CYBER-PHYSICAL CLOUD COMPUTING LAB
UNIVERSITY OF CALIFORNIA, BERKELEY

AN ALGEBRAIC MODEL OF COMPUTATION FOR SYSTEMS WITH DYNAMIC STRUCTURE

Eloi Pereira and Raja Sengupta

Working Papers
CPCC-WP-2012-07-03

CPCC Berkeley

July 2012

AN ALGEBRAIC MODEL OF COMPUTATION FOR SYSTEMS WITH DYNAMIC STRUCTURE

ABSTRACT

We introduce the Structure Model - a formalism for modelling and controlling networked systems with dynamic structure.

The pervasiveness of networked computing devices is raising attention on models of computation which entail the network structure as a first-class concept. Modelling how the network evolves and how entities interact is now paramount for a correct understanding of the overall system and for the design and synthesis of effective controllers.

We address this problem by introducing a formalism for modelling and controlling systems with dynamic structures. The Structure Model (SM) consists of a set of entities, and a set of unary relations and a bynary relation over the set of entities. The unary relations are used to model state properties of the entities while the binary relation is used to model interaction between entities such as communication. The SM can be manipulated using a set of algebraic operators. The dynamics of the structure are modelled using a transition system. The SM is designed to composed with an underlying model that describes the entities behaviour. We present how the structure can be abstracted from an underlying model and how the changes at the structure level can be propagated back to the system. This provides means controlling the system from a structure perspective.

The structure model aims at being agnostic to the underlying model of computation. In this work we compose the structure model with Nancy Lynch's Synchronous Network Model and present a case study using a version of the "agree and pursue" communication and control law.

KEYWORDS

Models of Computation, Dynamic Networks, Control

1 INTRODUCTION

With the advent of the internet and mobile devices computation has been becoming ubiquitous. Consider a smartphone that provides the user with information about the closest open restaurant, or a team of aerial robots that collaborate while searching for a target. In both examples, computing devices interact with their environment and with other computing devices. Thus, *interaction* has becoming a major aspect in system's design. Indeed, some researchers have been claiming that we are facing a paradigm shift on computation: from calculation to interaction [GSW06, Mil06].

A major aspect of interaction is communication. It is paramount to understand how entities share information, what are the constraints introduced by communication infrastructures, and how the topology of such infrastructures varies over time. In this paper we introduce an algebraic model of computation for modelling and controlling discrete-time structures. Fig. 1 depicts our approach.

We assume that the behavior of the system is modelled by a given concurrent Model of Computation (MoC). The structure, denoted by $\mathcal{S}$, is extracted by an abstraction map α from the underlying MoC. A structure controller takes $\mathcal{S}$ and produces a set of controllable events ε_c that steers the system in order to meet a given structure specification. The system can operate in adversarial environment. This is modelled by introducing a set of uncontrollable events Σ_u that can change $\mathcal{S}$ arbitrarily. Changes at the structure level are then mapped back to the underlying MoC by a control map β.

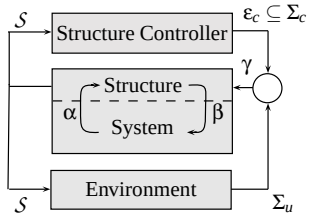


Figure 1: Structure controller architecture.

The literature is rich on concurrent models of computation, e.g. Kahn Process Network (KPN) [KM77], Milner's Calculus of Communicating Systems (CCS) [Mil83], Communicating Sequential Processes (CSP) by Hoare [Hoa78], and the actor model by Hewitt and Agha [Agh86]. While the communication topology on KPN, CCS and CSP does not vary over time, the actor model can model dynamic topologies by changing the set of addresses that each actor knows at each instance of time. Influenced by the idea of dynamic structures Milner, Parrow and Walker introduced π-calculus [MPW89, Mil99]. $\pi-$calculus allows channel names to be communicated through

other channels. This provides means for describing concurrent computations whose structure may change over time. Later, Milner introduced the Bigraph model [Mil09] which also entails changing of structure as a first-class concept using two graph structures that describes location and connectivity of computation.

The main contribution of the work presented in this paper is the introduction of the Structure Model - a formalism for extracting structural information of a system represented by a given model of computation, and for controlling the system from a structural perspective.

We aim at making SM applicable to a wide range of different MoCs that describes underlying behaviour of the system. In this paper we use as a case study the Synchronous Network Model (SNM) by Nancy Lynch [Lyn96]. We choose the SNM because it is a simple model but yet able to model a wide range of interesting concurrent systems.

The remainder of the paper goes as follows. In Section 2 we introduce the formalism for modelling discrete-time structures. Section 3 goes on to explain how a structure can be controlled in a centralized way. In Section 4 we research how to embed state of entities into an SM. In Section 5 we introduce a procedure to abstract the structure from a Synchronous Network Model (SNM) and how to control this SNM from the structure level. Section 6 introduces a case study based on the "agree-and-pursue control and communication law". In Section 7 we state the major conclusions and future work.

2 FORMALISM FOR MODELLING DYNAMIC STRUCTURES

In this section we start by formalizing the SM and then introduce a transition system to model the discrete-time behaviour. Our formalism is based on relational and algebraic structures[1], motivated by the work of Courcelle [Cou91] and Staruch [Sta05]. The former uses a logical approach to formalize graphs, while the later models multi-agent systems using partial algebraic structures.

Definition 1 (Structure Model)**.** *An SM is a relational structure of the form $\mathcal{S} = (Ent, \sigma, I_\sigma)$ where the domain Ent represents a set of entities of the system, σ is the signature of the structure and I_σ is an interpretation function which interprets symbols of the signature on the domain of entities. We denote the set of all possible entities as $\mathcal{E}$.*

Definition 2 (Signature of SM)**.** *The signature of $\mathcal{S}$ is a tuple $\sigma = (\tau_1, \tau_2, \ldots, \tau_i, ed)$ where τ_i are unary relation symbols. Each τ_i has an interpretation $I_\sigma(\tau_i) \subseteq Ent$ where $e \in I_\sigma(\tau_i)$ iff e has the property τ_i. ed is a binary relation symbol with an interpretation $I_\sigma(ed) \subseteq Ent \times Ent$ such that $(e_1, e_2) \in I_\sigma(ed)$ iff entities e_1 and e_2 are related up to some criterion (e.g. communication relation).*

An SM can be seen as a coloured graph where the set of vertices is the domain of $\mathcal{S}$, the colors are defined by the unary relations τ_i, and the edges are defined by the binary relation ed. Note that the signature could be easily enriched with more relation symbols that could represent different structure relations: communication topology, synchronization relations, hierarchical dependencies, etc. For the sake of a clear presentation we choose to keep only one binary relation in the models presented in this paper.

We proceed by formalizing how an SM can evolve over time. Let $\mathbb{S}_\sigma$ be the set of SMs obtained with all possible interpretations I_σ and domains $Ent \subseteq \mathcal{E}$. Let Σ be a set of events that may lead the current structure to change another structure.

Definition 3 (Structure Transition System)**.** *A Structure Transition System (STS) is a many-sorted algebra denoted as $\mathcal{T} = (\mathbb{S}_\sigma, \Sigma, \delta, \mathcal{S}_0)$ where $\delta : \mathbb{S}_\sigma \times \Sigma \to \mathbb{S}_\sigma$ is 2-arity function called structure transition function, and $\mathcal{S}_0 : \{\} \to \mathbb{S}_\sigma$ is a 0-arity function (constant) called initial structure.*

An STS formalizes how an SM evolves from the initial structure $\mathcal{S}_0$ to other another structure in the presence of an event. An execution trace of an STS is represented as a sequence $\mathcal{S}_0 \xrightarrow{\varepsilon_0} \mathcal{S}_1 \xrightarrow{\varepsilon_1} \mathcal{S}_2 \xrightarrow{\varepsilon_2} \ldots$ where $\varepsilon_i \in \Sigma$.

3 CONTROL DESIGN

In this section we formalize a how a STSs can be controlled. Our approach is inspired by the work of Ramadge and Wonham on supervisory control of Discrete Event Systems [RW87, RW89].

Let $\Sigma = \Sigma_c \cup \Sigma_u$ where Σ_c is the set of controllable events while Σ_u is the set of uncontrollable events. The events Σ_c can be disabled by a controller at any time while the set Σ_u denotes the set of events which the controller has no effect.

[1]Note that the term "structure" here refers to the mathematical logic definition: a set of functions and relation symbols interpreted in a given domain.

A control input for $\mathcal{T}$, denoted by γ, is a subset of Σ such that $\Sigma_u \subseteq \gamma$, i.e. all uncontrollable events are included in γ. The set of control inputs, denoted by Γ, is a subset of $2^{\Sigma 2}$.

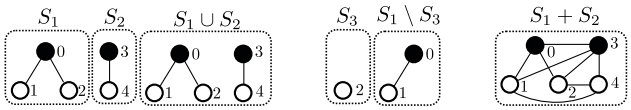
Assume that an SM at a given instance of time entails all the information needed for the controller to decide the next control input γ. Thus we can define a supervisory controller for the structure of the form $ctr : L \to \Gamma$ where $w \in L$ is a sequence of $\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2, \ldots$ with $\mathcal{S}_i \in \mathbb{S}$. w represents the sequence of SMs from the initial step until the current step. We assume an interleaving semantics for the interactions between the plant and the controller, i.e. the controller performs an action between two plant executions.

On Discrete Event Systems, the domain of a model is a set of primitive states, i.e. there aren't any signature on a given state. The domain of an STS are SMs that encapsulate information such as properties of entities (unary relations τ_i) or relations between entities (binary relation *ed*). We introduce an algebra that aims at making the manipulation of SMs explicit. This algebra is used for the specification of the structure transition function of STSs. The algebra is inspired by Algebraic Graph Theory [BW04]. In this section we introduce three different operators. At this stage of the work it is not clear yet if these operators are sufficient for modelling the desired structure-level controllers.

Definition 4 (Algebra on SMs)**.** *The algebra over the domain $\mathbb{S}$ is an algebraic structure denoted as $\mathcal{A} = (\mathbb{S}, \cup_{\mathbb{S}}, \setminus_{\mathbb{S}}, +_{\mathbb{S}})$ where $\cup_{\mathbb{S}} : \mathbb{S} \times \mathbb{S} \to \mathbb{S}$ is an 2-arity function called union, $\setminus_{\mathbb{S}} : \mathbb{S} \times \mathbb{S} \to \mathbb{S}$ is an 2-arity function called minus, and $+_{\mathbb{S}} : \mathbb{S} \times \mathbb{S} \to \mathbb{S}$ is a 2-arity function called join*[3].

The union function is defined as $\mathcal{S}_1 \cup_{\mathbb{S}} \mathcal{S}_2 = (Ent_1 \cup Ent_2, \sigma, I_\sigma)$ where I_σ is redefined such that: $(\forall r \in \sigma).I_\sigma(r) = I^1_\sigma(r) \cup I^2_\sigma(r)$ where I^i_σ is the interpretation of σ for structure S_i. The set of entities is the union of the entities of each structure, and the interpretation of the signature consists on the union of the types and edges of each SM. The union function can be interpreted as an operator that receives two structures and merge them to create a third one.

Example 1. *Fig. 2(a) presents two structures $\mathcal{S}_1$ and $\mathcal{S}_2$ and the union of both structures. $Ent_1 = \{0,1,2\}$, $Ent_2 = \{3,4\}$, and $Ent_{\mathcal{S}_1 \cup \mathcal{S}_2} = \{0,1,2,3,4\}$. The signature includes one unary relation BLACK where $BLACK = \{0\}$ for $\mathcal{S}_1$, $BLACK = \{3\}$ for $\mathcal{S}_2$ and $BLACK = \{0,3\}$ for $\mathcal{S}_1 \cup \mathcal{S}_2$*[4].



(a) Union (b) Minus (c) Join

Figure 2: Examples of the union, minus, and join operators.

The minus function is defined as $\mathcal{S}_1 \setminus_{\mathbb{S}} \mathcal{S}_2 = (Ent_1 \setminus Ent_2, \sigma, I_\sigma)$ i.e. the set of entities is the set of entities of S_1 except the entities that are also in S_2. The interpretation of σ is defined as follows:

- for each $\tau_i \in \sigma$, $I_\sigma(\tau_i) = I^1_\sigma(\tau_i) \setminus Ent_2$
- Let $connect : \mathcal{P}(\mathcal{E}) \times \mathcal{P}(\mathcal{E} \times \mathcal{E}) \to \mathcal{P}(\mathcal{E} \times \mathcal{E})$ be a function that, given a set of entities Ent and a set of edges ed, returns the subset of edges that contains any of the entities in Ent. The set of edges of the $\mathcal{S}_1 \setminus_{\mathbb{S}} \mathcal{S}_2$ is then $I_\sigma(ed) = I^1_\sigma(ed) \setminus connect(Ent_2, I^1_\sigma(ed))$

The minus function can be interpreted as an operator that takes a structure and removes a part from it.

Example 2. *Fig. 2(b) presents two structures $\mathcal{S}_1$ and $\mathcal{S}_2$ and $\mathcal{S}_1 \setminus \mathcal{S}_2$. $Ent_1 = \{0,1,2\}$, $Ent_2 = \{1\}$, and $Ent_{\mathcal{S}_1 \setminus \mathcal{S}_2} = \{0,2\}$. The signature includes the same unary relation BLACK where $BLACK = \{0\}$ for $\mathcal{S}_1$, $BLACK = \{\}$ for $\mathcal{S}_2$ and $BLACK = \{0\}$ for $\mathcal{S}_1 \setminus \mathcal{S}_2$.*

The join function is defined as $\mathcal{S}_1 +_{\mathbb{S}} \mathcal{S}_2 = (Ent_1 \cup Ent_2, \sigma, I_\sigma)$ i.e. the set of entities is the union of the domains of S_1 and S_2. The interpretation of σ is defined as follows:

- for each $\tau_i \in \sigma$, $I_\sigma(\tau_i) = I^1_\sigma(\tau_i) \cup I^2_\sigma(\tau_i)$

[2] 2^A denotes the power set of the set A, i.e. the set of all subsets of A including the empty set.

[3] We add $\mathbb{S}$ subscripted in each command to differentiate this commands from the usual set operators $\setminus$ and $\cup$ which we use extensively through the paper. For the sake of simplicity of notation we drop the subscript σ in $\mathbb{S}_\sigma$.

[4] We drop the subscript $\mathbb{S}$ on the operators for the sake of simplicity.

- Let $cross : \mathcal{P}(\mathcal{E}) \times \mathcal{P}(\mathcal{E}) \rightarrow \mathcal{P}(\mathcal{E} \times \mathcal{E})$ be a function that, given two sets of entities Ent_1 and Ent_2, it returns all of the edges from Ent_1 to Ent_2. Then, $I_\sigma(ed) = I^1_\sigma(ed) \cup I^2_\sigma(ed) \cup cross(Ent_1, Ent_2)$

The join operator is another operator for merging two structures. More than doing the union between two structures, the join operator also creates edges between all entities on the first structure to all entities on the second structure. This is useful when one wants to add a new entity to the system that should be connected to all the old entities.

Example 3. *Fig. 2(c) presents two structures $\mathcal{S}_1$ and $\mathcal{S}_2$ and $\mathcal{S}_1 + \mathcal{S}_2$. $Ent_1 = \{0,1,2\}$, $Ent_2 = \{1\}$, and $Ent_{\mathcal{S}_1+\mathcal{S}_2} = \{0,1,2,3,4\}$. The signature includes the same unary relation BLACK where $BLACK = \{0\}$ for $\mathcal{S}_1$, $BLACK = \{3\}$ for $\mathcal{S}_2$ and $BLACK = \{0,3\}$ for $\mathcal{S}_1 + \mathcal{S}_2$.*

We define an $\mathcal{A}-$expression as an expression that syntactically agrees with the signature of $\mathcal{A}$. For the sake of clarity we introduce a grammar for $\mathcal{A}-$expressions:

$$\mathcal{A}Exp ::= \mathcal{S} \mid \mathcal{A}Exp \cup_{\mathbb{S}} \mathcal{A}Exp \mid \mathcal{A}Exp \setminus_{\mathbb{S}} \mathcal{A}Exp \mid \mathcal{A}Exp +_{\mathbb{S}} \mathcal{A}Exp$$

where $\mathcal{S} \in \mathbb{S}$. We denote the set of all $\mathcal{A}-$expressions as $\mathcal{A}Exp$.

Each event in Σ of the transition system $\mathcal{T}$ maps to an $\mathcal{A}-$expression that takes the current structure and transforms it to the next structure.

4 MODELLING STATE OF ENTITIES

In this section we present two ways of embedding the state of the entities into the structure model: using unary relations and embedding the state-space directly into the SM.

4.1 Model state using unary relations

Let Q be the state-space of entities. For the sake of simplicity we assume that the entities have homogeneous bounded state-space. Let $\tau_q \subseteq Ent$ be a unary relation such that $e \in \tau_q$ iff the state of entity e is q where $q \in Q$. This solution allows modelling the state of entities into an SM without the need of extra notation.

Definition 5 (State consistency)**.** *Let $\mathcal{S} = (Ent, \sigma = \{\tau_{q_1}, \ldots, \tau_{q_n}, ed\}, I_\sigma)$ be an SM where $Q = \{q_1, \ldots, q_n\}$ denotes the state-space of entities. $\mathcal{S}$ is state-consistent iff $(\forall e \in Ent).(\exists! q \in Q).(e \in \tau_q)$*[5]

State consistency ensures that each entity $e \in Ent$ is at one and only one state $q \in Q$. If one wants to model non-determinism where a given entity can be at different states at a given time, then this should be introduced at the level of the SM (modelling different SMs) and not at the level of the state.

Example 4. *Consider an SM with the domain $Ent = \{v_1, v_2, \ldots, v_n, t_1, t_2 \ldots, t_m\}$ where v_i denotes a vehicle and t_j denotes a task. Vehicles can be assigned to tasks. Each vehicle is an entity of the system and is modelled as DES. All vehicles share the same DES definition where the state space is defined as $Q = \{IDLE, ASSIGNED, DONE, FAIL, TODO\}$. The state IDLE means that the vehicle is operative but not assigned to any task, the state ASSIGNED means that the vehicle is assigned to a task, the state DONE means that the vehicle finished a task, and the state FAIL means that the vehicle is inoperative. A task can be on state TODO which means that it was instantiated but not yet assigned, ASSIGNED, or DONE. The set of unary relations is then $T = \{\tau_{IDLE}, \tau_{ASSIGNED}, \tau_{DONE}, \tau_{FAIL}, \tau_{TODO}\}$ Fig. 3 presents an example of one vehicle that is assigned to two tasks.*

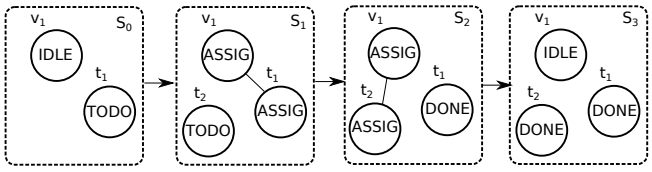


Figure 3: Example where both state of entities and structure changes over time.

To get the state of an entity requires, in the worst case, to check $|Q|$ predicates. This is might not be practical. Although, an SM must be seen as a logical MoC. It is not intended to perfectly match to the system's architecture. For that purpose we need a physical MoC, closer to the implementation. A physical MoC for systems with changing structure is under our future work scope and, thus, will not be addressed in this paper.

[5]The quantifier $\exists! a \in A.P(a)$ is called the uniqueness quantifier and means that there is one and only one a in A that satisfies the predicate P.

4.2 Embed the state-space into an SM

Another approach is to redefine the SM definition in order to include the state-space of the entities.

Let the function $state : Ent \to Q$ be defined as $state(e) = q \Leftrightarrow e \in \tau_q$ where $e \in Ent$ and τ_q's are defined as per the previous approach. We can now extend the definition of a SM from a relational structure to a many-sorted algebra where *state* is functional symbol. We call this algebraic model the Structure-State Model (SSM).

Definition 6 (Structure-State Model)**.** *A Structure-State Model is a many-sorted algebra denoted as* $\mathcal{S}_s = (Ent, Q, \{\sigma, state\}, \{I_\sigma, I_{state}\})$ *which results from the overloading of a structure model* $\mathcal{S} = (Ent, \sigma, I_\sigma)$*. Q is the state space of entities, state is a function symbol that is interpreted by* I_{state} *to a function of type* $Ent \to Q$ *which, given an entity, provides the state of such entity.*

For the remainder of the paper we are going to use the approach with the unary relations. The use of SSM remains future work.

5 COMPOSITION WITH AN UNDERLYING MODEL OF COMPUTATION

We aim at developing a model that can be used together with different MoCs that models the underlying dynamics of entities. In this section we compose the SM with Nancy Lynch's Synchronous Network Model. The composition consists of a map that extracts the SM from an SNM and a map that changes the SNM up to changes in the SM. The former can be seen informally as defining how the structure can be "observed" from an SNM point of view while the later defines how an SNM can be "controlled" from the structure level.

5.1 Synchronous Network Model

An SNM is defined as a set of processes *Proc* and a set of messages M. Each $p \in Proc$ is defined as a tuple $p = (states_p, start_p, msgs_p, trans_p, in_p, out_p)$ where $states_p$ is a (possibly infinite) set of states, $start_p \subseteq states$ is the set of initial states, $in_p \subseteq Proc$ denotes the incoming neighbours of p, $out_p \subseteq Proc$ denotes the outgoing neighbours of p, $msgs_p : states_p \times out_p \to M \cup \{null\}$ is a message-generation function, and $trans_p : states_p \times \{M \cup null\}^{|in_p|} \to states_p$ is a state-transition function.

For every round, each process p in the network sends a message to each of its outgoing neighbors (specified by *msgs* function) and receives a message from each of its incoming neighbours, computing its new state (specified by *trans* function). Note that a SNM forms a directed graph $G = (V, E)$ where $V = Proc$ and the set of edges E is formed according to the incoming and outgoing neighbors of each process.

In order to instantiate new SNM processes during execution it is convenient to have a set of "templates" abstracted from any incoming and outgoing neighbours. We define an SNM template has an SNM process where incoming neighbours and outgoing neighbours are abstracted by place-holders called ports. The sets in_{ports} and ou_{ports} are respectively the place-holders for incoming neighbours and outgoing neighbours. The functions *msgs* and *trans* are defined using the sets of ports instead of incoming and outgoing neighbours. We denote the set of process templates as P_{tmp}.

An n-process SNM can be instantiated using a partial function $inst_n : \mathcal{P}(P_{tmp})^n \times \mathcal{P}(Ports_{in} \times Ports_{out}) \hookrightarrow \mathcal{N}$ where $Ports_{in}$ and $Ports_{out}$ are respectively the set of all input ports and the set of all output ports for all the n process templates. $inst_n$ takes n process templates and a set of edges representing connections between input ports and output ports. After generating the processes, $inst_n$ binds the ports according to the set of edges. Note that all ports must be bonded i order to $inst_n$ to be defined. Otherwise it cannot produce a sound SNM.

5.2 Map from Synchronous Network Model to Structure Model

We define a map $\alpha : \mathcal{N} \times V_{states} \to \mathbb{S}$ that extracts the SM from an SNM. $\mathcal{N}$ is the set of all SNMs, and V_{states} is a set of vectors where each $[p_1 |_s, p_2 |_s, \ldots] \in V_{states}$ represents a snapshot of the state of all SNM processes[6].

The map computation of α consists of three steps: the generation of set Ent; the interpretation for relations τ_i; and the interpretation for relation ed. The first step generates a new entity $e \in Ent$ for each process $p \in Proc$. The second step interprets each τ_i as a predicate on the state of the processes. Thus, $e \in \tau_i$ iff the state of the corresponding process p satisfies a given predicate $f_{\tau_i}(p |_s)$. The interpretation of the relation symbol ed can be a direct map from the set of edges of the SNM or obtained as a predicate on the state of the SNM processes. The latter can be formalized as: $(e_1, e_2) \in ed$ if the state of the corresponding processes p_1 and p_2 satisfies a given predicate $f_{ed}(p_1 |_s, p_2 |_s)$.

[6]Given a tuple $x = (x_1, x_2, \ldots)$, we use the notation $x |_{x_1}$ to access to the value of x_1.

5.3 Map from Structure Model to Synchronous Network Model

In order to be able to control the SNM from the structure level we need to introduce the map: $\beta : \mathcal{S} \times \mathcal{S} \times \mathcal{N} \times V_{states} \to \mathcal{N}$ which takes the current structure, the new structure, and the old SNM and returns the new SNM. Note that adding and removing processes to and from an SNM might also require the redefinition of in_p, out_p, $msgs_p$ and $trans_p$ for some processes in *Proc*.

6 CASE STUDY

6.1 The Agree-and-Pursue Law (APL)

Our case study is an adaptation of the "agree-and-pursue control and communication law" (APL) used by Martínez et al. [MBCF07] to present an SNM-based model called Synchronous Robotic Networks. The example consists of a set of locally connected agents that can move on a circle. Each agent can travel clockwise or counter-clockwise. The APL can be informally defined as follows:

- Each agent transmits to its neighbours a message with its position, direction and the unique identifier (UID) of their "leader" denoted as "prior";
- Each agent compares its "prior" with the "prior" of all its incoming messages. If there exists a message with greater "prior" then the agent adopts it, assumes its direction, and increases its position proportionally to the difference between its current position and the position stated in the elected message. Otherwise the agent increases its position according to its previous pace.

We also assume that agents can be added and deleted during execution. This feature is not part of the original APL. We introduce it in order enrich the behavior of the structure.

6.2 APL modelled as an SNM

Consider an APL model with N_a agents. For the sake of simplicity we assume that agents are homogeneous in terms of communication range. The space is discretized into N_s samples of a unit circle. The set of agents is modelled as a set of SNM processes *Proc* where each for each $p \in Proc$, $s \in states_p$ is modelled as a tuple $s = (pos, dir, prior)$ where $pos \in \{0, 1, \ldots, N_s - 1\}$, $dir \in \{cw, ccw\}$ and, $prior \in \mathbb{N}_0$, and $start_p = (pos_0, dir_0, UID(p))$. pos_0 and dir_0 are arbitrary initial position and direction and the function *UID* provides a unique identifier for the process.

The sets in_p and out_p include all processes except p. The function *msgs* is defined as:

$$msgs_p(s, p') = \begin{cases} s, & \text{if } dist(s\,|_{pos}, s'\,|_{pos}) \leq \lambda \\ null, & \text{otherwise} \end{cases} \tag{1}$$

where s' is the state of p'. The function $trans_p$ implements the control algorithm informally stated above. Formally,

$$trans_p(s, \{m_1, m_2, \ldots, m_{N_a}\}) = \begin{cases} (s\,|_{pos} \pm \delta_p, s\,|_{dir}, s\,|_{prior}) & \text{if } s\,|_{prior} \geq M\,|_{prior} \\ (s\,|_{pos} \pm \eta dist(s\,|_{pos}, M\,|_{pos}), M\,|_{dir}, M\,|_{prior}) & \text{o.w.} \end{cases} \tag{2}$$

where $M \in \{m_1, m_2, \ldots, m_{N_a}\}$ is the message received with the greatest prior. $\eta \in \mathbb{N}$ is a fixed gain, and δ_p is the previous step for process p. We use the signal $\pm$ in the update of the position is replaced by $+$ if dir is *cw* and $-$ if dir is *ccw*. We denote the resulting SNM as A_{SNM}.

6.3 Structure Model for APL

Let $\mathcal{S} = (Proc, \{CW, ed\}, \{f_{CW}, f_{ed}\})$ denote the SM for APL where the set of entities are the set of SNM processes of A_{SNM}, CW is a unary relation that is true if the corresponding agent is rotating clockwise and false otherwise. CW is formally interpreted using a predicate $f_{CW}(p) = \top$ iff $p\,|_{dir} = cw$ and $f_{CW}(p) = \bot$ otherwise.

The binary relation *ed* models if two agents are within communication range. The interpretation f_{ed} is defined such that the following constraint is satisfied:

$$\forall p, q \in Proc.(p, q) \in ed \Rightarrow dist(p\,|_{pos}, q\,|_{pos}) \leq \lambda \tag{3}$$

An incoming edge and an outgoing edge between two entities means that the corresponding agents are within communication range. Note that this information cannot be extracted directly from the edges of A_{SNM}, since SNM doesn't allow changing the edges during execution. This behavior is entailed in the messages exchanged by processes: if a *null* message is sent between two agents then they are out of range. In the structure model the behavior of the communication topology is a first-class citizen and can be accessed syntactically by looking at the set of edges.

Note that we are not modelling all the state-space of the SNM processes at the structure level but only the state information that we find important for the sake of this example. This state-space abstraction is a common technique to prevent state-explosion of combinatorial algorithms. In the structure model, state-space abstraction and refinement can be done in a very natural way by adding and removing unary relations on the signature of the structure.

Fig. 4 depicts an example of three execution steps of an APL with seven agents with the corresponding SMs. The black disks denote agents rotating counter-clockwise while the unfilled disks represent agents rotating clockwise. The circumference surrounding each agent represents its communication range. At the bottom of Fig. 4 we present the corresponding STS. This STS is derived from the A_{SNM} using the map α at each step. The vectors v, v', and v'' are the state of the processes at each execution step. Note that APL has a bottom-up emerging dynamic structure: the communication topology changes due to the communication constraints, and the entities state changes due to their direction.

First we need to define the map β. We use the general definition provided in section 5. The only application-specific step is the definition of $inst_n$ for the instantiation of new processes. The instantiation needs to be sound up to the physical constraints of the system. Thus, the initial position of processes generated by $inst_n$ cannot violate the communication constraints stated in equation 3.

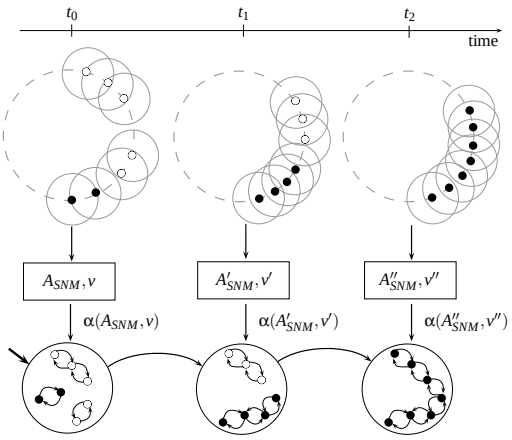


Figure 4: Example of 3 steps of an APL execution and the corresponding STS.

6.4 Structure-level Controller for APL

Consider that we want to have two "teams" of APL agents. Agents of each team only share information with their team-mates, and thus do not influence the behavior of agents outside the team. Let us design a structure-level controller that controls the network topology such that the system meets this new specification. The controller observes the current structure and if two agents from different teams share an edge (i.e. they are within communication range) then the controller disables communication between the agents by removing the edge.

Consider a new operator denoted as $\setminus_{ed} : \mathbb{S} \times (Ent \times Ent) \rightarrow \mathbb{S}$ such that $\mathcal{S}' = \mathcal{S} \setminus_{ed} (e_0, e_1)$ where $\mathcal{S}'$ has the same structure as $\mathcal{S}$ except $ed_{\mathcal{S}'} = ed_{\mathcal{S}} \setminus (e_0, e_1)$, i.e. the operator removes the edge from the original structure maintaining the remaining structure the same.

The following pseudo-code implements the team controller.

```
initialize A_SNM, states
define team_A, team_B
while true do {
  S = alpha(A_SNM, states)
  forall e in S.ed do {
    if (e.1 in team_A and e.2 in team_B) or (e.1 in team_B and e.2 in team_A)
    then S' = S \_ed e
    else S' = S
  }
  A_SNM = beta(S,S',A_SNM,states)
  states = exec(A_SNM,states)
}
```

The first step consists of initializing the agree and pursue law as a set of SNM processes, their corresponding initial states, and the definition of the two teams. Inside the infinite loop the first step consists of obtaining the current structure using the map α. The structure controller is represented by the "forall" loop which checks for edges shared by agents at different teams and removes them. The next step consists of propagating the new structure to the system using the map β. In this case β can disable communication channels even when the agents are within communication range. If

the corresponding edges get removed at the structure model β disables the capacity for both entities to communicate. A_{SNM} is then executed and the states of the processes are calculated.

This simple example shows the convenience of accessing the network topology as a first-class concept. Remember that the structure of an SNM is not explicit but rather implicit in the messages sent by each process. Thus, the structure model introduces a layer of abstraction that makes the structure explicit and easier to manipulate.

7 CONCLUSIONS AND FUTURE WORK

In this paper we introduce the Structure Model (SM) - an algebraic model of computation for modelling and controlling systems with dynamic structure. The SM is a mathematical structure defined by a set of entities, a set of unary relations and a binary relation. The unary relations are used to model the state of the entities while the binary relation models interactions between entities (e.g. communication network topology). An SM can be manipulated using a set of operators. The model for time evolution of SMs is formalized as a transition system. The structure model aims at being agnostic to the model of computation that describes the underlying system's behaviour. The composition of an underlying model with the SM allows extracting structural information from the system up to the structure level and propagate changes from the structure level back to the system. This allows the specification of structure-level controllers. We introduce a composition with the Synchronous Network Model and present a simple case study based on the "agree and pursue" communication and control law.

The structure model is at a early stage of development. We foreseen work to be done on the definition of the set of operators for manipulating structures. We are looking for the least amount of operators that are expressive enough to model useful controllers. This work will lead us to the design and implementation of a Domain-Specific Language (DSL) where structure-level controllers can be easily specified. We are also exploring the use of Bigraphs for extending the structure model with location of computation.

References

[Agh86] Gul Agha. *Actors: a model of concurrent computation in distributed systems.* MIT Press, Cambridge, MA, USA, 1986.

[BW04] Lowell W. Beineke and Robin J. Wilson. *Topics in Algebraic Graph Theory*. Cambridge University Press, New York, NY, USA, 2004.

[Cou91] Bruno Courcelle. Graphs as relational structures: An algebraic an logical approach. In *Proceedings of the 4th International Workshop on Graph-Grammars and Their Application to Computer Science*, pages 238–252, London, UK, 1991. Springer-Verlag.

[GSW06] D.Q. Goldin, S.A. Smolka, and P. Wegner. *Interactive computation: The new paradigm*. Springer-Verlag New York Inc, 2006.

[Hoa78] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21:666–677, 1978.

[KM77] Gilles Kahn and David B. Macqueen. Coroutines and networks of parallel processes. In *Information Processing 77*, pages 993–998. North Holland Publishing Company, 1977.

[Lyn96] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[MBCF07] S. Martinez, F. Bullo, J. Cortes, and E. Frazzoli. On synchronous robotic networks - Part I: Models, tasks and complexity. *IEEE Transactions on Automatic Control*, 52(12):2199–2213, 2007.

[Mil83] Robin Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25(3):267 – 310, 1983.

[Mil99] Robin Milner. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, 1st edition, June 1999.

[Mil06] R. Milner. Turing, computing, and communication. *Interactive Computation: The New Paradigm*, pages 1–8, 2006.

[Mil09] Robin Milner. *The Space and Motion of Communicating Agents*. Cambridge University Press, 2009.

[MPW89] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, part i and ii. *INFORMATION AND COMPUTATION*, 1989.

[RW87] Peter J. Ramadge and W. Murray Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, 25(1):206–230, January 1987.

[RW89] Peter J. Ramadge and W. Murray Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, January 1989.

[Sta05] Bozena Staruch. Extensions of partial structures and their application to modelling of multiagent systems. In *Monitoring, Security, and Rescue Techniques in Multiagent Systems*, volume 28 of *Advances in Intelligent and Soft Computing*, pages 293–304. Springer Berlin / Heidelberg, 2005. 10.1007/3-540-32370-8_22.