Formal Reasoning about Emergent Behaviours of Multi-Agent Systems

Hong Zhu

Department of Computing, School of Technology, Oxford Brookes University,
Wheatley Campus, Oxford OX33 1HX, UK, Email: hzhu@brookes.ac.uk

Abstract. *Emergent behaviour (EB) is a common phenomenon in multi-agent systems (MAS) where autonomous agents perform certain actions with only limited access to local information and make decisions individually, while the whole system demonstrates properties and behaviours that have strong global features. Because of the huge gap between individual agents' properties and behaviours and those of the whole system, specifying and reasoning about EBs are difficult. In this paper, we propose a framework to the specification of and reasoning about EBs based on our previous work on the SLABS language for the formal specification of MAS. We investigate the uses of the scenario specification in SLABS for the definition of EBs of MAS and study the properties of scenario inclusion, scenario transition and scenario update. The uses of these properties in the proofs of MAS' EBs are illustrated by an example.*

1. Motivation

In the development of software for service-oriented and Grid computing, it is essential but difficult to understand how the system as a whole will behave while its components behave autonomously. On the other hand the specification and design of each individual component must take into consideration of the state and behaviour of the whole system, which is the environment that individual components execute in and operate on. There is a wide gap between individual components' properties and behaviours and the whole system's properties and behaviours. This is known as the emergent behaviour (EB) problem; c.f. [1]. This paper proposes a formal system to facilitate the specification of and reasoning about systems that consist of multiple autonomous agents.

EB is a common phenomenon in multi-agent systems (MAS), such as in the Amalthaea system for web information retrieval and filtering [2], the ecosystem for resource allocation in a distributed environment [3], as well as in e-commerce (such as online auctions), simulation (such as ant colony optimisation), and many other application areas. Therefore, understanding EBs and facilitating the specification of and reasoning about EBs of MAS are very important to the development of MAS.

In the past few years, intensive research on EB of MAS has been done in artificial intelligence. Various mathematical models of specific types of MAS in particular application areas have been developed and studied, e.g. [3]. Formal logics of belief, desire and intension of intelligent agents were proposed, c.f. [4]. Formal specification of agent in Z notation was also investigated [5]. In our previous work, a formal specification language SLABS was proposed for engineering MAS [6, 7]. It has also be used in formal specification of evolutionary MAS [8]. However, we are still lack of a general formal logic system for specifying and reasoning about the EBs of MAS. This paper reports our work on such a logic system based on SLABS and the notion of scenarios.

The remainder of the paper is organised as follows. Section 2 briefly review the formal specification language SLABS. Section 3 studies the properties of scenarios and illustrates its uses in the specification of and reasoning about EBs. Section 4 concludes the paper with a discussion of further work.

2. Specification of MAS in SLABS

2.1. The specification language SLABS

SLABS stands for Specification Language for Agent-Based Systems [6,7]. A novel concept introduced by in SLABS is the notion of caste, which is a natural evolution of the OO concept of class. The agents in a MAS are classified by a partially ordered set of castes. Castes are the templates of agents which defines a set of structure, behaviour and environment features. An agent can join into or quit from a caste at run-time. Case studies have shown that caste can play a significant role in the development of MAS [9].

The specification of a MAS in SLABS consists of a set of specifications of castes in the following form.

Caste C <= C_1, …, C_k;
  ENVIRONMENT EC_1, …, EC_w;
    VAR $*v_1:T_1$, …, $*v_m:T_m$; $u_1:S_1$, …, $u_l:S_l$;
    ACTION $*A_1(p_{1,1}, \ldots, p_{1,n1})$, …, $*A_s(p_{s,1},\ldots, p_{s,ns})$;
      $B_1(q_{1,1},\ldots, q_{1,m1})$, …, $B_t(q_{t,1},\ldots, q_{t,mt})$;
    RULES R_1, R_2, …, R_h
End C;

The clause 'C <= C_1, ..., C_k' specifies that caste C inherits the structures, behaviours and environments of castes C_1, ..., C_k. We write $A \in_t C$ to denote that agent A belongs to caste C at time t.

The state space of an agent is described by a set of variables with keyword VAR. The set of actions is described by a set of identifiers with keyword ACTION. An action can have a number of parameters. An asterisk before the identifier indicates invisible variables and actions.

In SLABS, an agent's environment can be explicitly specified by clauses in the following forms to define a subset of the agents in the system that may affect its behaviour. (a) 'agent name' indicates a specific agent in the system; (b) 'All: caste-name' means all the agents of the caste; (c) "identifier: class-name" is a variable that any agent in the caste can be assigned to.

Agents' behaviours are defined by transition rules in the following form.

Behaviour-rule **::= [**<rule-name>**]** pattern**|[** prob**]**–>event,
[*If* Scenario**] [***where* pre-cond**]**;

where the pattern describes the pattern of the agent's previous behaviour. The scenario describes the situation in the environment. The event is the action to be taken when the scenario happens and the pre-condition is true, which is given in the where-clause. An agent may have a non-deterministic behaviour if multiple rules are applicable. The expression prob defines the probability for the agent to take the specified action on the scenario. It can be omitted so that the choices are non-deterministic.

A pattern describes the behaviour of an agent by a sequence of observable state changes and observable actions. It is written in the form of $[p_1, p_2, ..., p_n]$ where $n \geq 0$. Table 1 gives its formats and meanings.

Table 1. Meanings of the patterns

Pattern	Meaning
\$	The *wild card*, which matches with all actions
τ	*Silence*
X	*Action variable, which* matches an action
$Act\ (a_1, ...a_k)$	An action *Act* that takes place with parameters match $(a_1, ...a_k)$
$[p_1,..., p_n]$	The previous sequence of events match the patterns $p_1, ..., p_n$

A scenario is a combination of a set of agents' behaviours and states that describe a global situation in the operation of the system. Table 2 gives the format and semantics of scenario descriptions in SLABS.

Table 2. Semantics of scenario descriptions

Scenario	Meaning
Predicate	The state of the agents satisfies the predicate
$A=B$ (or $A \neq B$)	The identifiers A and B refer to the same (or different) agent
$A \in C$	Agent A is in the caste C
$A:P$	Agent A's behaviour matches pattern P
$\forall X \in C.Sc$	The scenario $Sc[X/A]$ is true for all agents A in caste C.
$\exists_{[m]} X \in C.Sc$	There are m agents in caste C such that $Sc[X/A]$ is true, where the default value of the optional expression m is 1.
S_1 & S_2	Both scenario S_1 and scenario S_2 are true
$S_1 \vee S_2$	Either scenario S_1 or S_2 or both are true
$\neg S$	Scenario S is not true

The following expressions can also occur in a scenario as a part of a predicate.

- *Set relation expressions* in the form of $\{X \in Caste \mid X : Pattern\}$, which is the set of agents whose behaviour matches the pattern;

Arithmetic relations may contain an expression in the form of (a) $A.V$, which refers to the variable V of agent A, (b) $\mu X \in C.Sc$, which is the number of agents A in caste C such that $Sc[X/A]$ is true, where Sc is a scenario.

The following is an examples of scenarios. It describes the situation that there are more agents in the caste Voter who vote Bush, vote(Bush), than those in the caste who vote for other candidates other than Bush.

μX(Voter.X:[vote(Bush)] >(X((Voter.X:[vote(Y)] & Y(Bush)

Agents behave in real-time concurrently and autonomously. A time index set T can be a subset of real numbers $[t_0, \infty)$, i.e. $T=\{t \mid t \in R \ \& \ t \geq t_0\}$.

A *run* r of a MAS is a mapping from time t to the set $\prod_{i=1}^{n} S_{A_i,t} \times \Sigma_{A_i,t}$, Where $S_{A,t}$ and $\Sigma_{A,t}$ are agent A's state space and the set of actions at time t. The behaviour of a MAS is defined by the set R of possible runs. For any given run r, the restriction of $r(t)$ on $S_{A,t} \times \Sigma_{A,t}$ written as $r_A(t)$, is a run of agent A in the context of r. We also write $R_A = \{r_A \mid r \in R\}$.

We assume that a MAS has the following properties. (a) Actions are instantaneous. (b) An agent may be silent τ (i.e. take no action) at a time moment t. (c) Any two actions taken by an agent must be separated by a none-zero period of silence. Consequently, an agent can take at most a countable number of non-silent actions in its lifetime.

The global state S_g of the system at any time moment t belongs to the set $\prod_{i=1}^{n} S_{A_i,t} \times \Sigma_{A_i,t}$. However, each agent A can only view the externally visible states and actions of the agents in its environment. In other words, an agent A can only view a part of S_g in the space $\prod_{X \in Env_{A,t}} S^V_{X,t} \times \Sigma^V_{X,t}$, where $Env_{A,t}$ denotes the set of agents that are in A's environment at time t, $S_{X,t}{}^V$ and $\Sigma_{X,t}{}^V$ denote the visible part of agent X's state and action at time t. The *history* of a run r up to time t, written as $r{\downarrow}t$, is a mapping that is the restriction of r to the subset $\{x \leq t \mid x \in T\}$ of T.

Let A be any given agent in a MAS. Let $c_1, ..., c_n, ... \in \Sigma_A - \{\tau\}$ be the sequence of non-silent actions taken by agent A in a run r and $t_1, t_2, ..., t_n, ... \in T$ are the times of the actions, i.e. $r_A^C(t_i) = c_i$ for all $i = 1, 2, ..., n, ...$ At a time moment $t \in T$, we say that c_n is agent A*'s* current action, and c_{n+1} the next action, if $t_n \leq t < t_{n+1}$. Let s_i be the state of agent A at time t_i, we write $Current(r_A{\downarrow}t) = <t_n, s_n, c_n>$, $Next(r_A{\downarrow}t) = <t_{n+1}, s_{n+1}, c_{n+1}>$, and $Events(r_A{\downarrow}t) = <<t_1, s_1, c_1>, ..., <t_n, s_n, c_n>>$.

Let Sc be a scenario. We write $A : r{\downarrow}t \models Sc$ to denote that from agent A's point of view, the scenario Sc occurs at time moment t in a run r. When taking a global view, we

omit the viewer and write $r \downarrow t \models Sc$. A formal definition of the notation can be found in [6].

The following define what meant by a correct implementation of a specification in SLABS. Let ***S*** be a formal specification in SLABS, ***M*** a MAS, and $RULE_A$ the set of behaviour rules specified in ***S*** for A.

Definition 1.

Agent A in ***M*** *always follows* a set R of rules, iff in all runs r of ***M***, $\forall t \in T.\exists <Sc \mid\!\rightarrow e; p> \in R.\ (A{:}r{\downarrow}t \models (Sc\ \&\ p) \Rightarrow Next(r_A{\downarrow}t)=e)$, and we write $A/M \models R$. A MAS ***M*** *always follows* ***S***, iff for all A in ***M***, A *always follows* $RULE_{A,t}$. Formally, $\forall A \in \boldsymbol{M}.(A/M \models RULE_{A,t})$, we write $\boldsymbol{M} \models_A \boldsymbol{S}$. □

Informally, at any time in a run of a MAS, an agent A that always follows a set R of behaviour rules will non-deterministically select one rule ρ from the applicable subset of R and then apply ρ, if the subset of applicable rules is non-empty.

It is worth noting that, the definition above gives the freedom to the agents to do anything that they like when there is no applicable rules. This enables the verification and validation of a MAS against a 'partial' specification of its behaviour. Such partial specifications are of particular importance in collaborative software development such as in service-oriented computing [10, 11]. The following definition still gives agents this freedom, but prevent them from taking actions unexpected.

Definition 2.

Let W be a subset of the actions that agent A can take. We say that A always *strictly follows* a set R of behaviour rules for actions in W, and write $A/M \models_{S(W)} R$, if A always follows R and $\forall t \in T.\forall r \in \boldsymbol{M},\ r_A{}^C(t)=e \in W \Rightarrow \exists t' \in T.\ \exists <Sc \mid\!\rightarrow e; p> \in R.(r{\downarrow}t' \models (Sc \& p)\ \&\ Next(r_A{\downarrow}t')=(t, e)$.

A MAS ***M*** *strictly follows* ***S***, written as $\boldsymbol{M} \models_S \boldsymbol{S}$, if and only if for all A in ***M***, A always strictly follows $RULE_A$ for all actions specified in ***S***. □

Definition 3.

Agent A in ***M*** *faithfully follows* a set R of rules, written as $A{:}r{\downarrow}t \models_F R$, iff for all runs r and time moments t, ($<Sc \mid\!\rightarrow e; p> \in R\ \&A{:}r{\downarrow}t \models (Sc\ \&\ p)$ implies that there is a run r' of ***M*** such that for all $u \leq t \in T.(r'(u)=r(u))$ and $Next(r'_A{\downarrow}t)=e$.

A MAS ***M*** *faithfully follows* ***S***, written as $\boldsymbol{M} \models_F \boldsymbol{S}$, if and only if for all A in ***M***, A faithfully follows $RULE_A$. □

Definition 4. (Correctness of implementation**)**

A MAS ***M*** is a correct implementation of specification ***S***, written as $\boldsymbol{M} \models \boldsymbol{S}$, iff ***M*** always follows ***S*** strictly and faithfully. Formally, $\boldsymbol{M} \models \boldsymbol{S} \Leftrightarrow \boldsymbol{M} \models_A \boldsymbol{S}\ \&\ \boldsymbol{M} \models_S \boldsymbol{S}\ \&\ \boldsymbol{M} \models_F \boldsymbol{S}$. □

2.2. Example: Autonomous sorting

In this subsection, we give an example of the formal specification of MAS. It is a simplified version of the sorting program in [12]. The original program can be found at URL http://diet-agents.sourceforge.net.

Example 1. (*SortingSpec*)

In this MAS, the agents can introduce one to another by passing through the identity of an agent that it knows.

```
CASTE Sociable;
  ENVIRONMENT ALL: Sociable;
  ACTION Introduce( Sociable /*to whom*/ , Sociable /*of whom*/ );
END.
```

Social agents are further divided into two sub-castes: *Linker* and *Mediator*. Each linker carries a value and can link to two other agents through channels *Higher* and *Lower*. Mediators only introduce agents to each other. An EB of the system may occur if each linker only connects through the *Higher* channel to an agent that carries a greater value and connects through the *Lower* channel to an agent that carries a less value. When all linkers are connected, the values carried by them are sorted. The mediator agents take random actions to introduce Linker agents to each other, which triggers the Linker agents to change their connections.

```
CASTE Linker <= Sociable;
  ENVIRONMENT Higher, Lower: Linker; All: Mediator;
  VAR *Value: INTEGER;
  BEGIN
    <I--Initialisation> <>  |→ Lower:= NIL; Higher:= NIL;
    <IH--Introduced to a better higher friend>
      [$]  |→ Higher:=Ag; IF ∃X∈ Sociable.X:[Introduce(Self, Ag)],
           WHERE Ag.Value > Self.Value & ((Higher = NIL) ∨
                (Higher ≠ NIL & Higher.Value > Ag.Value))
    <IL--Introduced to a better lower friend>
      [$]  |→ Lower:=Ag; IF ∃X∈ Sociable.X:[Introduce(Self, Ag)]
           WHERE Ag.Value < Self.Value & (Lower = NIL) ∨
                ((Lower ≠ NIL)& (Lower.Value < Ag.Value)))
END Linker;
CASTE Mediator <= Sociable;
BEGIN [$]  |→ Introduce(A, B); WHERE A ∈ Linker & B ∈ Linker
END Mediator.  □
```

An execution of a system that correctly implements ***SortingSpec*** will evolve into the state shown in Figure 1.

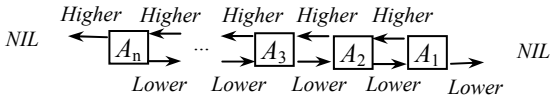

Figure 1. The emergent state of the MAS

This emergent state can be formally expressed in SLABS by the following scenario *Fully-Linked*.

Fully-Linked = $\exists_{[1]}$A∈Linker.(A.Higher=NIL)
& $\exists_{[1]}$A∈Linker.(A.Lower=NIL)
& ∀A∈Linker.(A.Higher≠NIL⇒∃B∈Linker.(A.Higher=B& B.Lower=A))
& ∀A∈Linker.(A.Lower≠NIL)⇒∃B∈Linker.(A.Lower=B & B.Higher=A)

The statement that in a run M of a multi-agent autonomous sorting system ***M*** that satisfies the above specification ***SortingSpec*** of autonomous sorting is in the scenario of *Fully-Linked* at time moment t can be formally expressed as $M{\downarrow}t \models$ *Fully-Linked*.

Let F_1={A∈Linker | A.Lower=NIL}, and
F_{n+1}={A∈Linker | ∃B∈F_n.(A.Lower=B & B.Higher=A)}.

When this statement is true, the system has that the following property.

$M{\downarrow}t \models$ *Fully-Linked* $\Rightarrow M{\downarrow}t \models \forall i \in \{1,\ldots,\|Linker\|\}.(\|F_i\|=1)$

Assume that the system is in the *Fully-Linked* scenario.

Let $A_i \in F_i$. The values carried by agents $A_1, A_2, \ldots, A_n$ are in the ascending order. This can be formally expressed by the following statement.

$$\forall i \in \{1,\ldots,||Linker||-1\}.(A_i.Value < A_{i+1}.Value).$$

Another very important property of the system is that in any run of a MAS that *strictly* follows ***SortingSpec***, it will eventually come to the scenario of *Fully-Linked*. In general, let $\boldsymbol{M}$ be any given MAS and Sc be a given scenario. We write $\boldsymbol{M} \rightarrowtail Sc$ to denote that $\forall M \in \boldsymbol{M}.\exists t \in T.\ (M{\downarrow}t \models Sc)$. The EB of autonomous sorting can then be formally expressed by the statement (S1) below.

$$\boldsymbol{M} \models \textbf{\textit{SortingSpec}} \Rightarrow \boldsymbol{M} \rightarrowtail \textit{Fully-Linked} \qquad \text{(S1)}$$

Note that, without the strictness condition, a linker may update its *Higher*/*Lower* channels without following the behaviour rules. Hence, it may connect to any agents regardless the value carried by them.

If the MAS strictly follows ***SortingSpec***, it will not only eventually come to the state of *Fully-Linked*, but also stay in the state although mediators are still making introductions to the linker agents. This is the *convergence* property of the MAS. In general, we write $M{\downarrow}@ \models Sc$ to denote that $\exists t \in T.(\forall t' \in T.(t' \geq t \Rightarrow M{\downarrow}t' \models Sc)$, and $\boldsymbol{M} \looparrowright Sc$ to denote that $\forall M \in \boldsymbol{M}.(\ M{\downarrow}@ \models Sc)$. Therefore, we have that

$$\boldsymbol{M} \models \textbf{\textit{SortingSpec}} \Rightarrow \boldsymbol{M} \looparrowright \textit{Fully-Linked}. \qquad \text{(S2)}$$

In the next section, we will discuss how to prove the above statements.

3. Reasoning about emergent behaviours

To enable the formal reasoning about MAS' behaviours, we define two relations on scenarios and an operator on scenarios and study their properties. For the sake of space, we will omit the proofs in the paper.

3.1. Scenario inclusion relation

One of the basic relation between scenarios is the inclusion relation $\Rightarrow$. Informally, $Sc_1 \Rightarrow Sc_2$ means that, if the system is in scenario Sc_1, it is also in scenario Sc_2.

Definition 5. (Scenario inclusion**)**

A scenario Sc_1 implies scenario Sc_2 in a MAS $\boldsymbol{M}$, written $\boldsymbol{M} \models Sc_1 \Rightarrow Sc_2$, if and only if for all runs M and at all time moments $t \in T$, $M{\downarrow}t \models Sc_1$ implies that $M{\downarrow}t \models Sc_2$. □

The inclusion relation has the following properties.

Lemma 1. For all agents A, and patterns $[P_1, P_2, \ldots P_n]$, we have that for all $\boldsymbol{M}$**,**

(1) $\boldsymbol{M} \models A{:}[P_1, P_2, \ldots P_n] \Rightarrow A{:}[P_2, \ldots P_n]$;

(2) $\boldsymbol{M} \models A{:}[P_1, \ldots, P_n] \Rightarrow A{:}[P'_1, \ldots, P'_n]$, if for all $i = 1,2,\ldots,n$, $P_i = P'_i$ or $P_i = \$$. □

Lemma 2.

(1) For all predicates $pred_1$ and $pred_2$ defined on the state space of a MAS $\boldsymbol{M}$, $\boldsymbol{M} \models pred_1 \Rightarrow pred_2$, if $pred_1 \Rightarrow pred_2$ is true in first order predicate logic.

(2) For scenarios S_1 and S_2, we have that for all $\boldsymbol{M}$, $\boldsymbol{M} \models S_1 \Rightarrow S_2$, if $\hat{S}_1 \Rightarrow \hat{S}_2$, where $\hat{S}_i$ is obtained from S_i, $i = 1, 2$, by replacing patterns with propositions. □

By the above lemmas, we can see that $\Rightarrow$ is just the logic connective implication if patterns are considered as propositions.

Example 2.

Consider the MAS specified in section 2.2. Suppose that a Linker agent A is connected to agent B through the *Higher* channel and A is introduced to agent C, which carries a value greater than the value carried by A but less than the value carried by B. This situation can be formally expressed as follows.

Better-Higher-Introduced
$= (A.Higher = B) \ \&\ \exists X \in Sociable.(X{:}[Introduce(A,C)]$
$\&\ (C.Value < B.Value)\ \&\ (C.Value > A.Value))$

Intuitively, according to Linker's specification, the IH behaviour rule should be enabled. By Lemma 2, we can formally prove that

(1) *Better-Higher-Introduced* implies the scenario of the IH rule, i.e. *Better-Higher-Introduced* $\Rightarrow \exists X \in Sociable.(X{:}[Introduce(A,C)])$

(2) *Better-Higher-Introduced* implies that the precondition of the IH rule is true, i.e. *Better-Higher-Introduced* $\Rightarrow C.Value > A.Value$ & $((A.Higher = NIL) \vee (A.Higher \neq NIL \ \&\ A.Higher.Value > C.Value))$ □

3.2. Scenario update

In Example 2, we have shown that agent A in scenario *Better-Higher-Introduced* can apply the behaviour rule IH. When the rule is applied, agent A's *Higher* channel will be updated so that it connects to C. Consequently, the system will be in a different scenario. We will write $Sc\hat{}(A{:}E)$ to denote the scenario after agent A taking an action E in the scenario Sc. The following definition formally defines this operator.

Definition 6. (Scenario update**)**

Let Sc be any given scenario, A an agent, and E an action that can be taken by agent A. We define $Sc\hat{}(A{:}E)$ to be the scenario that for all run r of the system $r{\downarrow}t_{n+1} \models Sc\hat{}(A{:}E)$ if and only if $r{\downarrow}t \models Sc$ and $Next(r_A{\downarrow}t) = \langle t_{n+1}, s_{n+1}, c_{n+1}\rangle$ and $E = \langle s_{n+1}, c_{n+1}\rangle$. □

The operator ^ has the following properties.

Lemma 3. For all scenarios S, S_1, S_2, agents A and B, and actions E, we have the following properties of the ^ operator.

(1) $\boldsymbol{M} \models (S_1 \& S_2)\hat{}(A{:}E) \Leftrightarrow S_1\hat{}(A{:}E)\ \&\ S_2\hat{}(A{:}E)$

(2) $\boldsymbol{M} \models (S_1\ or\ S_2)\hat{}(A{:}E) \Leftrightarrow S_1\hat{}(A{:}E)\ or\ S_2\hat{}(A{:}E)$

(3) $\boldsymbol{M} \models A{:}[P_1, P_2, \ldots P_n]\hat{}(A{:}E) \Leftrightarrow A{:}[P_1, P_2, \ldots P_n, E]$

(4) $\boldsymbol{M} \models B{:}[P_1, \ldots P_n]\hat{}(A{:}E) \Leftrightarrow (B{:}[P_1, \ldots P_n]) \& (A{:}[E])$, if $A \neq B$.

(5) $\boldsymbol{M} \models (\forall x \in C.S)\hat{}(A{:}E) \Leftrightarrow (\forall x \in C.S) \& (A{:}[E])$, if $A \notin C$.

(6) $\boldsymbol{M} \models (\forall x \in C.S)\hat{}(A{:}E)$
$\Leftrightarrow (\forall x \in C.(x \neq A \Rightarrow S)) \& (S[x/A]\hat{}A{:}E)$, if $A \in C$.

(7) $\boldsymbol{M} \models (\exists x \in C.S)\hat{}(A{:}E) \Leftrightarrow (\exists x \in C.S) \& (A{:}[E])$, if $A \notin C$.

(8) $\boldsymbol{M} \models (\exists x \in C.S)\hat{}(A{:}E)$
$\Leftrightarrow (\exists x \in C.(\ x \neq A\ \&\ S)$ or $(S[x/A]\hat{}(A{:}\ E)$, if $A \in C$,

where $S[x/A]$ is obtained by replacing free occurrences of the variable x systematically with A. □

Lemma 4. Let *Pred* be any predicate on the state of a MAS $\boldsymbol{M}$, A any agent in $\boldsymbol{M}$, and E an action that A can take. We have that $\boldsymbol{M} \models Pred\hat{}(A{:}E) \Leftrightarrow s.p.c(Pred)$, where $s.p.c(Pred, A, E)$ is the *strongest post-condition* of *Pred* w.r.t. to A's action E. □

Example 3. For example, consider the autonomous sorting system.

(1) The strongest post condition of the predicate ($C.Value > B.Value$) with respect to agent A's action $Higher{:=}C$ is that ($C.Value > B.Value$) & ($A.Higher{=}C$).

(2) The strongest post condition of the predicate ($A.Higher{=}B$) w.r.t. to the action $Higher{:=}C$ is ($A.Higher{=}C$). □

Example 4.

Continuing Example 2, we can see that after agent A's application of the IH rule, the system will be in the scenario that A is in the state *Better-Higher-Introduced*^ ($A{:}Higher{:=}C$). By the properties of ^ operator and ⇒, we can derive that *Better-Higher-Introduced*^($A{:}Higher{:=}C$) ⇒ ($A.Higher{=}C$). □

3.3. Scenario transition

Having proved that after agent A applies the behaviour rule IH in the scenario *Better-Higher-Introduced*, the system will be in the scenario $A.Higher{=}C$, we would like to formally express that there is a relationship between these two scenarios. The following definition defines a relation $\rightarrow$ on scenarios such that $S_1 \rightarrow S_2$ means the system in a state in scenario S_1 can evolve into a state in scenario S_2.

Definition 7. (Scenario transition**)**

Let S_1 and S_2 be two scenarios of a MAS $\boldsymbol{M}$. We say that S_1 *can lead to* S_2 in the system $\boldsymbol{M}$ and write $\boldsymbol{M} \models S_1 \rightarrow S_2$ if and only if there is a run M of the system $\boldsymbol{M}$ and time moments $t_1 < t_2 \in T$, we have that $M{\downarrow}t_1 \models S_1$ and $M{\downarrow}t_2 \models S_2$. □

The relation $\rightarrow$ has the following properties.

Lemma 5. Let S, S_1, S_2, S_3 be scenarios of a MAS.

(1) $\boldsymbol{M} \models S_1 \rightarrow S_2$ and $\boldsymbol{M} \models S_2 \rightarrow S_3$ imply that $\boldsymbol{M} \models S_1 \rightarrow S_3$;

(2) $\boldsymbol{M} \models S_1 \Rightarrow S_2$ and $\boldsymbol{M} \models S_2 \rightarrow S_3$ imply that $\boldsymbol{M} \models S_1 \rightarrow S_3$;

(3) $\boldsymbol{M} \models S_1 \rightarrow S_2$ and $\boldsymbol{M} \models S_2 \Rightarrow S_3$ imply that $\boldsymbol{M} \models S_1 \rightarrow S_3$. □

Lemma 6.

If an agent A in a MAS follows behaviour rule $<S| \rightarrow E; P>$, for all assignments α, $\alpha(S\&P) \rightarrow \alpha((S\&P)\hat{}(A{:}E))$. □

Example 5.

Let α be an assignment such that $\alpha(Ag){=}C$ and $\alpha(Self){=}A$. Let

$IH\text{-}Premise = \alpha((\exists X \in Sociable.(X : [Introduce(Self, Ag)]))$
$(Ag.Value > Self.Value \ \&\ ((Higher = NIL)$
$\vee (Higher \neq NIL \ \&\ Higher.Value > Ag.Value))))$

By Example 2, *Better-Higher-Introduced* ⇒ *IH-Premise*.

Let $IH\text{-}Result = \alpha((\exists X \in Sociable.(X : [Introduce(Self, Ag)]))$
$\&\ (Ag.Value > Self.Value \ \&\ ((Higher{=}NIL)$
$\vee (Higher{\neq}NIL \& Higher.Value > Ag.Value)))\hat{}\ (A{:}\ Higher{:=}Ag))$

By Example 4, *IH-Result* ⇒ $A.Higher{=}C$. By Lemma 6, *IH-Premise* → *IH-Result* → ($A.Higher{=}C$). □

By the properties of the scenario transitions, we can prove the reachability of a scenario in a MAS.

Example 6. (Reachability**)**

Consider the autonomous sorting MAS. We now prove that it can lead to the scenario of *Fully-Linked*. The following is an outline of the proof.

(1) Assume that there are $N>0$ Linker agents, $K>0$ Mediator agents, and Linker agents $A_i.Value < A_{i+1}.Value$, $i=1, \ldots, N-1$. Initially the system is in the scenario that no agents are linked to each other. That is,
$Initial\text{-}State = \forall X \in Linker.(X.Lower{=}NIL \& X.Higher{=}NIL)$

(2) From the initial state, assume that mediator agents introduce agents A_i and A_{i+1} to each other in the order that $i=1,2, \ldots, N-1$. Although this is probably not to happen when the mediators selects the introduction action at random, but this is still possible.

(3) Then, we can prove that $Linked_k \rightarrow Linked_{k+1}$ for all $k=0,1, \ldots, N$, where $Linked_k$ is depicted in Figure 2.

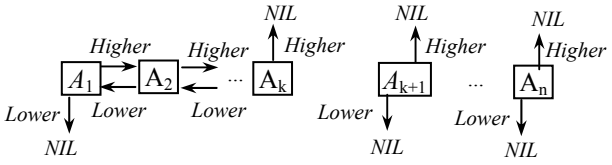

Figure 2. The scenario $Linked_k$

(4) Finally, notice that $Linked_N = Fully\text{-}Linked$. By Lemma 5(2), we have *Initial-State* → *Fully-Linked*.

Therefore, we can prove that autonomous sorting algorithm can reach the *Fully-Linked* state. □

3.4. Example: Autonomous sorting

Now, let's prove that a correct implementation will always evolve into the *Fully-Linked* state. Moreover, once reached this state, it will stay in the state.

For each Linker agent A_i, $i=1,2, \ldots, n$, in an autonomous sorting system $\boldsymbol{M}$, we define scenario $A_i\text{-}H\text{-}LinksTo_j$ and $A_i\text{-}H\text{-}NotLinked$ as follows.

$A_i\text{-}H\text{-}LinkedTo_j \Leftrightarrow (A_i.Higher{=}A_j)$,
$A_i\text{-}H\text{-}NotLinked \Leftrightarrow A_i.Higher{=}NIL$.

If $\boldsymbol{M}{\models}\boldsymbol{SortingSpec}$, we can prove that in any run r of the system $\boldsymbol{M}$, at all time moment t, $r{\downarrow}t{\models}A_i\text{-}H\text{-}LinkedTo_j$ implies that $i<j$. Otherwise, assume that at time moment t, we have that $i \geq j$. Then, there must be a time moment t' such that agent A_i assigned the value A_j to its *Higher* variable. Since $\boldsymbol{M}$ strictly follows the behaviour rules for Linker agents, A_i's assignment of A_j to its *Higher* variable must have been followed the behaviour rule IH. Therefore, we have that $A_i.Value < A_j.Value$. Thus, $i<j$. This is contradiction to the assumption. Consequently, agent A_i must be in one of the scenarios $A_i\text{-}H\text{-}NotLinked$ and $A_i\text{-}H\text{-}LinkedTo_j$ where $j = i+1, i+2, \ldots, n$. The set of scenarios is called *complete*. It is also *orthogonal* in the sense that at any time at most one of them can be true.

Similar to Example 6, we can prove that for all i, j and k, $i < j < k$, and $i, j, k = 1,2, \ldots, n$, $A_i\text{-}H\text{-}LinksTo_k \rightarrow A_i\text{-}H\text{-}LinksTo_j$ and $A_i\text{-}H\text{-}NotLinked \rightarrow A_i\text{-}H\text{-}LinksTo_j$. That is, we have the following state transition diagram for

agent A_i, where node labeled with k represents scenario A_i-*H-LinkedTo*$_k$ and node labeled with ∞ represents the scenario A_i-*H-NotLinked*.

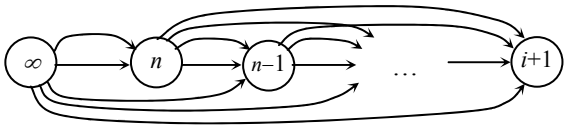

Figure 3. State transition diagram for Linker agent A_i

From this diagram, it is easy to see that Linker agent A_i will eventually evolve to the state in scenario A_i-*H-LinkedTo*$_{(i+1)}$.

Similarly, we can prove that for each Linker agent A_i, the following set of scenarios is complete and orthogonal.

A_i-*L-LinkedTo*$_j \Leftrightarrow (A_i.Lower = A_j)$, j=1, 2, …, i−1.

A_i-*L-NotLinked* $\Leftrightarrow A_i.Lower = NIL$.

Moreover, we have the following transitions between the scenarios. For all $i>j>k$ =1,2,…, n,

A_i-*L-LinksTo*$_k \rightarrow A_i$-*L-LinksTo*$_j$ and

A_i-*L-NotLinked* $\rightarrow A_i$-*H-LinksTo*$_j$.

Therefore, the Linker agent A_i will also eventually evolve into the state in scenario A_i-*L-LinkedTo*$_{(i-1)}$.

Notice that, *Fully-Linked* $\Leftrightarrow$ $\forall i \in (1,\ldots,n-1)$. A_i-*H-LinkedTo*$_{(i+1)}$ & $\forall i \in (2,\ldots,n)$. A_i-*L-LinkedTo*$_{(i-1)}$ & A_1-*L-NotLinked* & A_n-*H-NotLinked*.

Therefore, we proved that a correct implementation ***M*** of the specification ***SortingSpec*** will eventually evolve into the *Fully-Linked* state, i.e. statement (S1) is true.

Because both sets *L-LinksTo* and *H-LinksTo* of scenarios are complete and there is no state transition from the state of A_i-*H-LinkedTo*$_{(i+1)}$ or A_i-*L-LinkedTo*$_{(i-1)}$, the system will stay in the *Fully-Linked* state when it achieves it. Thus, statement (S2) is true.

It is worthy noting that, the following two scenarios have the same properties of the *Fully-Linked* scenario.

Fully-H-Linked =

$\forall i \in (1,\ldots,n-1)$. A_i-*H-LinkedTo*$_{(i+1)}$ & A_n-*H-NotLinked*,

Fully-L-Linked =

$\forall i \in (2,\ldots,n)$. A_i-*L-LinkedTo*$_{(i-1)}$ & A_1-*L-NotLinked*.

We can prove the following statements.

$\boldsymbol{M} \models \textbf{\textit{SortingSpec}} \Rightarrow \boldsymbol{M} \rightarrowtail$ *Fully-H-Linked*, (S1.H)

$\boldsymbol{M} \models \textbf{\textit{SortingSpec}} \Rightarrow \boldsymbol{M} \not\rightarrowtail$ *Fully-H-Linked*, (S2.H)

$\forall t \in T.\ (M\downarrow t \models$ *Fully-H-Linked*

$\Rightarrow \forall i \in \{1,\ldots,\|Linker\|-1\}.(A_i.Value < A_{i+1}.Value))$.

The similar statements hold for *Fully-L-Linked* scenario. Therefore, they can also be considered as the emergent states of autonomous sorting. The following formally state relationships between these three emergent states.

Fully-Linked $\Rightarrow$ *Fully-H-Linked*,

Fully-Linked $\Rightarrow$ *Fully-L-Linked*.

An interesting property of the emergent states *Fully-H-Linked* and *Fully-L-Linked* is that the system still dynamically change it state and perform actions when it is in such a scenario. This is the dynamic feature of EB.

4. Conclusion

In this paper, we presented a framework to specify and prove the EBs of MAS. The method is illustrated by an example of autonomous sorting. The concept of scenario plays the central role in the method. Based on the formal semantics of the specification language SLABS, we investigated the properties of the scenario inclusion relation $\Rightarrow$, the scenario transition relation $\rightarrow$ and the update operator ^ on scenarios. We are further studying the properties of scenarios, such as complete and orthogonal systems of scenarios. Our preliminary study shows that the concept of scenarios is expressive and suitable for the study of EBs. We are also investigating how the concepts proposed in this paper are related to existing formalisms in software specification and proof, such as Hoare logic, process algebra, temporal logic, and modal logics, etc.

Acknowledgement

The paper is written when the author is visiting the Future Technologies Group of the BT's Pervasive ICT Centre at Ipswich, UK. The author is most grateful to the research group, especially Fang Wang, Cefn Hoile, *et al*. for the friendly and stimulating research environment and numerous invaluable discussions on related topics.

References

[1] Johnson, S., Emergence: The Connected Lives of Ants, Brains, Cities, and Software, Scribner, September, 2002.

[2] Moukas, A., Amalthaea: Information Discovery and Filtering Using a Multi-Agent Evolving Ecosystem, Journal of Applied Artificial Intelligence 11(5) , 1997, 437–457.

[3] Walsh, W. E., et al., Some Economics of Market-Based Distributed Scheduling**,** in: Proc. of 18th International Conference on Distributed Computing Systems, May 1998, 612-619.

[4] Wooldridge, M., Reasoning About Rational Agents, MIT Press, 2000.

[5] D'Inverno, M. and Luck, M., Understanding Agent Systems, 2nd Edition, Springer, 2004.

[6] Zhu, H. SLABS: A Formal Specification Language for Agent-Based Systems, Int. J. of Software Engineering and Knowledge Engineering 11(5) (Nov. 2001), 529-558.

[7] Zhu, H., A Formal Specification Language for Agent-Oriented Software Engineering, in Proc. of AAMAS'2003, July, 2003, Melbourne, Australia, 1174-1175.

[8] Zhu, H., Formal Specification of Evolutionary Software Agents, Proc. of ICFEM'2002, Springer LNCS 2495, 2002, 249-261.

[9] Zhu, H., The role of caste in formal specification of MAS, in Proc. of PRIMA'2001, LNCS 2132, Springer, 1-15.

[10] Zhu, H., Zhou, B., Xinjun Mao, X., Shan, L. Duce, D., Agent-Oriented Formal Specification of Web Services, Proc. of AAC-GEVO'2004, Wuhan, China, Oct. 2004.

[11] Zhu, H. and Shan, L., Agent-Oriented Modelling and Specification of Web Services, Post-Conference Proc. of WORDS2005, Sedona, Arizona, USA, Feb., 2005.

[12] Marrow, P., et al., Agents in decentralised information ecosystems: the DIET approach. Proc. of AISB'01 Symposium on Information Agents for Electronic Commerce, York, UK, 2001, 109-117.