Verifying bigraphical models of architectural reconfigurations

Alejandro Sánchez*†, Luìs Soares Barbosa† and Daniel Riesco*
*Departamento de Informática, Universidad Nacional de San Luis, San Luis, Argentina
Email: {asanchez,driesco}@unsl.edu.ar
†HASLab - INESC TEC & Universidade do Minho, Braga, Portugal.
Email:{asanchez,lsb}@di.uminho.pt

***Abstract*—ARCHERY is an architectural description language for modelling and reasoning about distributed, heterogeneous and dynamically reconfigurable systems. This paper proposes a structural semantics for ARCHERY, and a method for deriving *labelled transition systems* (LTS) in which states and transitions represent configurations and reconfiguration operations, respectively. Architectures are modelled by bigraphs and their dynamics by parametric reaction rules. The resulting LTSs can be regarded as Kripke frames, appropriate for verifying reconfiguration constraints over architectural patterns expressed in a modal logic. The derivation method proposed here applies the approach in [1] twice, and combines the results of each application to obtain a label representing a reconfiguration operation and its actual parameters. Labels obtained in this way are minimal and yield LTSs in which bisimulation is a congruence.**

I. INTRODUCTION

Complex software systems are built by plugging together heterogeneous computational entities in very large, often loosely coupled, highly dynamic configurations. Change being the norm, rather than the exception, dynamic reconfiguration emerged as a main theme in architectural design.

Such is the motivation for the ARCHERY language [2], [3]. Its basic specification concept is that of an *architectural pattern*, which comprises a set of architectural elements (connectors and components) specified by their behaviours and interfaces (set of ports). An architecture describes a particular configuration of instances of pattern elements through a set of attachments linking their ports, and a set of renamings changing the externally visible names of ports. Both patterns and elements act as types for behaviours which are kept and referenced through typed variables. The language supports hierarchical composition and allows attachments between instances located at arbitrary nesting levels.

In this work we describe a few language extensions intended to specify reconfiguration scripts and constraints. Scripting operations are devoted to the creation and removal of instances, attachments, renamings and variables, as well as for moving instances and variables. Constraints, on the other hand, can be associated to a pattern or to an element, serving a number of purposes. For example, constraints can be used to ensure that the relevant design principles in a pattern are preserved by the application of reconfiguration scripts. For the purpose of this work, we limit constraints to be expressed in a Hennessy-Milner logic [4]. The logic is interpreted over a Kripke structure where the frame is a *labelled transition system* (LTS) in which states represent configurations, and transitions, represent reconfiguration operations.

ARCHERY was originally introduced in [2] and its extensions in [3]. The former reference is focused on the behavioural dimension: the language semantics being given by a translation to a process algebra. The latter deals with structural concerns providing an encoding into *bigraphical reaction systems* (BRSs) [5]. This work provides structural semantics by translation to a BRS as well, but allowing for attachments among variables located at arbitrary nesting levels in an architecture.

The theory of BRSs has two main objectives. One is to study systems in which locality and placing varies independently, a characteristic which allows for an elegant representation of ARCHERY's static and dynamic specifications. The other is to provide a general unifying theory, precisely defined within a framework of monoidal categories, that allows for the study of different calculi for concurrency and mobility. Moreover, recent results [6] provide further evidence of the expressiveness of BRSs as a meta-language for domain-specific modelling languages.

Reference [1] introduces an approach to derive LTSs in which behavioural equivalence is a congruence. In the sequel, we refer to such work as Leifer's approach. An instance of a BRS comprises a bigraph, which relates a single set of nodes in two orthogonal ways, represented by a *place-graph* and a *link-graph*, and a set of parametric reaction rules. A bigraph corresponds to an arrow in a suitable category. Thus, the composition of bigraphs C and a is defined if the domain of C matches the image of a. Bigraph a is typically called an *agent* if its domain is a distinguished object that does not admit composition. In contrast, a bigraph C, that can be composed in its domain, is called a *context*. Given a reaction rule and an agent, Leifer's approach allows for the derivation of a label that corresponds to the minimal context which enables the reaction to occur.

However, labels derived in this way do not convey sufficient information for carrying out analysis and verification of architectural reconfigurations. For this labels must represent operations and their actual parameters. Additionally, behavioural congruence must be preserved as well. This paper describes a method for deriving such labels and enables, in this way, the combination of the expressive power of BRSs with modal logic and its tools, in order to describe the software architecture domain.

The rest of the paper is organised as follows: section II describes ARCHERY and its extensions; section III introduces its structural semantics; section IV presents the method to derive appropriate labels; and section V draws conclusions and mentions, future, and on going work. Being essentially a short technical paper, the interested reader is referred to [2], [3] for further details and applications of the language.

II. (RE)CONFIGURATION OF ARCHITECTURAL PATTERNS

ARCHERY is structured as a core language and two extensions, referred to as ARCHERY-CORE, ARCHERY-SCRIPT, and ARCHERY-CONSTRAINT, respectively. We describe and illustrate the language using a refinement of the *Pipes & Filters* architectural pattern. The pattern prescribes architectures built up of two elements, a connector and a component type, respectively named *Pipe* and *Filter*, that can be arranged to transform a stream of data. The main underlying design principle is that filters can only be connected through pipes.

An ARCHERY-CORE model comprises one or more patterns and a main architecture referenced by a variable. Listing 1 shows the specification of the *Pipes & Filters* pattern and the declaration of an instance.

```
1  pattern PipeFilter()
2  element Pipe()
3    act acc, fwd;
4    proc Pipe() = acc.fwd.Pipe();
5    interface in acc; out fwd;
6  element Filter()
7    act rec, trf, snd;
8    proc Filter() = rec.trf.snd.Filter();
9    interface in rec; out snd;
10 end
11 pf:PipeFilter=architecture PipeFilter()
12 instances
13   f1:Filter=Filter(); f2:Filter=Filter();
14   p1:Pipe = Pipe();
15 attachments
16   from f1.snd to p1.acc; from p1.fwd to f2.rec;
17 interface
18     f1.rec as r1; f2.send as s1;
19 end
```

Listing 1. ARCHERY example pattern and architecture.

A pattern has a unique identifier, and includes, an optional list of formal parameters, and one or more architectural elements. Each architectural element in a pattern is described by an identifier, an optional list of formal parameters, a description of its *behaviour*, and an *interface*. Its behaviour consists of a list of actions and a list of process expressions. The head of the latter constitutes the element's initial behaviour which must have all parameters bound to an actual value. Process expressions are specified in a slightly modified subset of mCRL2 [7], yielding sequential processes. The behaviour of instances of element `Filter`, is shown in lines 7 and 8, and consists of a loop: receive data (`rec`), transform it (`trf`), and submit the result (`snd`). The interface, on the other hand, contains one or more ports. Each *port* is defined by a polarity, either `in` or `out`, and a token that must match an action name in the list of action definitions. For instance, the interface of `Filter` defines two ports in line 9. ARCHERY adopts a water flow metaphor for ports: an `in` port receives input from *any* port connected to it, and an `out` port sends output to *all* ports connected to it. Ports are synchronous: if needed, a suitable process algebra expression can be used to emulate any other kind of port behaviour.

An architecture defines a set of variables and describes the configuration adopted by their instances. It contains a token that must match a pattern name, an optional list of actual arguments, a set of variables, an optional set of attachments, and an optional interface. The actual arguments must match in type and order those of the pattern acting as its type. A variable has an identifier and a type that must match an element or pattern name. Allowed values are instances of a type (element or pattern) that does not necessarily need to match the variable's type. Variables in an architecture must have as type an element defined in the pattern that such architecture is an instance of. Each attachment includes port references to an output and an input port (see line 16 for examples). A port reference is an ordered pair of identifiers: the first one matching a variable identifier, and the second a port of the variable's instance. The architecture interface is a set of one or more port renamings. Each port renaming contains a port reference and a token with the external name of the port. An example interface is shown in line 18.

Reconfigurations are specified using ARCHERY-SCRIPT. We assume a configuration manager is associated to an architecture to monitor conditions, stopping, reconfiguring and restarting instances. Reconfiguration operations are shown in the first and second columns of Table I. Listing 2 shows an example script that creates a variable and an instance, both of type `Filter`, moves the variable to the architecture in `pf`, and attaches it to two other instances. Note that the second attachment (line 4) violates the main design principle of the pattern by connecting two filters together.

TABLE I. RECONFIGURATION OPERATIONS

Name	Operation	Modality
Create variable	`v:type`	`vnew[(a)]`
Destroy variable	`destroy(v)`	`destroy[(a)]`
Create instance	`v=type()`	`inew[(a,t)]`
Destroy instance	`clear(v)`	`clear[(a)]`
Attach	`attach(`v_f`,`p_o`,`v_t`,`p_i`)`	`attach[(`a_f`,`n_o`,`a_t`,`n_i`)]`
Detach	`detach(`v_f`,`p_o`,`v_t`,`p_i`)`	`detach[(`a_f`,`n_o`,`a_t`,`n_i`)]`
Add renaming	`show(v,`p_i`,`p_e`)`	`show[(`a_v`,`n_i`,`n_e`)]`
Remove renaming	`hide(v,`p_e`)`	`hide[(`a_v`,`n_e`)]`
Move instance	`imove(`v_s`,`v_t`)`	`imove[(`a_s`,`a_t`)]`
Move variable	`vmove(`v_t`,`v_d`)` `vmove(v)`	`vmove[(`a_t`,`a_d`)]` `vmove[(a)]`

```
1    f3:Filter = Filter();
2    vmove(f3, pf);
3    attach(p1, fwd, f3, rec);
4    attach(f1, snd, f3, rec);
```

Listing 2. ARCHERY example script

ARCHERY-CONSTRAINT aims at the specification and verification of reconfiguration constraints. In the context of this work an Hennessy-Milner logic [4] is used to express constraints. A constraint φ is an expression of the form

$$\varphi ::= \top \mid \bot \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \rightarrow \varphi_2 \mid [m]\varphi \mid \langle m\rangle\varphi.$$

The logic is interpreted over a Kripke frame $\mathfrak{F} = \langle W, \{R^m\}_{m\in MOD}\rangle$ where W are configurations, and MOD

is a set of modalities representing reconfiguration operations. A modality in MOD represents an operation and its actual parameters. The syntax for modalities is shown in Table I. Arguments `a` of the operation are of the form `[t][:q]` where `t` is a type and `q` a path, and both are optional. Types are either patterns or elements and paths are absolute and relative references to variables and ports, respectively. Paths, in turn, are of the form `n[.n]*`. A constant or a variable can be used when specifying a modality.

For example, the main design principle of the *Pipes & Filters* mentioned above, can be formalised as constraint φ_1. The constraint indicates that in any configuration, there cannot be an attachment operation that connects two variables with type `Filter`. This constraint rules out the script in Listing 2 since operation in line 4 makes φ_1 false. Other example constraints are φ_2 and φ_3 that respectively indicate that configurations that attach a pipe p (p matching a variable name in reconfiguration operations) to itself are not accepted and that it is not possible to rename a port of a pipe, but is possible to do so for a filter.

$$\varphi_1 : [attach(Filter, \_, Filter, \_)]\ \bot.$$
$$\varphi_2 : [attach(p : Pipe, \_, p : Pipe, \_)]\ \bot.$$
$$\varphi_3 : [show(Pipe, \_, \_)]\ \bot \wedge \langle show(Filter, \_, \_)\rangle\ \top$$

III. BIGRAPHICAL MODELLING

An ARCHERY-CORE specification is modelled as a bigraph, *i.e.*, an inhabitant of category $\mathrm{BG}(\Theta_S)$, where $\Theta_S = (\Sigma_S, \phi_S)$ is a sorted signature composed of a signature Σ_S and a predicate sorting ϕ_S. Table II shows the controls in Σ_S as follows: the first column indicates the control name and arity, the second column defines its activeness, and the third column provides a brief description of the represented object. The encoding of the pattern and architecture in Listing 1 yields expressions (1) and (2), respectively. The names PF, P, F of nodes with controls Pat and Elem in (1) stand for types `PipeFilter`, `Pipe`, `Filter`, respectively. These names are linked to nodes with controls Var and Inst in (2), and indicate the type of modelled variables and instances. Variable and port identifiers are modelled as names. Then, attachments and renamings are represented as linkings obtained by substituting the names of nodes representing ports, by unique names (which in our example are $ren1, ren2, att1$ and $att2$).

$$\mathsf{Pat}_{PF}.(\mathsf{Elem}_{F}.(\mathsf{In}_{rec} \mid \mathsf{Out}_{snd}) \mid \mathsf{Elem}_{P}.(\mathsf{In}_{acc} \mid \mathsf{Out}_{fwd})) \quad (1)$$

$$\begin{aligned}&\mathsf{Var}_{pf,PF}.(/r1/s1\ \mathsf{Inst}_{PF}.(\mathsf{RIn}_{ren1,r1} \mid \mathsf{ROut}_{ren2,s2}\\ &\quad \mid \mathsf{Var}_{f1,F}.({}^{ren1}/_{rec}\ {}^{att1}/_{snd}\ \mathsf{Inst}_{F}.(\mathsf{In}_{rec} \mid \mathsf{Out}_{snd}))\\ &\quad \mid \mathsf{Var}_{p1,P}.({}^{att1}/_{acc}\ {}^{att2}/_{fwd}\ \mathsf{Inst}_{P}.(\mathsf{In}_{acc} \mid \mathsf{Out}_{fwd}))\\ &\quad \mid \mathsf{Var}_{f2,F}.({}^{att2}/_{rec}\ {}^{ren2}/_{snd}\ \mathsf{Inst}_{F}.(\mathsf{In}_{rec} \mid \mathsf{Out}_{snd})))))\end{aligned} \quad (2)$$

TABLE II. CONTROLS FOR Σ_s

Ctrl	Activeness	Represented Item
Pat_x	passive	pattern
Elem_x	passive	element
In_x	atomic	in port within an instance
Out_x	atomic	out port within an instance
Inst_x	active	instance
Var_{xy}	active	variable
RIn_{xy}	atomic	renaming of an in port
ROut_{xy}	atomic	renaming of an out port

The predicate sorting ϕ_S is defined according to the sorting logic described in [6] to limit ourselves to a subcategory of bigraphs which contains the ones representing ARCHERY specifications. This logic ensures sufficient mathematical structure in the subcategory to derive labels applying the approach in [1]. The fragment of ϕ_S that concerns bigraphs modelling architectures is as follows: (a) if a node v is parent of a node u and has associated control Var, then u has associated control Inst; (b) if a node v is parent of nodes u an w, and has associated control Var, then u is equal to w; (c) if a node v is parent of a node u and has associated control Inst, then u has associated control in the set $\{\mathsf{Var}, \mathsf{Out}, \mathsf{In}, \mathsf{RIn}, \mathsf{ROut}\}$.

This fragment of ϕ_S suffices to characterise bigraphs modelling variables and architectures. They adopt the form of a list of lists, in which each list node can be a list of lists itself, as is described in Lemma 1. The rest of the predicate (not shown here) ensures that bigraphs modelling patterns, attachments, and renamings, are well-formed.

Lemma 1 (Bigraphical model of architectures)**.** Given a bigraph in $\mathrm{BG}(\Theta_S)$, modelling an architecture referenced by a variable, and $k, k' \in \mathbb{N}_0$, its place-graph is of form

$$V^k = \mathsf{Var}.\mathsf{Inst}.(U^0 \mid \ldots \mid U^j \mid \ldots \mid U^k) \quad (3)$$

where each U^j adopts the form of $V^{k'}$, or has as associated control any in the set $\{\mathsf{In}, \mathsf{Out}, \mathsf{RIn}, \mathsf{ROut}\}$.

Proof (hint). The proof is by induction over the place-graph of a bigraph in $\mathrm{BG}(\Theta_S)$.

Reconfiguration scripts specified in ARCHERY-SCRIPT are modelled as instances of $\mathrm{BG}(\Theta_D, \mathcal{R}_D)$ – a BRS with sorted signature $\Theta_D = (\Sigma_D, \phi_D)$ and set of parametric reaction rules $\mathcal{R}_D$. Signature Σ_D adds a set Σ_r of controls to Σ_S, where each control in Σ_r represents a reconfiguration operation in Table I. A reaction rule in $\mathcal{R}_D$ models the changes that a reconfiguration operation entails on a bigraph, and each control in Σ_r partially triggers one of such reaction rules. Predicate sorting ϕ_D is the conjunction of ϕ_S and the condition that if a node v with control in Σ_r has parent w, then w has a control in Σ_r as well. Therefore, any inhabitant of $\mathrm{BG}(\Theta_S)$ is also an inhabitant of $\mathrm{BG}(\Theta_D, \mathcal{R}_D)$. Nodes with control in Σ_r may be nested one inside another. Upon reaction, the external node is removed and its content is left in an active context to trigger the next reaction. This mechanism ensures the sequential activation of the nested nodes in the script. Set $\mathcal{R}_D$ is not exhaustively shown here. Instead, the form assumed by any redex L of a reaction rule (L, R) in $\mathcal{R}_M \subset \mathcal{R}_D$ is given in Lemma 2, and a parametric reaction rule, modelling attachment operations, is presented as an example.

$$\begin{aligned}&\mathsf{Var}_{f,x_1}.(\mathsf{Inst}_{x_2}.(\mathsf{Out}_{o} \mid d_0))\\ &\quad \parallel \mathsf{Var}_{t,y_1}.(\mathsf{Inst}_{y_2}.(\mathsf{In}_{i} \mid d_1)) \parallel \mathsf{Att}_{f,o,t,i,att}.d_2 \twoheadrightarrow\\ &\mathsf{Var}_{f,x_1}.({}^{att}/_{o}\ \mathsf{Inst}_{x_2}.(\mathsf{Out}_{o} \mid d_0))\\ &\quad \parallel \mathsf{Var}_{t,y_1}.({}^{att}/_{i}\ \mathsf{Inst}_{y_2}.(\mathsf{In}_{i} \mid d_1)) \parallel d_2\end{aligned} \quad (4)$$

The rest of the paper focuses on $\mathcal{R}_M$, a subset of $\mathcal{R}_D$ to modify architectures. A similar approach is taken to extend the results to $\mathcal{R}_D$. Let Q^k, with $k \in \mathbb{N}_0$, be a parametric redex that matches bigraphs of the form V^k, as defined in (3).

Lemma 2 (Redex)**.** For any (L, R) in $\mathcal{R}_M$, the redex L is of the form $Q \parallel O$, where $Q = 1$, or $Q = Q^k$, or $Q = Q^k \parallel Q^{k'}$, and O matches a node o with any control in Σ_r.

Proof (hint). The proof proceeds by showing that for any $(L, R) \in \mathcal{R}_M$ and for any bigraph a matching L, a also matches the form $Q \parallel O$.

IV. Deriving labels for reconfigurations

This section provides a derivation method that yields labels suitable for carrying out analysis and verification of reconfigurations. The method underpins the construction of an LTS that can be used to compare reconfigurations and as a Kripke frame for the logic presented in section II.

Definition 1 (Reconfiguration label derivation)**.** Given a $(L, R) \in \mathcal{R}_M$, triggered by nodes of the form O, and a bigraph $b \in \text{BG}(\Theta_D, \mathcal{R}_D)$ with form $Q \parallel O$, modelling an architecture and a reconfiguration operation, labels are derived as follows: 1) derive bigraphs o and $args$ from b with two applications of Leifer's approach [1] considering as agents, bigraphs matching Q and O, respectively; 2) obtain the pair of contexts (B, A), taken as the bigraphical label in the sequel, with $A \parallel B$ of the form $Q \parallel O$, by decomposing $args \parallel o$ as $(A \parallel B)\ d$, where d is the instantiation of the parameters of the redex L; 3) construct the label by concatenating the name of the operation modelled by the control associated with B, with the names of nodes in A, which represent variable types and identifiers, and port identifiers.

For instance, consider the attach operation in line 3 of Listing 2: `attach(p1,fwd,f3,rec)`. Applying the method to the corresponding bigraph and reaction rule yields the bigraphical label

$$(\mathsf{Att}_{p1,fwd,f3,rec,att3},\ \mathsf{Var}_{p1,P}.(\mathsf{Inst}_P.(\mathsf{Out}_{fwd} \mid Id)) \parallel \mathsf{Var}_{f3,F}.(\mathsf{Inst}_F.(\mathsf{In}_{rec} \mid Id))) \quad (5)$$

representing the operation and its arguments, which leads to label `attach(Pipe : p1, fwd, Filter : f3, rec)`. The application of this method yields labels according to the form shown in the third column of Table I.

Theorem 1 (Reconfiguration labels)**.** Labels derived through the method given in Definition 1 for a bigraph in $\text{BG}(\Theta_D, \mathcal{R}_D)$ are of the form $op(arg_1, \ldots arg_n)$, where op is the name of a reconfiguration operation, and each arg_i is an actual parameter.

Proof (hint). The proof assumes a bigraph of the form $Q \parallel O$ and for each $(L, R) \in \mathcal{R}_M$ finds the form of the minimal context for each application of Leifer's approach [1]. Such forms are obtained by transforming expressions through the categorical and bigraphical axioms collected in [8].

An LTS can be constructed with the derived labels and used as a model for the logic introduced in section II. Moreover, well known behavioural equivalences, such as bisimilarity ($\sim$), can be applied to compare two reconfigurations. Our next result shows that bisimilarity is congruential within such LTSs, *i.e.*, if agents a and b are bisimilar ($a \sim b$), for any context C, we have that Ca and Cb are bisimilar as well ($Ca \sim Cb$).

Theorem 2 (Congruential bisimulation)**.** The first and second components of a bigraphical label (B, A), derived using the method given in Definition 1, are minimal contexts that characterise the operation and its actual parameters, respectively, and yield an LTS in which bisimulation is a congruence.

Proof (hint). The minimality of each component of (B, A) and what they model are enforced by construction. The congruential bisimulation in an LTS with labels (B, A), requires showing that the LTS that results from using as label the component A (B, respectively), is the same as the one constructed applying [1] considering bigraphs modelling operations (architectures, respectively) as agents. Then, the proof can proceed in a similar way to that of Theorem 7.16 in [9].

V. Conclusion and Future Work

This work presents a label derivation method and extends structural semantics for the ARCHERY language. Architectures and reconfigurations specified in ARCHERY are encoded into a BRS, and the derivation method in Definition 1 allows for the construction of an LTS in which states and transitions represent configurations and reconfigurations. Then, such LTS allows for the comparison of reconfigurations using bisimulation, and becomes a Kripke frame over which ARCHERY constraints, expressed in the Hennessy-Milner logic [4], can be interpreted.

Future and ongoing work includes adding hybrid features and recursion (in the style of modal $\mu-$calculus) to the logic.

Acknowledgment

This work is funded by ERDF - European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project FCOMP-01-0124-FEDER-020537

References

[1] J. J. Leifer and R. Milner, "Deriving bisimulation congruences for reactive systems," in *Proceedings of the 11th International Conference on Concurrency Theory*, ser. CONCUR '00. London, UK, UK: Springer-Verlag, 2000, pp. 243–258.

[2] A. Sanchez, L. S. Barbosa, and D. Riesco, "A language for behavioural modelling of architectural patterns," in *Proceedings of the Third Workshop on Behavioural Modelling*, ser. BM-FA '11. New York, NY, USA: ACM, 2011, pp. 17–24.

[3] ——, "Bigraphical modelling of architectural patterns," in *Formal Aspects of Component Software. 8th International Symposium, FACS 2011, Oslo, Norway, September 14-16, 2011, Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 7253. Berlin Heidelberg: Springer, 2012, pp. 313–330.

[4] M. Hennessy and R. Milner, "Algebraic laws for nondeterminism and concurrency," *J. ACM*, vol. 32, no. 1, pp. 137–161, Jan. 1985.

[5] R. Milner, "Bigraphical reactive systems," in *CONCUR*, ser. Lecture Notes in Computer Science, K. G. Larsen and M. Nielsen, Eds., vol. 2154. Springer, 2001, pp. 16–35.

[6] G. Perrone, "Domain-specific modelling languages in bigraphs," Ph.D. dissertation, IT University of Copenhagen, 2013.

[7] J. F. Groote, A. Mathijssen, M. Reniers, Y. Usenko, and M. van Weerdenburg, "The formal specification language mCRL2," in *Methods for Modelling Software Systems: Dagstuhl Seminar 06351*, 2007.

[8] R. Milner, "Axioms for bigraphical structure," *Mathematical Structures in Computer Science*, vol. 15, no. 6, pp. 1005–1032, 2005.

[9] ——, *The space and motion of communicating agents*. Cambridge University Press, 2009, vol. 54.