A logic language for run time assessment of spatial properties in self-organizing systems

DE ANGELIS, Francesco, DI MARZO SERUGENDO, Giovanna

Abstract
The assessment of emergent global behaviors of self-organizing applications is an important task to accomplish before employing such systems in real scenarios, yet their intrinsic complexity make this activity still challenging. In this paper we present a logic language used to verify graph-based global properties of self-organizing systems at run-time. The logic language extends a chemical-based coordination model based on logic inference recently proposed. The logic formulae defined by using the language operators depict the intended global spatial properties arising from local interactions among components. Logic formulae are evaluated in a distributed manner by using an inference procedure which checks them against the current global state of the system, verifying whether the intended emergent global behavior actually appears in the system. As examples of spatial properties we consider color patterns: at first we show how to verify specified patterns of identified colors in sets of nodes directly connected, then we present other formulae verifying the appearance of global patterns of colors without specifying the [...]
A logic language for run time assessment of spatial properties in self-organizing systems

Francesco Luca De Angelis  
Institute of Services Science  
University of Geneva  
Carouge 1227 GE, Switzerland  
Email: francesco.deangelis@unige.ch

Giovanna Di Marzo Serugendo  
Institute of Services Science  
University of Geneva  
Carouge 1227 GE, Switzerland  
Email: giovanna.dimarzo@unige.ch

Abstract—The assessment of emergent global behaviors of self-organizing applications is an important task to accomplish before employing such systems in real scenarios, yet their intrinsic complexity make this activity still challenging. In this paper we present a logic language used to verify graph-based global properties of self-organizing systems at run-time. The logic language extends a chemical-based coordination model based on logic inference recently proposed. The logic formulae defined by using the language operators depict the intended global spatial properties arising from local interactions among components. Logic formulae are evaluated in a distributed manner by using an inference procedure which checks them against the current global state of the system, verifying whether the intended emergent global behavior actually appears in the system. As examples of spatial properties we consider color patterns: at first we show how to verify specified patterns of identified colors in sets of nodes directly connected, then we present other formulae verifying the appearance of global patterns of colors without specifying the colors themselves. We conclude the examples with the computation of mathematical functions, like the verification of the existence of a maximum value in a specific node of the system.

I. INTRODUCTION

Formal assessment of emergent global behaviors in self-organizing systems is challenging and usually follows a process that studies the system in several realistic scenarios. The selection of the coordination model used to define the interactions among the entities of the system plays an important role in the engineering process of a self-organizing system, and affects the choice of the middleware and platforms used in the development stages. This means that the adoption of the verification techniques for the assessment of emergent global behaviors has to be performed by taking into account the employed coordination model.

A promising strategy to cope with these problems consists in designing coordination processes with formalisms favoring formal assessments. Following this approach, we have recently proposed a chemical-based coordination model where interactions among system properties are expressed under the form of combinations of logic programs, named Logic Fragments [1]. The features of our model, called Logic Fragment coordination model, are summarized as follows:

- **Injection of ad-hoc chemical laws on-the-fly**: Logic Fragments define ad-hoc coordination laws that can be injected on-the-fly in the shared container of the chemical-based coordination model.

- **Design-time assessment**: Given that they contain logical rules, interactions enforced by Logic Fragments can be formally verified at design time.

- **Run-time assessment of local properties**: Given that Logic Fragments are combinations of logic programs, they can be also used to reason about the state of the individual components of the system at run-time.

In several typologies of self-organizing systems, it is also important to verify the current global state of the system at run-time: this happens for example when some components of the system are in charge of realizing distributed interlock mechanisms, when they have to perform collaborative actions unplanned at design-time or when a formal certification of a spatial pattern is required; in all these cases, to be implemented such tasks require some knowledge about the states of the entities of the system.

In this paper we extend the Logic Fragment coordination model and define a spatial language to verify graph-based spatial properties of self-organizing systems. The language encapsulates Logic Fragments in statements that are evaluated in a distributed manner at run-time, involving several system entities at the same time. The language has several important features: (i) it supports the decomposition of complex global behaviors into several less complex local properties; (ii) it allows the verification of local properties by using powerful non-classical logic languages (used to write logic programs in Logic Fragments) and (iii) it provides a set of spatial and inference operators for automatic aggregation of locally inferred results, which realize inference processes at the global level of the system.

The remaining part of this paper is structured as follows: Section II contains the related work and in Section IV we define our language for the evaluation of spatial properties. In Section V we present two examples concerning the evaluation of spatial patterns and in Section VI we present the future improvements of our work.

II. RELATED WORK

Several coordination models proposed in literature resort to the logic formalization of the coordination laws. The ACLT model [2] represents the ancestor of tuple-based coordination models allowing the injection of logic theories in the shared container; in that case, logic agents can perform deductions on the shared information by accessing the tuple space through a
matching mechanisms based on unification of first-order logic unitary clauses.

The techniques used to formally assess systems global behaviors can be grouped in two macro-categories. The first category is represented by techniques for formal verification at design time; e.g. [3] defines a spatial logic used to formalize topological properties verifiable through a model checker. In [4], an extension of Signal Temporal Logic is proposed to verify properties in continuous-time, discrete-space systems. Design time techniques (e.g. [4]) evaluate formulae offline, resorting to statistical system parameters (such as transition probabilities among spatial locations) usually obtained through stochastic simulations or to numerical integrations.

The second category contains methodologies and techniques used at run time to deduce the emergence of a specific global behavior on the basis of some pattern behaviors appearing in the system (e.g. [5]).

In our approach we assume to have no knowledge about system parameters and formulae are evaluated at run-time through a language with two levels of logic inferences: (i) a local one, at the level of the individual entities of the system (Logic Fragments containing logic programs written in several logic languages); and (ii) a global one, whose spatial and inference operators use the local knowledge generated by Logic Fragments to deduce properties at the global level.

Our language could also be used to empower global-to-local languages (e.g. [6]) enhancing their capabilities with respect to the inference of global properties.

III. Background

The Logic Fragment coordination model [1] (Fig.1(a)) generalizes the chemical-based coordination model [7], by guiding coordination through logic inference. In this model agents communicate indirectly by introducing and modifying tuples (named Live Semantic Annotations - LSAs) in a shared container named LSA tuple space. LSAs can contain two distinct types of information: (i) properties of type \( \text{name}, \text{value}_1, \ldots, \text{value}_n \) representing passive data; and (ii) Logic Fragments, combinations of logic programs defining ad-hoc chemical reactions. In the LSA tuple space, LSAs interact among each others according to predefined chemical laws named Eco-Laws: Logic Fragments are executed by the Logic Eco-Law and further Eco-Laws are used to spread and merge together several LSAs.

A. Logic Fragments

Logic Fragments are combinations of logic programs ([8]), sets of logic formulae expressed in a well-defined logic language (e.g. first-order logic) used to reason about a specific domain.

Example 1:

\[
deliver(x, y) \leftarrow \text{active}(y), \text{mex}(x, y) \\
\text{active}(\text{agentA}) \leftarrow \\
\text{mex}(m, \text{agentA}) \leftarrow
\]

In an hypothetical domain, the first clause of the definite program shown in Example 1 may be interpreted as follows: if there exists a message \((\text{mex}(x, y))\) for the receiver \(y\) with content \(x\) and \(y\) is active (e.g. ready to receive messages) then the message is delivered to the receiver \((\text{deliver}(x, y))\). The remaining clauses are named facts and they state some conditions that are verified in the domain: \(\text{agentA}\) is active \((\text{active}(\text{agentA}))\) and \(m\) is the content of a message for that agent \((\text{mex}(m, \text{agentA}))\). Combining all these clauses, \((\text{deliver}(m, \text{agentA}))\) is inferred. The “execution” of logic programs produces inferred literals either by performing the evaluation of particular semantics or computing inference procedures.

The rationale underlying the usage of logic programs in a coordination model is three-fold: (i) tuples are coordination entities conveying information; (ii) coordination algorithms can be decoupled into two parts, logic (defining what the algorithm has to do) - the logic can be easily expressed by means of combinations of logic program - and control (stating how every single step has to be implemented); (iii) tuple properties \((\text{name}, \text{v}_1, \ldots, \text{v}_n)\) can be treated as input literals \((\text{Name}(\text{v}_1, \ldots, \text{v}_n))\) for logic programs, starting the inference process that realizes a coordination mechanism. Thus, Logic Fragments can be used to formally define ad-hoc chemical reactions evaluated by the Logic Eco-Law.

In the following definitions we assume that \(\text{Prop}, \text{Const}\) and \(\text{Var}\) are finite mutually disjoint sets of relation symbols, constants and variables respectively. If not declared, we assume that variables are depicted with upper case letters and constants with lower case letters. A literal \(P\) is an expression of type \(P(X_1, \ldots, X_n)\) or \(\neg P(X_1, \ldots, X_n)\) where \(P \in \text{Prop}\) and \(X_i \in (\text{Const} \cup \text{Var})\) for \(i = 1, \ldots, n\). A ground literal is a literal without variables. The set of all ground literals w.r.t. a set \(\text{Const}\) set of relation symbols \(\text{Prop}\) is denoted \(\text{G}\). The power set of \(\text{G}\) is depicted \(\mathcal{P}(\text{G})\).

Logic Fragments ([1]) are evaluated w.r.t. the current content \(L\) of the LSA Space through a function \(v_L : \mathcal{L} \to \mathcal{P}(\mathcal{G}) \cup \{\infty\}\), which associates a Logic Fragment with its set of inferred literals (or to \(\infty\) if the constraints to start the evaluations of the programs are not satisfied). The inferred literals are generated as follows (Fig.1(b)):

- A subset of LSAs is converted into facts for logic programs by using user-defined functions named generators.
- Facts are passed to logic programs and a logical inference procedure is executed.
- The inferred literals can be combined with the ones generated by other fragments by using Logic Fragment operators.

![Figure 1: Logic Fragment Coordination Model](image-url)
• The final inferred literals are then injected in the container under the form of LSAs.

IV. SPATIAL LANGUAGE

The spatial language we present integrates Logic Fragments in logic formulae enriched with spatial operators; the goal is to distribute Logic Fragments over the nodes of the network, reasoning on their consequents at a local level to deduce spatial properties of the whole system.

Definition 1 (Grammar for spatial statements)

\[ SS ::= (LF) \mid (SF) \]
\[ SF ::= (IO) \mid (SO) \mid \neg(SF) \]
\[ IO ::= \text{Inf}(T) \mid \text{Inf}(T), (QI) (P) \]
\[ QI ::= \forall \mid \exists \]
\[ SO ::= (OP) (SS) \]
\[ OP ::= \forall \text{ neigh} \mid \exists \text{ neigh} \]

(Spatial statement)

Definition 2 (Unified set)

Given \( S \subseteq P(G) \cup \{\infty\} \) and \( T \) a literal, a unified set w.r.t. \( S \) and \( T \) is the set \( U(S,T) \subseteq S \) containing the elements of \( S \) that unify with \( T \). A unified set induces a set \( C_{X_1}, ..., C_{X_n} \), where for \( 1 \leq i \leq n, X_i \) is a variable of \( T \) and \( C_{X_i} \) is the set of constants to which \( X_i \) can be bound (i.e. the values substituted to \( X_i \) by all unifiers).

Definition 3 (Nesting level of spatial formulae)

Given a spatial statement \( SS \) and a spatial formula \( SF \) contained in \( SS \), the nesting level of \( SF \) is the difference between the number of left and right parenthesis encountered before \( SF \) reading \( SS \) from left to right.

Example: In \( SS \equiv (LF, \text{Inf}(P(X))) \Rightarrow (\text{Inf}(P(Y))) \), the nesting level of \( \text{Inf}(P(X)) \) is one; and that of \( \text{Inf}(P(Y)) \) is two.

Spatial statements (Def.1) represent the main construct of the spatial language; they are composed of two entities: one Logic Fragment - evaluated on a specific node of the network - and one spatial formula, used to reason about the consequent of the Logic Fragment. A spatial statement \( SS = (LF, SF) \) defines a spatial property to check against the state of the system and the result of its evaluation recursively depends on the evaluation of the spatial formula \( SF \). A spatial formula is a combination of operators connected with logical connectives (\( \land, \lor \) and \( \Rightarrow \)), whose evaluation depends on the evaluation of its inner operators. Several logics are eligible for being employed in the logical evaluation of a spatial formula, such as the classical two-valued logic up to the multi-valued paraconsistent logic described in [9]; this choice is influenced by two factors: (i) the logic language used to define the internal logic programs and (ii) the temporal semantics used to evaluate Logic Fragments whose starting conditions (\( \varphi \) functions) are not satisfied. In this context we adopt the classical two-valued logic for evaluating spatial formulae, leaving the integration with other logics to future work. Spatial statements are evaluated by a special Eco-Law, named Spatial Eco-Law.

A. Semantics of inference operators

We assume that the \( \text{Inf} \) operator is evaluated in a node with LSA Space container \( L \).

\( \text{Inf}(T) \), contained in a spatial formula \( SF \) of a spatial statement \( (LF, SF) \) and \( T \) a literal used as a template: \( \text{Inf}(T) = \text{false} \iff U(V L)(LF), T) = \emptyset \). Otherwise \( \text{Inf}(T) = \text{true} \) and the induced set \( C_{X_1}, ..., C_{X_n} \) is accessible to all inference operators belonging to spatial formulae with nesting level higher than the formula containing \( \text{Inf}(L) \) (i.e. inner spatial formulae). Note that only the induced sets for variables appearing in relation symbols of inner spatial formulae are useful to be really computed; this makes Algorithm 2 reduce the amount of data sent over the network.

Example: We consider an example in which we want to verify if a node satisfies a property of type \( A(X, 4) \), with \( X \) a generic value. The spatial statement used is \( SS \equiv (LF, \text{Inf}(A(X, 4))) \) and we assume \( v_L(LF) \equiv \{A(1, 4), A(2, 4), A(0, 5), B(23)\} \). Then \( U(v_L(LF), A(X, 4)) = \{A(1, 4), A(2, 4)\} \neq \emptyset \), \( \text{Inf}(A(X, 4)) \) is equal to \( \text{true} \) and \( C_X = \{1, 2\} \). The property is then satisfied.

\( \text{Inf}(T, QI \ P) \), contained in a spatial formula \( SF \) of a spatial statement \( (LF, SF) \) and \( T \) a literal used as a template, \( P \) a predicate over a set \( V \subseteq \text{Var} \) and \( QI \in \{\forall, \exists, \exists!\} \). Let \( C_{Y_1}, ..., C_{Y_m} \) the induced sets w.r.t. the unified sets associated with the outer spatial formulae; similarly to the previous case let \( U(v_L(LF), T) \) be the unified set w.r.t. \( v_L(LF) \) and \( T \) and let \( C_{X_1}, ..., C_{X_n} \) the set induced by \( U(v_L(LF), T) \). Depending on the value of \( QI \) there are three possible cases:

I. \( \text{Inf}(L, \forall \ P) = \text{true} \iff \) all assignments of variables \( Y_1, ..., Y_n, X_1, ..., X_m \in (C_{Y_1} \times \ldots \times C_{Y_n} \times C_{X_1} \times \ldots \times C_{X_m}) \) satisfy \( P \).

II. \( \text{Inf}(L, \exists \ P) = \text{true} \iff \) there exists at least one assignment of variables \( Y_1, ..., Y_n, X_1, ..., X_m \in (C_{Y_1} \times \ldots \times C_{Y_n} \times C_{X_1} \times \ldots \times C_{X_m}) \) that satisfies \( P \).

III. \( \text{Inf}(L, \exists! \ P) = \text{true} \iff \) there exists exactly one assignment of variables \( Y_1, ..., Y_n, X_1, ..., X_m \in (C_{Y_1} \times \ldots \times C_{Y_n} \times C_{X_1} \times \ldots \times C_{X_m}) \) that satisfies \( P \).

Example: We consider again the property \( A(X, 4) \), this time adding a constraint on \( X \), that must be less than a variable \( Y \) of an outer formula (not showed). \( X \) and \( Y \) can assume several values, so we require that at least for one couple of assignment the predicate \( X < Y \) holds. We assume that the set of values associated with \( Y \) is \( C_Y = \{2\} \). The first new spatial statement is \( SS_1 \equiv (LF, \text{Inf}(A(X, 4), \exists (X < Y))) \), with \( v_L(LF) \equiv \{A(1, 4), A(2, 4), A(0, 5), B(23)\} \). Like before, we obtain \( U(v_L(LF), A(X, 4)) = \{A(1, 4), A(2, 4)\} \neq \emptyset \) and \( C_X = \{1, 2\} \). The assignment \( X = 1, Y = 2 \) satisfies the predicate \( (1 < 2) \), then \( SS_1 \) is equal to \( \text{true} \) and the property is satisfied. With the same parameters, the second statement \( SS_2 \equiv (LF, \text{Inf}(A(X, 4), \forall (X < Y))) \) is equal to \( \text{false} \) because the assignment \( X = 2, Y = 2 \) does not satisfy the predicate.
B. Semantics of graph-based spatial operators

- $\forall_{\text{neigh}} (LF, SF) = \text{true}$ iff the spatial formula $SF = \text{true}$ in every (direct) neighbor of the node evaluating the spatial statement.
- $\exists_{\text{node}} (LF, SF) = \text{true}$ iff there exists at least one neighbor $N$ of the node evaluating the spatial statement such that $SF = \text{true}$ in $N$.
- $\forall_{\text{node}} (LF, SF) = \text{true}$ iff the spatial formula $SF$ is true for every node on the network.
- $\exists_{\text{node}} (LF, SF) = \text{true}$ iff there exists at least one node $N$ of the network such that the spatial formula $SF$ is true when evaluated in $N$.

C. Well-formed spatial statements

Given that variables are typeless, we assume that if the intended type of variable used in a predicate is not compatible with the constants inferred during the evaluation of the spatial formula then the default value $\text{false}$ is assigned to the operator $\text{Inf}$. In order to avoid meaningless spatial formulae we add additional constraints on the rules of the grammar in Def.1.

**Definition 4 (Well-formed spatial statement)**

A spatial statement $SS$ is well-formed if and only if (i) every predicate in an $\text{Inf}$ operator contains at least one variable and (ii) every variable appearing in the predicate has been defined either in the same $\text{Inf}$ operator or in the one of an outer spatial formula.

Algorithm 1 Evaluation of spatial statements

```plaintext
1: function evalSS(LF, SF, $\emptyset$, null, null, true, false)
2: return $\emptyset$
3: end function
4: function evalSS(LF, SF, indSets, OP, sender, toEval, toSend)
5: $vSF = \emptyset$
6: if toEval = true then
7: $Q = \mathcal{v} (LF)$
8: $vSF = evalSF(SF, Q, indSets)$
9: end if
10: if toSend = true then
11: if OP $\in \{ \exists_{\text{node}}, \forall_{\text{node}} \}$ then
12: $\text{rece} = \text{neighbors} \setminus \{\text{sender}\}$
13: else
14: $\text{rece} = \text{neighbors}$
15: end if
16: send to $\text{rece}$ (OP, LF, SF, indSets)
17: return $\text{waitForValues}(OP, SF, \text{rece}, vSF)$
18: end if
19: return $vSF$
20: end function
```

Algorithm 2 Evaluation of Spatial Formulae

```plaintext
1: function evalSF(SF, $Q$, indSets)
2: if SF is a spatial operator $OP(LF_1, SF_1)$ then
3: if OP $\in \{ \exists_{\text{node}}, \forall_{\text{node}} \}$ then
4: return evalSS(LF_1, SF_1, indSets, OP, null, true, false)
5: else
6: return evalSS(LF_1, SF_1, indSets, OP, null, false, true)
7: end if
8: end if
9: valuesInf = evalOpInf(SF, $Q$, indSets);
10: listSF = getListSpatialFormulae(SF);
11: valuesSF = $\emptyset$
12: for each element $SF_i \in \text{listSF}$ do
13: if canShortcut(SF, valuesInf, valuesSF) then
14: return true
15: else
16: valuesSF = aggregateValues(SF_i, Q, indSets)
17: add valuesSF to valuesSF
18: end if
19: end for
20: return $\mathcal{v}SF = \text{aggregateValues}(SF, valuesInf, valuesSF)$
21: return $\mathcal{v}SF$
22: end function
```

Algorithm 3 When a message is received from the network

```plaintext
1: function receiveMessage(OP, LF, SF, indSets, sender $S$)
2: $SS$ is the sender of the current message
3: if OP = (LF, SF) already received or sent then
4: send to $S$ (discard)
5: else
6: if OP $\in \{ \exists_{\text{node}}, \forall_{\text{node}} \}$ then
7: $\text{valuesSS} = \text{evalSS}(LF, SF, \text{indSets}, OP, S, true, false)$
8: else
9: $\text{valuesSS} = \text{evalSS}(LF, SF, \text{indSets}, OP, S, true, true)$
10: end if
11: send to $S$ (valuesSS)
12: end if
13: end function
```

D. Evaluation of spatial statements and temporal semantics

The evaluation of spatial statements has to take into account the following system parameters:

- Network topology can be static (fixed nodes) or dynamic (mobile nodes).
- Spatial properties can be topology-dependent or topology-independent.
- Spatial properties can be time-dependent (dynamic) or time-independent (once defined their evaluation does not change).

In this work we consider time-independent spatial properties with static topologies. We observe that spatial properties change because of the evolution of the content $L$ of the generic node of the network over time, which entails a change in the evaluation $\mathcal{v}_t$ of the Logic Fragments in the spatial statements. Depending on the assumptions on the delays for transferring information between two nodes, verifying time-dependent properties in networks with mobile nodes is much more challenging and we will analyze them in future work.

**Time-independent properties**: from the observation above, to have time-independent properties it is sufficient that the evaluation of every Logic Fragment (of the spatial statement associated with the property) remains constant in the temporal interval needed to evaluate the whole spatial statement. This
Algorithm 4 Wait for values to aggregate logically

1: function WAITFORVALUES(OP, SF, receivers, defaultResp)
2:     responses = ∅
3:     if defaultResp ≠ null then
4:         responses = responses ∪ {defaultResp}
5:     end if
6:     loop
7:         wait for response for SF from receiver R ∈ receivers
8:         responses = responses ∪ {response}
9:     end if
10:     if received responses from all receivers then break loop
11:     end loop
12:     if ∃ neigh then
13:         if ∃ true ∈ responses then return true
14:         else return false
15:     end if
16:     else
17:         if ∃ false ∈ responses then return false
18:         else return true
19:     end if
20: end if
21: end function

A. Verifying properties in neighbors

In this first example we consider the sensor network in Fig.3: node A wants to verify whether, for every one of its neighbors, there exists at least one neighbor that has detected an event e. To perform the event detection we use the Logic Fragment LF = le, which infers one literal of the form Event(x), meaning that the node has detected the event x. We also assume that node C is the only one that detected e. The spatial statement injected by A is the following one:

\[
SS = \left(\bigvee_x \bigvee_{neigh} (\exists_{neigh} (le, Inf(Event(e))))\right)
\]

The evaluation of SS on A starts with the computation of the Logic Fragment LF = ∅ (whose consequent is the empty set); the same node has also to evaluate the spatial operator \(\bigvee_{neigh} f_{fa}\). To compute it, A sends \(f_{fa}\) to its neighbor B (Fig.3(a)). To compute \(f_{fa}\), B needs to evaluate LF = ∅ and LF = \(\exists_{neigh} f_{fa}\), so it sends \(f_{fa}\) to its neighbors. Nodes A, C and D compute le, verify whether Event(e) is inferred and send back their responses (respectively false, true, false) to B (Fig.3(b)). \(f_{fa}\) is evaluated to true on B because of the response sent by C; B sends then the response true to A, which finally evaluates SS to true because B is its only existing neighbor. (Fig.3(c)).

![Figure 2: Evaluation of spatial operators, time-independent topology-dependent spatial properties with static nodes. Red arrows depict negative (false) responses, green ones positive ones (true).](image)

![Figure 3: Evaluation of the spatial statement SS.](image)

V. EXAMPLES

We assume that in every node of the networks presented in the following examples there exists a running instance of the Logic Fragment coordination model. For all the figures, we keep assuming that red arrows depict negative responses, green ones positive ones.
\[ SS \equiv \left( \emptyset, \forall_{\text{nodes}} (\text{Inf}(\text{Col}(y)) \Rightarrow \forall_{\text{neighs}} (\text{Inf}(\text{Col}(y))) \lor \text{Inf}(\text{Col}(y)) \Rightarrow \forall_{\text{neighs}} (\text{Inf}(\text{Col}(y)))) \right) \]

To improve the readability we define \( SS = (\emptyset, \forall_{\text{nodes}} f_a) \), with:

\[ f_a \equiv (\text{Inf}(\text{Color}(y)) \Rightarrow \forall_{\text{neighs}} (\text{Inf}(\text{Color}(y)) \lor \text{Inf}(\text{Color}(y))) \lor \text{Inf}(\text{Color}(y)) \Rightarrow \forall_{\text{neighs}} (\text{Inf}(\text{Color}(y))) \]

The evaluate \( SS \) node \( A \) needs to calculate \( \forall_{\text{nodes}} f_a \); this entails the creation of a tree rooted in \( A \) (blue segments in Fig.4(a)) in which every node computes \( f_a \). To evaluate \( f_a \), every node computes either \( f_b \) or \( f_d \), which require the computation of the spatial operators \( \forall_{\text{neighs}} \) (e.g. nodes \( F \) and \( G \) Fig.4(b)) as performed in the previous example. Once evaluated \( f_b \) or \( f_d \), leaf nodes return the result of \( f_a \) to their parent nodes immediately (Fig.4(c)), whereas intermediary nodes wait for the responses generated by their descendants to aggregate the value of \( f_a \), before passing it to their parent nodes.

(a) Evaluation of \( SS \) on \( A \), (b) Evaluation of \( f_b \) on \( G \) and \( f_d \) implying the evaluation of on \( F \).

(c) Aggregation of the final result for \( f_a \) on \( G \) and \( F \) and responses for \( D \) and \( E \).

(d) Multicolor pattern: every node has neighbors with colors different than itself.

Figure 4: Evaluation of statement \( SS \).

C. Spatial formulae with predicates

In the remaining examples we show the benefits of using predicates in spatial formulae.

- **Color patterns**: we consider again the previous example; if we assume that the colors of the nodes are yellow and purple, then the pattern of of the Section V-B can be verified by using a more compact spatial statement:

\[ SS \equiv (\emptyset, \forall_{\text{nodes}} (\text{Inf}(\text{Col}(X)) \Rightarrow \forall_{\text{neighs}} (\text{Inf}(\text{Col}(Y), \exists(X \neq Y)))) \]

\( X \) is a variable bound to the constant of the \( \text{Col} \) literal inferred in a node, whereas \( Y \) to the color constants of every one of its neighbors. The predicate \( \exists(X \neq Y) \) assures that those colors are different. Relaxing the assumption on the colors, \( SS \) verifies also the existence of other patterns like the one in Fig.4(d).

- **Maximum value**: by adopting a similar strategy, if we define a total order \( \leq \) over colors, a node can easily verify if its color is the “maximum” one in the network resorting to the following spatial pattern:

\[ SS \equiv (\emptyset, \forall_{\text{nodes}} (\text{Inf}(\text{Col}(y)) \Rightarrow (\forall_{\text{neighs}} (\text{Inf}(\text{Col}(Y), \exists(Y \leq X)))) \]

VI. CONCLUSION AND FUTURE WORK

We have presented a logic language to verify graph-based topology-dependent spatial properties in self-organizing systems. The language resorts to the Logic Fragment coordination model, a chemical-based coordination model in which coordination is steered by logic inference. Logic Fragments are combinations of logic programs used to reason about the state of a single node of the network. The language provides a set of spatial operators used to reason about the global state of the system: spatial operators distribute Logic Fragments over the nodes of the network to infer information local to each node; locally inferred data is then logically aggregated at a global level, producing information about the spatial properties of the whole system. We have shown the verification of properties starting by well-defined spatial patterns known \textit{a priori}; in future work we will implement the logic Eco-Law and focus on graph-based time-dependent spatial properties, introducing new operators to deduce the existence of spatial patterns only partially defined by the language; these improvements will pave the way for further inference procedures on spatial statements able to deduce the existence of spatial properties without resorting to (complete) template formulae.

REFERENCES


