrm{C}, BRef)$	Call

A model-driven approach to AbC programming



An example: Stable Marriage with Attributes

- Match men and women based on their preferences on partner's attributes
  - attributes: agents characteristics
  - preferences: interested values of partners attributes
  - An examples with 2 attributes and 2 preferences

Id	Wealth	Body	Preferences
m1	rich	strong	eyes=amber ∧ hair=red
m2	rich	weak	eyes=green ∧ hair=dark
m3	poor	strong	eyes=green ∧ hair=red
m4	poor	weak	eyes=amber ∧ hair=red

Id	Eyes	Hair	Preferences
w1	amber	dark	wealth=poor ∧ body=weak
w2	amber	dark	wealth=rich ∧ body=strong
w3	green	red	wealth=rich ∧ body=strong
w4	green	dark	wealth=rich ∧ body=weak

- Man: iteratively proposes while gradually relaxing expectations (predicates)
- Woman: performs "select and swaps"

SMA properties checking with UMC

We verified for all input spaces of problems of size of 2

- Termination - `True`
- Soundness of outcomes:
  - completeness - `True`
  - symmetry - `True`
  - uniqueness - `False`
- Liveness properties:
  - If a woman sends 'yes' she will eventually receive a 'toolate' or 'confirm' message - `True`
  - If a man receives a 'split', he will eventually send a new proposal - `False` (he may immediately receive another 'yes', and settle down)

Agent-based modelling (ABM)

In ABMs, complex collective phenomena are seen as emergent , i.e. as the result of individual behaviour + interaction. There are two main approaches:

Bottom-up approach

1. Describe the behaviour of the *individual agents*
2. Put many agents together (e.g., by building a simulation)
3. See what happens

Top-down approach

- Dynamical systems (population dynamics)
- Dynamic stochastic general equilibrium (economics)
- Fluid dynamics (traffic networks)

Insights

Yet a new *language* that naturally capture MASs

Domain-specific Provide appropriate abstractions and constructs

Flexible General enough to describe many scenarios

Rigorous Formally specified semantics

With accompanying *techniques and tools* to formally verify MASs

- Exploit (& evolve with) the state of the art
- Avoid being tied to one single verification technique
- Push-button analysis and verification

Background: Virtual stigmergies (1/2)

Stigmergy

- Stigmergy = Communication mediated by the environment
- Virtual Stigmergy = Communication mediated by a distributed data structure

Activity

- Agents *assign* a value v to a variable x
  - A *timestamp* t is retrieved from a clock
  - (x, v, t) is stored locally (i.e., in the agent's memory)
- Agents *use* a variable x in an expression *expr*
  - The agent retrieves the local value of x
  - The local value is used to evaluate *expr*

Questions

1. Where is the communication?
2. Why the timestamps?

Background: Virtual stigmergies (2/2)

Asynchronously, each agent

- *Propagates* (x, v, t) to neighbours after an assignment to x
- *Queries* neighbours "do you have something newer than (x, v, t)?" after using x (*confirmation*)

Newer (x, v', t') is newer than (x, v, t) when $t' > t$

Neighbourhood via a link predicate ... more on this later ...

Agents react to these messages by

- Updating and propagating local value (when receiving a newer value)
- Propagating their up-to-date value (in reply to queries)

A Language with Attribute based Stigmergy

- LAbS: A Language with Attribute-based Stigmergies
- No direct communication
- A system can contain multiple virtual stigmergies, each containing multiple variables
- "Neighbourhood" defined by a predicate *link* over attributes:

$$link(a_1, a_2) = true \iff a_1, a_2 \text{ are neighbours}$$

- Each stigmergy may have a different definition of neighbourhood, which applies to all its variable
- Also supports shared variables (*environment*)

LAbS assignments

Basic building blocks of an agent's behaviour
Each agent may perform assignments to variables:

$$x \leftarrow e \quad \text{Local}$$
$$x \Leftarrow e \quad \text{Environment}$$
$$x \leftsquigarrow e \quad \text{Stigmergy}$$

(e is an arithmetic expression)
Multiple assignments to variables of the same kind:

$$x, y, z \leftarrow 1, 2, 3$$

LAbS processes

Process = Composition of assignments describing an agent's behaviour
If P and Q are processes, then:

$P; Q$	Do P until it terminates, then do Q (sequence)
$P + Q$	Do either P or Q (choice)
$P \mid Q$	Alternate the execution of P and Q (parallel)

Guarded process:

$g \to P$ Evaluate expression g, then:

- If true, continue as P
- If false, block

Infinite behaviours by defining and referring to *named processes*:

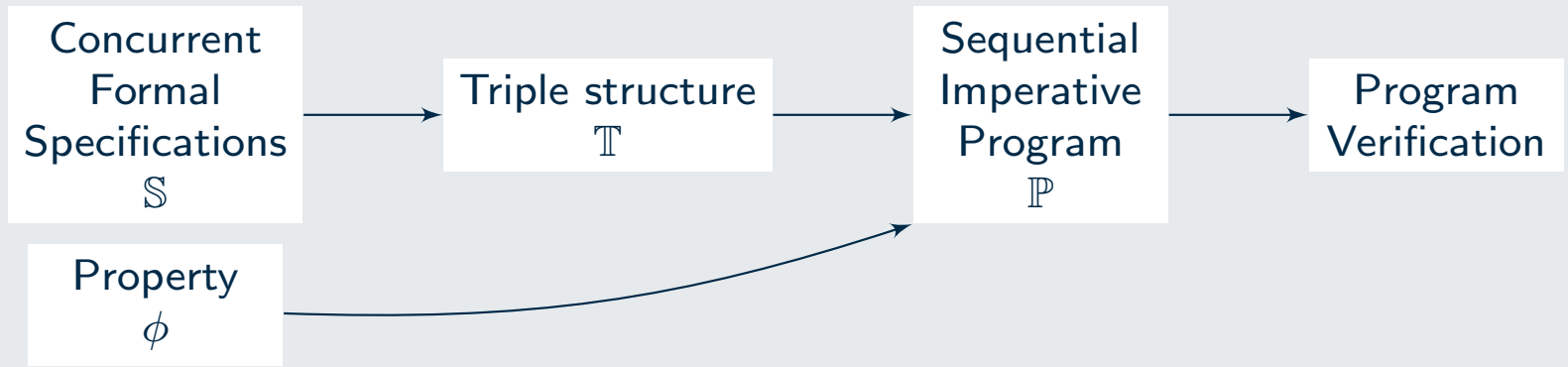
$K \triangleq P; K$ Repeatedly do P

Verification

Given a LAbS system $\mathbb{S}$ and a property ϕ, we would like to generate a program $\mathbb{P}$ such that

$$\mathbb{P} \text{ is successfully verified} \iff \phi \text{ holds in } \mathbb{S}.$$

$\mathbb{P}$ is called an *emulation program* for $\langle \mathbb{S}, \phi \rangle$



Triple structures

- Intermediate representation of a behaviour
- Independent of the specification language

Each elementary action (e.g., assignments) μ is encoded as a triple

$$\langle \triangleright, \mu, \triangleleft \rangle$$

$\triangleright$, $\triangleleft$: predicates over a vector of integers *pc* (*program counter*)

$\triangleright$ *Entry condition:* if satisfied, then μ may be performed

$\triangleleft$ *Exit condition:* when μ is performed, $\triangleleft$ will constrain the future values of *pc*

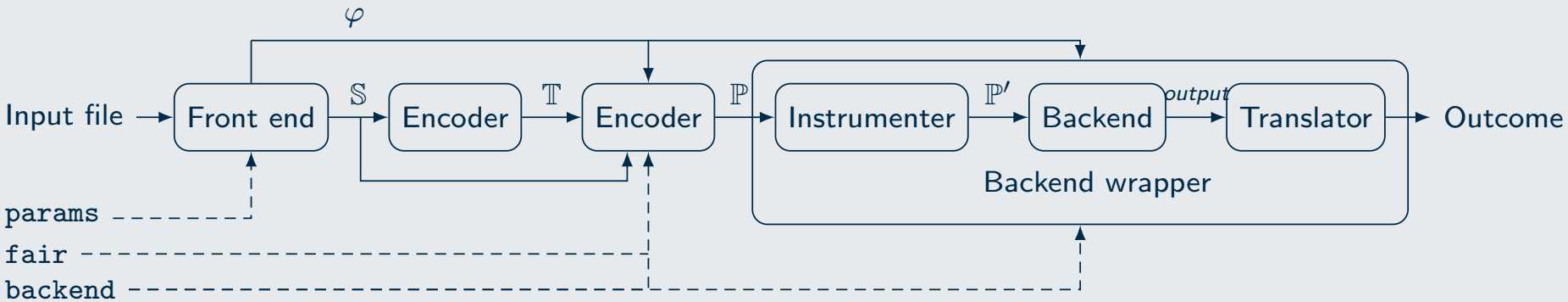
A triple structure $\langle T, pc_0 \rangle$ is a set of triples + an initial value of *pc*

Implementation:

- Sliver: A tool for the verification of Labs systems
- The input language is LabsM, a machine-readable extension of Labs
  - Nondeterministic initialisation of variables
  - Tuples (i.e., composite stigmergic variables)
  - Arrays
  - A simple property language for invariants/emergent properties
  - Parameterised specifications
- The output language is either C or LNT

Sliver Workflow (1/2)

C-based workflow



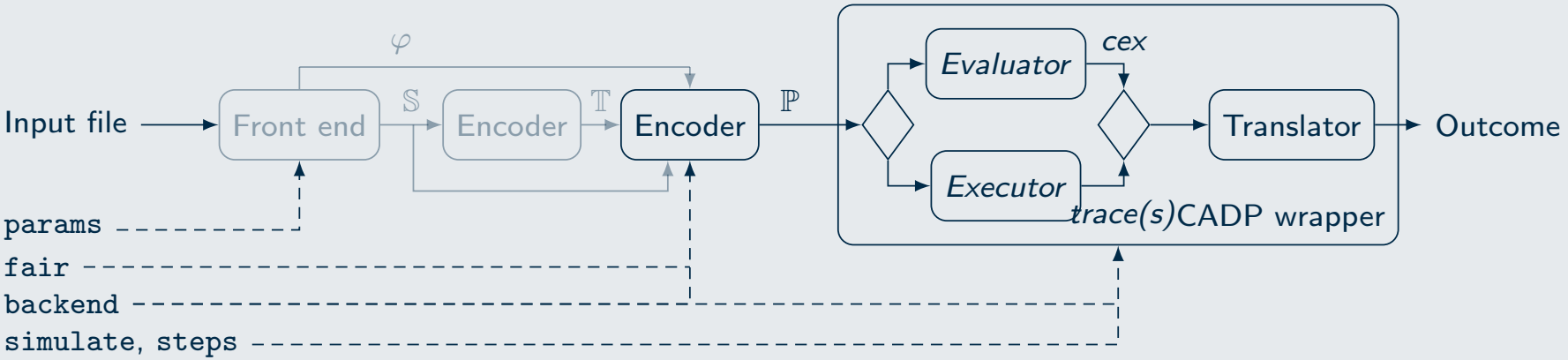
`params` Parameter values

`fair` Changes the behaviour of the `next()` function

- Full interleaving
- Round-robin

Sliver Workflow (2/2)

LNT-based workflow



`simulate` Allows to choose between

- Verifying the system
- Generating n simulation traces

`steps` Max. length of each simulation trace

Experimental evaluation: Examples

To show the feasibility of our approach, we built LabsM specification of several real-world MASs both stigmergy-based and environment-based.

Stigmergy-based systems

`leader` Leader election

`flock` A system of agents that agree to move in the same direction when they are close enough (flocking behaviour)

`boids` Another flocking model, with more complex behaviour

- Agents may be either *leaders* or *followers*
- *Cohesion*: faraway followers move closer to their leader

`formation` A system of agents moving along a segment and trying to maintain a minimum distance from each other

Experimental evaluation: Examples

Environment-based systems

Two *majority protocols*:

- Each agent starts with an opinion (Y or N)
- Agents may change their opinion (or go into "intermediate states") when they meet others
- Suppose that initially a majority of agents is N: then, a protocol is *correct* iff. eventually all agents are N.

`approx` Approximate majority protocol (incorrect)

`maj` A provably correct protocol

Experimental evaluation: Properties

`leader` Eventually, all agents choose 0 for leader

`flock` Eventually, all agents move in the same direction

`boids` Eventually, all agents choose the same leader

`formation`
- Agents never go outside the segment
- Agents eventually stay far apart from each other

`approx` Minority never wins

`maj`
- Minority never wins
- Eventually, Majority wins

Invariance verification tasks and results

We selected 9 tools implementing reachability analysis based on:

- Symbolic execution
- Bounded Model Checking
- Explicit-value analysis
- Predicate abstraction
- Automata-based verification
- k-induction (kI)
- Property directed reachability (PDR)

system $\times$ parameters $\times$ technique $=$ 48 verification tasks:

- At least one conclusive verdict for each system
- All conclusive results were consistent
- kI and PDR were able to complete all tasks with outstanding performance

Invariance verification tasks and results

TABLE 5.1: Results of the *invariance* verification tasks for C emulation programs. $-^a$: Timeout (12 hours). $-^b$: Inconclusive analysis reported by the tool. $-^c$: Out of memory (32 GB). $-^*$: The tool requires an array-free encoding.

Techniques \ Systems	formation	approx-a	approx-b	maj
Symbolic execution (Symbiotic)	0.01 ✓	206.83 ✗	60.47 ✗	$-^a$
Bit-precise BMC (CBMC)	$-^a$	0.01 ✗	0.01 ✗	$-^a$
Word-level BMC (ESBMC)	$-^a$	0.08 ✗	0.07 ✗	$-^a$
Word-level BMC (SMACK)	$-^a$	0.67 ✗	1.83 ✗	$-^a$
Explicit-value analysis (CPAchecker)*	$-^c$	337.08^b	$-^c$	1.03 ✓
Predicate abstraction+CEGAR (CPAchecker)*	0.34 ✓	0.08 ✗	0.03 ✗	$-^c$
Automata+CEGAR (Automizer)	5.98 ✓	1.17 ✗	45.7 ✗	$-^a$
k-induction (CPAchecker)*	$-^c$	0.17 ✗	0.01 ✗	$-^c$
k-induction (2LS)*	0.01 ✓	0.12 ✗	0.05 ✗	$-^a$
k-induction (ESBMC)	0.01 ✓	0.01 ✗	0.05 ✗	0.01 ✓
PDR (Seahorn)	0.3 ✓	0.03 ✗	0.5 ✗	4.67 ✓
PDR (VVT)	0.03 ✓	0.01 ✗	0.01 ✗	0.01 ✓
	✓	✗	✗	✓

Emergence verification tasks and results

3 tools, implementing termination analysis based on:

- Symbolic execution
- Bounded Model Checking with Completeness threshold
- Summarization based on intervals
- Summarization based on equalities

system $\times$ parameters $\times$ technique $=$ 16 verification tasks:

- At least one conclusive verdict for each system except `maj`
- All conclusive results were consistent
- BMC+Completeness verified all systems (except `maj`)

Bibliography

- L. Di Stefano, R. De Nicola, O. Inverso: Verification of Distributed Systems via Sequential Emulation. ACM Trans. Softw. Eng. Methodol. 31(3): 37:1-37:41 (2022)
- R. De Nicola, T. Duong, M. Loreti: Provably correct implementation of the AbC calculus. Sci. Comput. Program. 202: 102567, Elsevier 2021.
- R. De Nicola, G.L. Ferrari, R. Pugliese, F. Tiezzi: A formal approach to the engineering of domain- specific distributed systems. J. Log. Algebraic Methods Program. 111: 100511, Elsevier 2020.
- Y. Abd Alrahman, R. De Nicola, M. Loreti: Programming interactions in collective adaptive systems by relying on attribute-based communication. Sci. Comput. Program. 192: 102428, Elsevier 2020.
- R. De Nicola, L. Di Stefano, O. Inverso: Multi-agent systems with virtual stigmergy. Science of Computer Programming, Volume 187, February 2020. Elsevier 2020.
- Y. Abd Alrahman, R. De Nicola, M. Loreti: A calculus for collective-adaptive systems and its behavioural theory, Info&Co, vol. 268, Elsevier 2019
- Di Stefano's Thesis: https://hdl.handle.net/20.500.12571/10181
- Code: https://github.com/labs-lang

Thank you!