Autonomous Agents and Multi-Agent Systems manuscript No.
(will be inserted by the editor)

An Agent-Based Model of the Emergence and Evolution of a Language System for Boolean Coordination

Josefina Sierra-Santibáñez

Received: 30 April 2017 / Accepted: 16 February 2018

Abstract This paper presents an agent-based model of the emergence and evolution of a language system for Boolean coordination. The model assumes the agents have cognitive capacities for invention, adoption, abstraction, repair and adaptation, a common lexicon for basic concepts, and the ability to construct complex concepts using recursive combinations of basic concepts and logical operations such as negation, conjunction or disjunction. It also supposes the agents initially have neither a lexicon for logical operations nor the ability to express logical combinations of basic concepts through language. The results of the experiments we have performed show that a language system for Boolean coordination emerges as a result of a process of self-organisation of the agents' linguistic interactions when these agents adapt their preferences for vocabulary, syntactic categories and word order to those they observe are used more often by other agents. Such a language system allows the unambiguous communication of higher-order logic terms representing logical combinations of basic properties with non-trivial recursive structure, and it can be reliably transmitted across generations according to the results of our experiments. Furthermore, the conceptual and linguistic systems, and simplification and repair operations of the agent-based model proposed are more general than those defined in previous works, because they not only allow the simulation of the emergence and evolution of a language system for the *Boolean coordination* of *basic properties*, but also for the *Boolean coordination of higher-order logic terms of any Boolean type* which can represent the meaning of nouns, sentences, verbs, adjectives, adverbs, prepositions, prepositional phrases and subexpressions not traditionally analysed as forming constituents, using linguistic devices such as syntactic categories, word order and function words.

Keywords Origins and Evolution of Language · Agent-Based Models · Boolean Coordination · Logical Representations and Reasoning · Language Games

This work has been partially supported by funds from the Spanish Ministry for Economy and Competitiveness (MINECO) and the European Union (FEDER funds) under grant GRAMM (TIN2017-86727-C2-1-R), grant APCOM (TIN2014-57226-P), and grant 2014-SGR-890.

Josefina Sierra-Santibáñez
Department of Computer Science, Technical University of Catalonia, Barcelona, Catalonia, Spain
E-mail: Maria.Josefina.Sierra@upc.edu

1 Agent-Based Models of Language Emergence and Evolution

The question of the *origins and evolution of language* has received much interest in the last two decades. It has been approached from different disciplines such as anthropology, historical linguistics, evolutionary biology or artificial intelligence. In particular, in artificial intelligence *agent-based models*, implemented and tested in computer simulations, have been used to study the emergence and evolution of language [23], [9], [33], [12]. Depending on whether these models emphasise the role of biological evolution or the role of cultural evolution, agent-based models can be classified into two areas: *Biolinguistics*, which assumes that the structure of language is determined to a large extent by biological aspects [18]; and *Evolutionary Linguistics*, which supposes that language is primarily shaped by cultural factors [36,48].

Agent-based models involve a population of autonomous agents that interact with each other playing language games. A language game [58] is typically an interaction between two agents, a speaker and a hearer. The speaker has a communicative goal, conceptualises the world for language, transforms this conceptualisation into an utterance, and communicates that utterance to the hearer. The hearer tries to parse the utterance, reconstruct its meaning, and map it onto its own internal representation of the world. Speaker and hearer normally use extralinguistic means to determine the outcome of a language game and, depending on that outcome, employ different strategies to expand and adapt their internal languages in order to be more successful in future language games.

The agents in these models are initially endowed with some cognitive abilities that are assumed to be necessary to take part in the particular language game studied and to develop a language system that enables the population to succeed in their linguistic interactions (e.g. the ability to construct complex concepts, or to use and detect linguistic devices such as word order, syntactic categories or case markers). In the simulations, the agents are made to play a series of language games where they configure possible language systems and try them out. The goal of the experiments is to find out whether the population as a whole communicates effectively, and to observe the conceptualisations and linguistic constructions that emerge in the population as a result of the processes of collective invention and negotiation, as well as the evolution over time of several features of the emerging language systems, such as the average size of the agents' lexicons, the vocabulary and grammatical constructions preferred by the population, or the stability of such language systems across different generations.

Constructing agent-based models of different aspects of language emergence and evolution, and testing their effectiveness through agent-based simulations or robotic experiments is not only useful for understanding the origins and evolution of language, it also provides valuable insight for guiding the design of truly autonomous agents which should be able to construct symbolic representations of unpredictable environments by themselves, and communicate such representations to human or artificial agents using some form of natural language. In particular, a large body of work on the origins and evolution of language has been concerned with the grounding of language, i.e. how words and sentences get their meaning [44,53,51,48]. Language acquisition and language use play a very important role in the development of grounded representations, which is one of the major outstanding problems facing Artificial Intelligence (AI) according to [57], who state that *"The lack of a comprehensive understanding of grounding is a major impediment to the development of genuinely intelligent systems that can generate their own representations and that know what they are doing"*, and also that *"A sound grounding capability provides basic infrastructure for cognition and intelligence. Consequently, how, and how well an agent is grounded is of significant interest and crucial importance in AI."*

Theories of language evolution study language change at two different levels: that of language systems and that of language strategies. *Language systems* capture the regularity observed in some part of the vocabulary or the grammar of a language [47], for example, colour terms, case markers, tense-aspect distinctions or coordination. Language systems group a set of paradigmatic choices both on the side of meaning (the conceptual system) and on the side of form (the linguistic system). The *conceptual system* contains the semantic distinctions that are expressible in the language system and can therefore be used as building blocks for conceptualisation. The *linguistic system* includes the syntactic categories, lexicon and/or grammatical constructions necessary to turn a conceptualisation into a concrete utterance. Linguists call the approach underlying a language system a *language strategy*. A natural language, such as Chinese, English or Spanish, comprises many different language systems, which are closely integrated.

Agent-based experiments in evolutionary linguistics aim to explain how particular language systems and language strategies may emerge and evolve [50]. Examples of language systems that have been studied using agent-based models are: (1) *case systems* to express the role of participants in events [46,7]; (2) *tense-aspect distinctions* [21]; (3) *agreement markers* to group words together [8]; (4) *word-order-based phrase-structure* to indicate how the arguments of different properties and relations in an utterance relate to each other [52,19]; or (5) *vocabularies* in co-evolution with *semantic categories* [51,31,32].

In this paper we study the emergence and evolution of a language system for *coordination*, a linguistic phenomenon which involves combining syntactic categories using coordinators like *and, or* and *but*, in order to communicate logical combinations of certain concepts. A range of different categories may undergo coordination, as can be seen in the following examples:
a. Terry [$_{np\backslash s}$ [jumps] and [runs]].
b. Jo read the [$_{n}$[book] or [magazine]].
c. John ran [$_{s\backslash s}$[in Pittsburgh yesterday] and [in Cleveland today]].
d. [$_{s/np}$ [Joan bought] or [Peter found]] the dog.
e. After the fight, John was [$_{n/n}$[bloody] but [unbowed]].

The fundamental principle of coordination is that any pair of categories of the same Boolean type can be coordinated to produce a result of the same category [11]. A Boolean type is simply one that eventually produces a Boolean category after applying to all of its arguments. In fact, any category producing a nominal or sentential result is Boolean. Thus, nouns, sentences, adjectives, adverbs, prepositions, prepositional phrases and verbs, among others, can be coordinated.

The previous examples share the property that the coordinator appears as a Boolean operator that has been distributed past the coordinate categories. For example, *'Terry and Sandy studied'* can be read as *'Terry studied and Sandy studied'*, where the Boolean operator applies not only to the proper nouns but to the complete coordinate clauses. This form of coordination, known as *Boolean coordination* and exhibited by the previous sentences, is the type of coordination we will focus on.

There is also a different type of coordination, often referred to as *nonBoolean coordination*, that falls outside the scope of this paper. Examples of nonBoolean coordination are *'Terry and Sandy met'*, *'Broke is strong enough to carry Jody and Francis'*, *'the dogs and the cats fought'* or *'the professors and the dean met'*. To represent the meaning of these sentences, a scheme is needed to coordinate noun phrases to produce sets, which can then be interpreted collectively or distributively. Furthermore, such scheme must be able to account for examples coordinating singular and plural noun phrases, as in the examples.

Boolean coordination is not only an interesting and well studied linguistic phenomenon, understanding how logical combinations of concepts can be constructed and communicated using language is of great value to AI. Logical operations such as negation, conjunction or disjunction allow building complex concepts from simpler ones. Logical combinations are frequently used in natural language descriptions of objects, events, actions and situations, as well as in the definition of new concepts from previous ones. The ability to construct and communicate logical combinations plays, therefore, a crucial role in the construction of sound grounded representations by autonomous agents from their interactions with the environment and other agents. Logical operations are also very important for intellectual development [38], because they allow the generation of structured units of meaning and lay the foundations for deductive reasoning. In [39,40], we proposed a language game called the *evaluation game* and used it to study the *emergence of logical operations*, such as conjunction, negation or disjunction, from a semantic point of view. In the present paper, we focus on the linguistic expression of logical combinations, using syntactic devices such as syntactic categories, word order and function words.

The rest of this paper is organised as follows. Firstly, we describe the conceptual system of a language system for Boolean coordination. In particular, we specify the formalism the agents use to represent the concepts they try to communicate in the language game studied in this paper, which addresses the communication of logical combinations of basic concepts. Secondly, we present the linguistic system of a language system for Boolean coordination. Specifically, we introduce the grammatical formalism the agents use to represent the syntactic categories and lexicon they construct during the simulations, and the linguistic devices they may use to express the relation between the Boolean operator of a logical combination and its arguments in a sentence. Thirdly, we describe the particular type of linguistic interaction that allows the agents to construct a language system for Boolean coordination, paying special attention to cognitive abilities for invention, adoption, abstraction, repair and adaptation. Fourthly, we present the results of some experiments which study the emergence of such a language system and its transmission across generations. And finally, we summarise the main contributions of the paper.

2 Conceptual System

We consider a scenario in which a group of agents try to communicate about *subsets of objects* of the set of all the objects pasted on a white-board situated in front of them [49]. We assume the agents are capable of detecting some properties of the objects, e.g. whether they are light, dark, or in upper, lower, left or right position. These properties allow them to build perceptually grounded representations of the subset of objects that constitutes the topic of the language game they participate in at a given moment. For the purposes of this paper, such properties are represented by unary predicate symbols (or constants of type $Ind \rightarrow Bool$ of higher-order logic[1]) in the agents' memories. In particular, the formalism the agents use to represent the meanings they build during the simulations is the *simply typed λ-calculus*, and specifically *higher-order logic*.

We also suppose that, at the beginning of a simulation run, the agents have already learnt to perform some *Boolean operations,* such as negation($\neg$), conjunction($\wedge$) or disjunction($\vee$), at the semantic level. These operations are represented in their memories using logical con-

[1] *Bool* is the type of Boolean values and *Ind* the type of individuals. See section 2.1.1 for a definition of *symple types* in the λ-Calculus.

nectives such as `and`, `not` and `or`, or more generally the polymorphic coordination combinators $Coor_\sigma(and)$ and $Coor_\sigma(or)$ defined in section 3.2, and the negation operator Neg_σ defined in section 3.3. These logical operators allow the agents to build complex meanings such as `'the objects which are either up or to the left, but not both'`, which is a logical combination of basic properties and can be represented by the higher-order logic term[2] $\lambda x.\text{and}(\text{or}(\text{up}(x))(\text{le}(x)))(\text{not}(\text{and}(\text{up}(x))(\text{le}(x))))$.

Furthermore we assume that, at the beginning of a simulation run, the agents have a common lexicon for referring to basic properties. However, they do not have lexical entries for logical operators (e.g. `and`,`not`,`or`). Therefore, they cannot communicate the logical terms they can construct with such operators to other agents. The goal of the simulations is that the agents learn to express such complex meanings and jointly construct a set of linguistic conventions that allows the communication of higher-order logic terms expressing Boolean combinations of basic concepts.

2.1 Simply Typed λ-Calculus

The λ-calculus offers an elegant solution not only to the problem of providing denotations for the basic expressions of a language but also for productively composing these basic meanings into larger units. It was invented by Church [13], with the goal of defining a uniform language with which to describe functions. But in this paper, we will apply it to natural-language: firstly, as a formalism in which to represent the meanings of the basic expressions of a language; and secondly, as the *compositional* method that defines the meaning of higher-order logic terms expressing logical combinations of basic concepts in terms of the meanings of these concepts.

2.1.1 Simple Types

In first-order logic there are two types of expressions: terms and formulas. In the simply typed λ-calculus there are infinitely many types of expressions. These types are all constructed from a finite set of basic types. A common choice of basic types in linguistics consists of a type *Bool* of *Boolean values* for propositions and a type *Ind* of *individuals*. The full set of types is then built up hierarchically by closing the set of basic types under the construction of total function types.

Definition 1 *(Simple Types)* From a nonempty set *BasTyp* of *basic types*, the set *Typ* of types is the smallest set such that
a. $BasTyp \subseteq Typ$
b. $(\sigma \to \tau) \in Typ$ if $\sigma, \tau \in Typ$

A type of the form $(\sigma \to \tau)$ is said to be a *functional type*, the elements of which map objects of type σ onto objects of type τ. For instance, the functional type $(Ind \to Bool)$ is the type of functions from individuals to propositions, or in more familiar terms, the type of properties or unary predicates. There are types for Boolean operations, e.g. $(Bool \to Bool)$ is the type of unary Boolean functions, usually assigned to the negation operator. We can also construct second-order types, such as $((Ind \to Bool) \to (Ind \to Bool))$, which is the

[2] up and le are constants of type $Ind \to Bool$ denoting the properties of being in an upper and left position respectively. If the agents used First Order Logic instead of Higher Order Logic to represent their meanings, they would approximate the meaning of this noun phrase by using formula $(\text{up}(x) \vee \text{le}(x)) \wedge \neg(\text{up}(x) \wedge \text{le}(x))$.

type of a function mapping unary predicates into unary predicates, i.e. a property-modifier type, which is usually assigned to adjectives, prepositional phrases and relative clauses[3].

2.1.2 λ-Terms

We build up expressions in the λ-calculus out of variables and constants of the various types. Thus, we assume for each type τ that we have the following sets.
a. Var_τ: a countably infinite set of variables of type τ.
b. Con_τ: a collection of constants of type τ.

Variables will be employed as in first-order logic to express binding operations such as quantification, but they will also be used in the construction of new function terms.

Definition 2 *(λ-Terms)* The collections Term_τ of *λ-terms* of type τ are defined as the smallest sets such that
a. $\text{Var}_\tau \subseteq \text{Term}_\tau$
b. $\text{Con}_\tau \subseteq \text{Term}_\tau$
c. $\alpha(\beta) \in \text{Term}_\tau$ if $\alpha \in \text{Term}_{\sigma\to\tau}$ and $\beta \in \text{Term}_\sigma$
d. $\lambda x.\alpha \in \text{Term}_\tau$ if $\tau = \sigma \to \rho$, $x \in \text{Var}_\sigma$ and $\alpha \in \text{Term}_\rho$

A term of the form $\alpha(\beta)$ is said to be a *functional application* of α to β. If *square* is a constant of type $Ind \to Bool$ and o_1 is a constant of type *Ind*, then $square(o_1)$ will be a term of type *Bool*. Similarly, if $upper$ is a constant of type $(Ind \to Bool) \to (Ind \to Bool)$, then[4] $upper(square)$ will be a term of type $Ind \to Bool$, and $upper(square)(o_1)$ a term of type *Bool*. A term of the form $\lambda x.\alpha$ is said to be a *functional abstraction*. The λ-calculus gets its name from the original notation involving Greek letter λ, used by Church in the definition of *abstraction terms*. Application must always involve a functional type, while abstraction always produces a functional type. For instance, assuming x and r_1 are a variable and a constant of type *Ind*, and *move* is a constant of type $Ind \to (Ind \to Bool)$, we have an abstraction term such as $\lambda x.move(x)(r_1)$, which denotes a function from individuals to the proposition that they were moved by r_1.

In this paper we will only use unary functions, although they may apply to a single argument to produce a result which itself might be a function. The introduction of n-ary tuples into the λ-calculus is done by composing *pairs* (also called *products*). There are two operations, commonly referred to as *currying* and *uncurrying* that allow mapping back and forth between functions defined in terms of products and unary functions. The result of currying a function is to take a function defined on pairs and convert it into a function that takes the elements of the pair one at a time to produce a result. Conversely, uncurrying takes a function of type $\sigma \to \tau \to \rho$, which applies to an argument of type σ to produce a function from type τ objects into type ρ objects, and converts it into a function which takes its arguments of type σ and τ simultaneously in the form of a pair. The important fact to note is that currying and uncurrying establish a one-to-one relationship between objects of type $\sigma \times \tau \to \rho$ and those of type $\sigma \to \tau \to \rho$. Thus, adding products to the λ-calculus does not really provide any additional representational power.

[3] When we omit parentheses from functional types, we assume associativity is applied to the right, so that $Ind \to Ind \to Bool$ is equivalent to $Ind \to (Ind \to Bool)$.

[4] Assuming *square* and $upper$ have the standard natural language interpretations, $square(o_1)$ will be true if o_1 denotes an object that is a square, $upper(square)$ will denote the property of being a square situated in an upper position, and $upper(square(o_1))$ will be true if o_1 is a square which is in an upper position.

Following [37], it is common in presentations of natural-language semantics for all functions and relations to be in curried form. Thus rather than treat the *moving* relation as a binary[5] relation of type $(\texttt{Ind} \times \texttt{Ind}) \rightarrow \texttt{Bool}$, we curry it to produce a higher-order function of type $\texttt{Ind} \rightarrow \texttt{Ind} \rightarrow \texttt{Bool}$. The benefit of currying from a natural language perspective is that we have a simple term in the object language, namely `move(x)`, to correspond to the verb phrase *'moves x'*.

2.2 Higher-Order Logic

By higher-order logic we mean a logic that involves constants and variables of arbitrary finite type, with every kind of object being a first-class citizen over which quantification and abstraction may be performed [11]. We treat higher-order logic as an application of the simply typed λ-calculus. In particular, we assume that the set of basic types of higher-order logic is $BasTyp = \{Bool, Ind\}$. This determines the type theory of higher-order logic.

Definition 3 *(Higher-Order Logical Constants)* We assume the existence of the following collection of *logical constants*
a. $\mathbf{not} \in \mathbf{Con}_{Bool \rightarrow Bool}$
b. $\mathbf{and} \in \mathbf{Con}_{Bool \rightarrow Bool \rightarrow Bool}$
c. $\mathbf{eq}_\tau \in \mathbf{Con}_{\tau \rightarrow \tau \rightarrow Bool}$
d. $\mathbf{every}_\tau \in \mathbf{Con}_{(\tau \rightarrow Bool) \rightarrow Bool}$
e. $\iota_\tau \in \mathbf{Con}_{(\tau \rightarrow Bool) \rightarrow \tau}$

The reason these are called logical constants has to do with their receiving fixed interpretations with respect to any model of higher-order logic: **not** and **and** are simply the higher-order analogues of the usual operations of negation and conjunction. The indexed constant $\mathbf{eq}_\tau$ is interpreted as the equality relation between two objects type τ.

We also have a family of universal *generalized quantifiers*[6] for all types τ. Because λ-abstraction is the only way to bind variables in the λ-calculus, quantifiers in higher-order logic are not treated as binding operators, as is standard in first-order logic. Instead, we treat quantifiers over objects of type τ as properties of properties of type τ. In particular, if $\tau = Ind$, constant $\mathbf{every}_{\mathbf{Ind}}$ is of type $(Ind \rightarrow Bool) \rightarrow Bool$. A formula $\forall x(\phi)$ in first-order logic will be rendered as $\mathbf{every}_{\mathbf{Ind}}(\lambda x.\phi')$, where ϕ' is the translation of ϕ into higher-order logic. The truth conditions will be such that $\mathbf{every}_{\mathbf{Ind}}(P)$ will denote $\mathbf{yes} \in \mathbf{Dom}_{\mathbf{Bool}}$ in a model if and only if P denotes the property that is true of every individual.

Finally, we have a constant ι_τ, called the *description operator*, for each type τ. The description operator is used for mapping singleton sets to the object they contain. More precisely, $\iota_\tau(P)$ picks out an element a of type τ if a is the unique element that has the property denoted by P. If P does not denote a singleton set, $\iota_\tau(P)$ still denotes an element of type τ, but the identity of the element is unconstrained by the logic.

We adopt a standard set of abbreviations for the logical constants, and define some new connectives in terms of them.

a. $\phi \wedge \psi \stackrel{\text{def}}{=} \mathbf{and}(\phi)(\psi)$
b. $\neg\phi \stackrel{\text{def}}{=} \mathbf{not}(\phi)$

[5] Treating *move* as a binary relation means interpreting $move(x, r_1)$ as *'r_1 moves x'*. Currying *move* implies that $move(x)$ means *'moves x'*, and $move(x)(r_1)$ means *'r_1 moves x'*.

[6] Generalized quantifiers, e.g. $\texttt{every}_\tau$ or $\texttt{some}_\tau$, quantify over objects of type τ.

c. $\alpha =_\tau \beta \stackrel{\text{def}}{=} \mathbf{eq}_\tau(\alpha)(\beta)$
d. $\forall x^\tau(\phi) \stackrel{\text{def}}{=} \mathbf{every}_\tau(\lambda x.\phi)$
e. $\mathbf{or}(\phi)(\psi) \stackrel{\text{def}}{=} \phi \vee \psi \stackrel{\text{def}}{=} \neg((\neg\phi) \wedge (\neg\psi))$
f. $\phi \rightarrow \psi \stackrel{\text{def}}{=} (\neg\phi) \vee \psi$
g. $\exists x^\tau(\phi) \stackrel{\text{def}}{=} \mathbf{some}_\tau(\lambda x.\phi)$

Just as in first-order logic, we can define the existential quantifier in terms of the universal quantifier.

$$\mathbf{some}_\tau \stackrel{\text{def}}{=} \lambda P^{\tau\rightarrow\mathbf{Bool}}.\mathbf{not}(\mathbf{every}_\tau(\lambda x^\tau.\mathbf{not}(P(x))))$$

The definitions of logical notions for higher-order logic are as follows: a *formula* is an expression of type *Bool*; a formula is *valid* if it is true in every model; and two terms are *logically equivalent* if they have the same value in every model.

3 Linguistic System

As mentioned earlier, we suppose that at the beginning of a simulation run the agents have a common lexicon for referring to basic properties. For example, the agents' lexicons[7] may initially contain the following six lexical entries.

$sup \Rightarrow \texttt{up}:pr:1.0$	$izq \Rightarrow \texttt{le}:pr:1.0$	$cla \Rightarrow \texttt{li}:pr:1.0$
$inf \Rightarrow \texttt{do}:pr:1.0$	$der \Rightarrow \texttt{ri}:pr:1.0$	$osc \Rightarrow \texttt{da}:pr:1.0$

The agents use *categorial grammar* to represent the linguistic system of the language system they build during the simulations. A lexical entry $e \Rightarrow \alpha : A : s$ is an association (e, A, α, s) between a linguistic expression e, a syntactic category A, a higher-order logic term α, and a score s, such that α is a formal representation in higher-order logic of the meaning of e, A is its syntactic category, and s is the score of the association. The score of a lexical entry is a real number in the interval $[0.0, 1.0]$ that estimates its usefulness in previous communication. For example, the first lexical entry above (i.e. $sup \Rightarrow \texttt{up}:pr:1.0$) states: 1) that expression *sup* denotes the subset of objects that are in an upper position, or the property of being in an upper position (i.e. `up`); 2) that the syntactic category of *sup* is *pr* (i.e. a *property*[8]); and 3) that the score of this lexical entry is 1.0.

3.1 Categorial Grammar

In *categorial grammar* [2,6,29,30], every syntactic category corresponds to some higher-order type, with the assumption being that expressions of each category can be assigned meanings of the appropriate type. We assume some finite set $BasCat = \{np, n, s, pr\}$ of basic categories, which abbreviate *noun phrase, noun, sentence* and *property* respectively, and are associated with the following higher-order types $Type(np) = Ind$, $Type(s) = Bool$, $Type(n) = Ind \rightarrow Bool$ and $Type(pr) = Ind \rightarrow Bool$. *BasCat* is used to generate an infinite set of functional categories, each of which specifies (possibly complex) argument and result categories.

[7] See definition 6 on page 9 for formal definitions of lexicon and lexical entry.
[8] See section 3.1 for a formal definition of the set of syntactic categories used in this paper.

Definition 4 *(Syntactic Categories)* The set *Cat* of *syntactic categories* determined by the set *BasCat* is the smallest set such that
a. $BasCat \subseteq Cat$
b. $(A\backslash B) \in Cat$ if $A, B \in Cat$
c. $(B/A) \in Cat$ if $A, B \in Cat$

A category B/A or $A\backslash B$ is said to be a *functor* category, and to have an argument category A and a result category B. A *functional category* of the form B/A is called a *forward functor* and looks for its argument A to the right, while the *backward functor* $A\backslash B$ looks for its argument to the left.

The fundamental operation in *applicative categorial grammar* is the concatenation of an expression assigned to a functional category and an expression of its argument category to form an expression of its result category, with the order of concatenation being determined by the functional category. For instance, $np\backslash s$ would be the type assigned to both verb phrases and intransitive verbs: they look backward for a noun phrase to produce a sentence. The category n/n is used for prenominal adjectives: they look forward for nouns to produce nouns. The category $n\backslash n$, on the other hand, is used for postnominal modifiers. Similarly, $(np\backslash s)/np$ would be assigned to transitive verbs which look for an object noun phrase to the right and a subject noun phrase to the left[9].

The correspondence between the set of basic syntactic categories *BasCat* and the set of higher-order types *Typ* is extended to the set *Cat* of all syntactic categories as follows.

Definition 5 *(Type Assignment)* We extend $Type : BasCat \to Typ$ to functor categories as follows
$Type(A\backslash B) = Type(A) \to Type(B)$
$Type(B/A) = Type(A) \to Type(B)$

Definition 6 *(Categorial Lexicon)* A categorial lexicon is a relation $Lex \subseteq BasExp \times (Cat \times \mathrm{Term}_\tau)$ such that if a lexical entry $\langle e, \langle A, \alpha \rangle\rangle \in Lex$, then $\alpha \in \mathrm{Term}_{Type(A)}$.

Agents assign a score s to each lexical entry $\langle e, \langle A, \alpha \rangle\rangle$. Thus we abbreviate lexicon entries such as $\langle e, \langle A, \alpha \rangle\rangle$ as $e \Rightarrow \alpha : A : s$, where e is a basic expression, A is its syntactic category, α is a higher-order term of type $Type(A)$ denoting e's meaning, and s is the score of the lexical entry.

Categorial grammar *phrase-structure rules* are based on functional application, and on the intuitive notions of B/A as a forward-looking functor and $A\backslash B$ as backward-looking.

Definition 7 *(Application Schemes)* The following phrase-structure application schemes are assumed:

$$\alpha : B/A : s_1, \quad \beta : A : s_2 \quad \Rightarrow \quad \alpha(\beta) : B : s_1 \cdot s_2 \qquad \text{(Forward)}$$
$$\beta : A : s_1, \quad \alpha : A\backslash B : s_2 \quad \Rightarrow \quad \alpha(\beta) : B : s_1 \cdot s_2 \qquad \text{(Backward)}$$

The forward scheme states that if e_1 is an expression of syntactic category B/A with meaning α and score s_1, and e_2 is an expression of syntactic category A with meaning β and score s_2, then $e_1 \cdot e_2$ (the concatenation of e_1 and e_2) is an expression of syntactic category B with meaning $\alpha(\beta)$ and score the product of s_1 and s_2. Similarly, the backward scheme states that if e_1 is an expression of syntactic category A with meaning β and score s_1, and e_2 is an expression of syntactic category $A\backslash B$ with meaning α and score s_2, then $e_1 \cdot e_2$ (the concatenation of e_1 and e_2) is an expression of syntactic category B with meaning $\alpha(\beta)$ and score the product of s_1 and s_2.

[9] We will omit parentheses within categories, taking the forward slash to be left associative, the backward slash to be right associative, and the backward slash bind more tightly than the forward slash. Thus $(np\backslash s)/np$ is equivalent to $np\backslash s/np$.

3.2 Type-Logical Approach to Coordination

One of the best known linguistic applications of categorial grammar is to *Boolean coordination*. The associativity of the Lambek calculus[10] [29,30], together with some polymorphic categories required by coordinators such as *and* and *or*, allow formalising a wide range of instances of coordination of Boolean categories, including both traditional constituents and categories not traditionally analysed as forming constituents.

[37] introduced a categorial notion of *coordination* into his grammars, and this construction was later generalized by [20,26]. [43] showed how the logic of categorial-grammar category assignments could be used to provide a syntactic basis for Gazdar's semantic conception.

The categorial-grammar approach to Boolean coordination is based on the principle that two categories of the same Boolean type may be coordinated to produce a category of the same Boolean type [20,26].

A *Boolean type* is simply one that eventually produces a Boolean category after applying to all of its arguments.

Definition 8 *(Boolean Types)* The set *BoolType* of *Boolean types* is the smallest set such that
a. $Bool \in BoolType$
b. $\tau \rightarrow \sigma \in BoolType$ if $\tau \in Type$ and $\sigma \in BoolType$

Thus a Boolean type is of the form $\sigma_1 \rightarrow \ldots \rightarrow \sigma_n \rightarrow Bool$, for arbitrary types σ_i. In fact, any category producing a nominal or sentential result is Boolean. For instance, nouns, adjectives, prepositions, prepositional phrases and verbs are all Boolean categories. Therefore, expressions of any of these categories can be coordinated, as we saw in section 1.

Examples of Boolean coordination share the property that the coordinator appears as a Boolean operator that is distributed past the coordinate categories. Thus the *semantics of coordination* can be handled by means of the following generalized coordination combinator that applies pointwise to pairs of Boolean terms of the same type [11].

Definition 9 *(Polymorphic Coordination Combinator) If* $\sigma \in$ BoolType*, then* $\mathbf{Coor}_\sigma$ *is of type* $(\text{Bool} \rightarrow \text{Bool} \rightarrow \text{Bool}) \rightarrow \sigma \rightarrow \sigma \rightarrow \sigma$ *and is defined recursively by*
a. $\mathbf{Coor}_{\mathbf{Bool}}(\alpha)(\beta_1)(\beta_2) \stackrel{\text{def}}{=} \alpha(\beta_1)(\beta_2)$
b. $\mathbf{Coor}_{\gamma\rightarrow\tau}(\alpha)(\beta_1)(\beta_2) \stackrel{\text{def}}{=} \lambda x^{\gamma}.\mathbf{Coor}_{\tau}(\alpha)(\beta_1(x))(\beta_2(x))$

Note that Coor_σ defines a family of combinators, one for each Boolean type σ. The parameter σ must be instantiated in order to determine a proper λ-term. For example, if `li` and `up` are constants of type $Ind \rightarrow Bool$, x is a variable of type Ind, and `and` is the ordinary higher-order logical constant of conjunction of type $Bool \rightarrow (Bool \rightarrow Bool)$, the following term denotes the property of being light and in an upper position, or the set of objects which are at the same time light and in an upper position.

$$\begin{aligned} & Coor_{Ind\rightarrow Bool}(and)(light)(upper) \\ & \equiv \lambda x.Coor_{Bool}(and)(light(x))(upper(x)) \\ & \equiv \lambda x.and(light(x))(upper(x)) \end{aligned} \tag{1}$$

[10] Lambek extended pure applicative categorial grammar according to a simple algebraic interpretation of the slashes. In the Lambek Calculus an expression is assigned to category A/B (respectively, to category $B\backslash A$) if and only if when it is followed (respectively, preceded) by an expression of category B, it produces an expression of category A. However, *applicative categorial grammar* only respects one half of the biconditional.

Arguments of higher arity (e.g. $Coor_{Ind\to(Ind\to Bool)}$) are treated similarly, reducing their arity one argument at a time.

To analyse the basic *syntax of coordination*, we need coordinator categories for every Boolean category, where a *Boolean category* is simply a category whose type is a Boolean type. In particular, we adopt the following infinite family of lexical entries [43]:

$$y \Rightarrow \mathbf{Coor}_{\sigma}(\mathbf{and}) : A\backslash A/A \qquad Typ(A) = \sigma \in BoolType$$
$$o \Rightarrow \mathbf{Coor}_{\sigma}(\mathbf{or}) : A\backslash A/A \qquad Typ(A) = \sigma \in BoolType$$

The following example illustrates simple property coordination. Note that we have shortened the two-step application of phrase structure schemes to one step, and that we have simplified the semantics by using the equivalence in (1).

$$\dfrac{\dfrac{cla}{\mathbf{li}:pr}\text{Lx} \quad \dfrac{y}{\mathbf{Coor_{Ind\to Bool}}(\mathbf{and})}\text{Lx} \quad \dfrac{sup}{\mathbf{up}:pr}\text{Lx}}{\lambda x.\mathbf{and}(\mathbf{li}(x))(\mathbf{up}(x)) : pr}$$

3.3 Ambiguity and Scope Markers

The following categorial lexicon could be constructed by an agent during the simulations. It uses three types of syntactic categories (*pr, pr/pr* and *pr\pr/pr*), an instantiation of the negation operator (defined below), and two instantiations of the coordination combinator (defined in section 3.2). It allows expressing all higher-order logic terms that can be constructed by recursively combining unary predicates (i.e. constants of type $Ind \to Bool$) with negation, conjunction and disjunction.

This lexicon can be used to parse and produce expressions where the words associated with the Boolean operators `and,or` are placed between the expressions associated with their arguments, and the word associated with `not` is placed before the expression associated with its argument. For example, an agent using this lexicon would produce expression 'claysup' to communicate the higher-order logic term $and(up)(li)$. As we will see in the following, the agents can construct lexicons which place the words associated with `and,not` and `or` in different positions[11].

$$sup \Rightarrow \texttt{up} : pr : 1.0 \tag{2}$$
$$inf \Rightarrow \texttt{do} : pr : 1.0 \tag{3}$$
$$izq \Rightarrow \texttt{le} : pr : 1.0 \tag{4}$$
$$der \Rightarrow \texttt{ri} : pr : 1.0 \tag{5}$$
$$cla \Rightarrow \texttt{li} : pr : 1.0 \tag{6}$$
$$osc \Rightarrow \texttt{da} : pr : 1.0 \tag{7}$$
$$no \Rightarrow \mathrm{Neg}_{Ind\to Bool}(not) : pr/pr : 0.86 \tag{8}$$
$$y \Rightarrow \mathrm{Coor}_{Ind\to Bool}(and) : pr\backslash pr/pr : 0.93 \tag{9}$$
$$o \Rightarrow \mathrm{Coor}_{Ind\to Bool}(or) : pr\backslash pr/pr : 0.75 \tag{10}$$

[11] Syntactic categories *pr/pr* or pr\pr are used to place the word associated with `not` before or after the expression associated with its argument. Categories *pr/pr/pr, pr\pr/pr* and *pr\pr\pr* place the words associated with `and,or` before, between or after the expressions associated with their arguments.

The *negation operator* Neg_σ, of type $(Bool \to Bool) \to \sigma \to \sigma$ for $\sigma \in BoolType$, is defined by
a. $\text{Neg}_{Bool}(\text{not})(\alpha) \stackrel{\text{def}}{=} \text{not}(\alpha)$
b. $\text{Neg}_{\gamma\to\tau}(\text{not})(\alpha) \stackrel{\text{def}}{=} \lambda x^\gamma.\text{Neg}_\tau(\text{not})(\alpha(x))$

This lexicon, however, does not allow the unambiguous communication of certain meanings. For example, it generates the same expression, *'noizqysup'*, to communicate the higher-order logic terms `not(and(up)(le))` and `and(up)(not(le))`, which are not logically equivalent. Natural language, on the other hand, provides means to mark the difference between both meanings: `not(and(up)(le))` can be expressed in English as *'the objects which are not both in an upper position and to the left'*, and `and(up)(not(le))` as *'the objects which are not to the left and which are in an upper position'*. English does so by introducing a word (*'both'*) which indicates that the conjunction following it forms a group. Therefore, the negation (`not`) applies to the entire conjunction, rather than just to the first element of such a conjunction. Thus, a refined version of the previous lexicon could use expression *'nogrupoizqysup'* to communicate term `not(and(up)(le))` and expression *'noizqysup'* to communicate term `and(up)(not(le))`, where *'grupo'* is a word that plays the same role as *'both'* or *'either'* in English, and which might have been invented by an agent during the simulation.

A similar problem comes up if we try to use the lexicon presented above to express term `or(and(le)(do))(up)`, which can be paraphrased in English as *'the objects which are in an upper position, or down and to the left'*. It generates expression *'supoinfyizq'*, which is ambiguous with respect to the scope of words *'y'* and *'o'*. For example, *'y'* could join expressions *'inf'* and *'izq'*, or expressions *'supoinf'* and *'izq'*. In English, a 'comma' is used to delimit the two main parts of the expression. In speech the disambiguating role of punctuation marks is played by pauses, and in this work we will treat pauses as words which indicate that the expression following them forms a group and, therefore, that the Boolean operators which form that group are within the scope of another operator. Thus, a refined version of the previous lexicon would generate expression *'supogrupoinfyizq'* to communicate term `or(and(le)(do))(up)`.

The linguistic system constructed by the agents in the experiments described in the present paper uses, therefore, three types of linguistic devices to indicate the relation between the word associated with a Boolean operator and the expressions associated with its arguments in an utterance: *syntactic categories, word order* and *function words*. The linguistic systems in [46,27,41] use only *syntactic categories* and *word order*. The agent-based models presented in [21,8] investigate the emergence and evolution of morphological linguistic systems, where meaning is expressed using compound word forms that have a lexical core plus different types of markers represented by affixes[12]. In morphological systems the semantic relations between words are expressed through grammatical agreement, instead of word ordering. Grammatical agreement means that certain grammatical features, such as tense-aspect or case, are shared among words which are semantically related. The agent-based models described in [52,19] study the emergence of *word-order-based phrase-structure*, which allows the definition of atomic units (containing a single word) and group units (consisting of several atomic and/or non-atomic units), thus introducing hierarchical structure. It is worth noting that the functional syntactic categories used in the present paper also generate hierarchical structure, because they allow concatenating a subexpression of a functional syntactic category (i.e. $A\backslash B$ or B/A) and a subexpression of its argument category (i.e. A) to form an expression of the result syntactic category (i.e. B).

[12] Affixes are groups of letters attached to words.

[42] also uses syntactic categories, word order and function words as linguistic devices, and it argues that the higher expressiveness of the resulting linguistic system, which allows the unambiguous communication of every propositional logic formula, compensates for the added difficulty of parsing and producing sentences with function words. However, the approach proposed in the present paper is more general than that of [42], because it uses higher-order logic and categorial grammar instead of propositional logic and definite clause grammars, and therefore it allows formalising not only the coordination of expressions denoting basic properties (i.e. constants of type $Ind \rightarrow Bool$ of higher-order logic), but also the coordination of expressions denoting higher-order logic terms of any Boolean type.

4 Language Game

The language game used in the experiments is played by two agents, a *speaker* and a *hearer*, randomly chosen from the population.

1. The speaker chooses a *meaning* (a higher-order logic *term*) from its conceptualisations of the subset of objects that constitutes the topic of the language game[13], *generates* a linguistic *expression* for that term using its lexicon or *invents* a new one, and communicates that expression to the hearer.
2. The hearer tries to *interpret* the expression communicated by the speaker. If it can *parse* it using its lexicon, it extracts a meaning; otherwise, the speaker communicates the meaning it had in mind to the hearer, and the hearer tries to *adopt* the lexical entries necessary to generate the expression used by the speaker to communicate that meaning.
3. A language game *succeeds* if the hearer can parse the expression communicated by the speaker and if its interpretation of that expression is *logically equivalent* to the meaning the speaker had in mind; otherwise, the language game *fails*. Depending on the *outcome* of the language game, the hearer expands its lexicon or *adapts* the scores of its lexical entries to be more successful in future language games.

4.1 Generation and Invention

At the early stages of a simulation run the agents cannot use their lexicons to generate expressions for most meanings, because they all begin with a common lexicon for basic properties, but no lexicon for Boolean operations. In order to let language get off the ground, the agents are allowed to invent new expressions for those meanings they cannot communicate using their lexicons. A new expression E for a higher-order logic term `F` of the form $\texttt{not}(\alpha)$ or $\otimes(\alpha)(\beta)$, where $\otimes$ is a dyadic Boolean operator (`and`,`or`), is invented as follows: a new word is invented for `F`'s main functor (`not`, $\otimes$), an expression is generated for each argument (α and/or β), and the two or three expressions produced are concatenated in random order[14]. New words are sequences of letters randomly chosen from a finite alphabet. In the experiments described in the present paper, we also assume that set of sequences of letters the agents can generate to invent new words is so large that homonymy never arises in the language constructed by the population.

[13] A conceptualisation of a subset of objects (i.e. a *meaning*) is a higher-order logic term that is true for all the objects in the subset and false for the rest of the objects in the speaker's and hearer's context.

[14] If `F` is a constant of type $Ind \rightarrow Bool$, i.e. a basic property, invention is not necessary, because an entry for `F` already exists in the common lexicon.

For example, if at the beginning of a simulation run an agent knows the lexicon for basic properties described in section 3.3 (which consists of lexical entries 2 to 7) and needs to express a meaning of the form `and(up)(li)`, it can invent a new word for the Boolean operator `and` (for example *plus*), use the words in his lexicon for `up` and `li` (i.e. *sup* and *cla*), and invent the expression *supclaplus* by concatenating the three words in random order.

Once an agent can generate an expression for a particular meaning using its lexicon, it does not invent new expressions for that meaning. Instead, it selects the expression with the highest score from the set of all the expressions it can generate for that meaning. The *score of an expression* generated using some lexical entries is computed multiplying the scores of those lexical entries.

The score of a lexical entry indicates the confidence an agent has that such a lexical entry could be understood by the rest of the agents in the population. Given that the scores of the lexical entries associated with basic properties are always equal[15] to 1.0, the product of the scores of the lexical entries used in the generation of an expression is, in fact, the product of the scores of each word associated with a Boolean operator occurring in such an expression. This product tries to capture the confidence an agent has that the rest of the agents in the population would be able to parse such an expression, on the one hand, and the probability that the rest of the agents in the population would be able to construct a partial parse tree of such an expression if they cannot parse it completely, on the other. This is so, because the agents apply simplification and repair operations to learn the meanings of unknown words as the play language games, and they need to construct such partial parse trees of unknown expressions for this purpose. In order to assess the probability that an agent would be able to construct a partial parse tree for a sentence, we take into account the following facts: 1) every time the agent finds an unknown word for a Boolean operator in the sentence, the agent must carry out a search process to determine what parse subtree might be associated with it; 2) the search process for each unknown word takes place within the search processes generated by the unknown words preceding it in the sentence. Therefore, the number of options the agent must consider in order to find a partial parse tree for a sentence gets multiplied by the branching factor of the seach space associated with the unknown word every time the agent finds a new instance of an unknown word in the sentence. This multiplicative effect of each instance of an unknown word in the difficulty of constructing a partial parse tree for a given sentence is the reason why we compute the score of a sentence generated using some lexical entries multiplying the scores of such lexical entries, and the score of a meaning obtained by parsing a sentence using some lexical entries (see section 4.2) multiplying the scores of such lexical entries[16].

The agents' abilities to *parse* and *generate* expressions by using their categorial lexicons and the phrase-structure application schemes described in section 3.1 have been implemented in Prolog[17] [15, 14]. In particular, a *chart parser* [24, 25, 5, 22] and an *expression generator* for the categorial lexicons used in the experiments have been designed.

[15] The reason for this is that the lexical entries for basic properties are part of the common lexicon of basic concepts initially shared by all the agents in the population.

[16] It should be noted that although the choice of words for expressing different Boolean operators is independent of each other, the choice of a lexical entry for expressing a particular operator in a given sentence is always the same for all the occurrences of such an operator in the higher-order term expressed by such a sentence. Likewise, the choice of a lexical entry for parsing a particular word in a given sentence is always consistent for all the occurrences of such word in the sentence.

[17] In particular, the Ciao Prolog System [10], available from www.clip.dia.fi.upm.es, has been used.

4.2 Interpretation and Adoption

In a language game the hearer tries to interpret (i.e. parse) the expression E communicated by the speaker using its own lexicon. However, at the early stages of a simulation run the agents cannot parse most of the expressions communicated by the speakers, because they all begin with a common lexicon for basic properties, but no lexicon for Boolean operations. When this happens, the speaker communicates the higher-order logic term `F` it had in mind to the hearer, and the hearer tries to adopt the lexical entries necessary to generate the expression E used by the speaker to communicate `F`. That is, the hearer tries to construct some lexical entries that allow interpreting expression E as meaning `F`; and it adopts such lexical entries, if it can construct them.

Once an agent can parse an expression using its lexicon, it selects the meaning with the highest score from the set of all the meanings it can obtain by parsing that expression as its interpretation `I` of the expression communicated by the speaker. The *score of a meaning* obtained by parsing an expression using some lexical entries is computed multiplying the scores of those lexical entries[18].

Although in real life people do not communicate the meaning they have in mind to their interlocutors, as the speaker does in a language game when the hearer cannot parse the utterance, they usually provide some form of contextual or gestural feedback. Actually, some capacity for *intention-reading* seems to be necessary for language acquisition [54]. Studies on syntax emergence and evolution such as [8,21,27,7,41,42,52,19] also use language games in which the meaning intended by the speaker is communicated to the hearer. The difficulty of studying language evolution leads researchers to focus on particular aspects of it, assuming results from research works addressing issues such as lexicon and concept formation [44,31,39,51], or speech evolution [16,59].

4.3 Simplification through Abstraction

Invention and adoption allow the agents to construct and learn associations between expressions and meanings. For example, when an agent acting as speaker cannot express a meaning `F` using its lexicon, it invents an expression E to communicate `F`, and constructs an association of the form $E \Rightarrow \texttt{F}:C$, where C is the syntactic category[19] of expression E. Similarly, an agent acting as hearer who cannot parse the expression E communicated by the speaker using its lexicon also constructs an association $E \Rightarrow \texttt{F}:C$, where E and `F` are the expression and the meaning used by the speaker, respectively. From these associations the agents construct new lexical entries that they incorporate to their lexicons and use in succeeding language games to generate and interpret other sentences.

However, the agents do not build lexical entries directly from the associations $E \Rightarrow \texttt{F}:C$ they construct during the invention or adoption steps of a language game. They apply two *simplification rules* to such associations, in order to generalise them and remove from them those parts they can generate using lexical entries which already belong to their lexicons.

Let us illustrate how these simplification rules work with an example. Suppose an agent constructs association $\mathit{supyizq} \Rightarrow \texttt{and}(\texttt{le})(\texttt{up}) : \mathit{pr}$ during the invention or adoption steps of a language game. Instead of adding this association to its lexicon as a new lexical entry, it

[18] The same arguments as those used in section 4.1 can be used to justify the method used to compute the score of a particular meaning obtained by parsing a given expression.

[19] In the experiments, the syntactic category C of the expressions invented by the agents is pr.

first tries to simplify it using its previous knowledge of language, i.e. the lexical entries in its lexicon. For example, subexpression '*sup*' of '*supyizq*' can be generated by lexical entry 2 (i.e. $sup \Rightarrow \texttt{up} : pr : 1.0$). Therefore, association $supyizq \Rightarrow \texttt{and}(\texttt{le})(\texttt{up}) : pr$ can be simplified by: 1) eliminating subexpression '*sup*' from its left-hand side; 2) replacing constant `up` in the semantic part of the right-hand side with a variable of the same type, and abstracting over it; and 3) transforming the syntactic category on the right-hand side (i.e. *pr*) into a functor category $pr\backslash pr$ whose argument is the syntactic category of the subexpression removed from the left-hand side (i.e. the syntactic category of '*sup*'), and whose direction (i.e. forward or backward) is determined by the position the subexpression removed (i.e. '*sup*') occupied in the original expression of the left-hand side (i.e. '*supyizq*'). The result is the following association, which is both simpler and more general:

$$\text{yizq} \Rightarrow \lambda\beta_{Ind \rightarrow Bool}\ \texttt{and}(\texttt{le})(\beta) : pr\backslash pr \tag{11}$$

The particular type of simplification illustrated in the previous example, which transforms association $supyizq \Rightarrow \texttt{and}(\texttt{le})(\texttt{up}) : pr$ into association $yizq \Rightarrow \lambda\beta_{Ind \rightarrow Bool}\ \texttt{and}(\texttt{le})(\beta) : pr\backslash pr$ by using lexical entry $sup \Rightarrow \texttt{up} : pr : 1.0$, gives rise to the first simplification rule introduced in this paper (defined as follows).

Rule 1 *(Backward Simplification)* An agent can simplify an association of the form $e_1 \Rightarrow \alpha : c_1$ into association $e_3 \Rightarrow \lambda x_\tau \alpha' : c_2\backslash c_1$, if there is a proper subterm γ of α such that
1. expression e_γ of syntactic category c_2 can be generated to communicate the higher-order logic term γ using the agent's lexicon;
2. there exists a non-empty expression e_3 such that $e_\gamma \cdot e_3$ (i.e. the concatenation of expressions e_γ and e_3) is e_1; and
3. α' is the result of replacing term γ with variable x_τ in α, where τ is the type of γ and x_τ is a variable of type τ.

Similarly, using lexical entry 4 (i.e. $izq \Rightarrow \text{le} : pr : 1.0$), an agent can eliminate subexpression '*izq*' from the left-hand side of association 11, replace constant `le` with variable α of type $Ind \rightarrow Bool$, abstract over α, and add a forward argument to the syntactic category of '*yizq*', obtaining

$$\text{y} \Rightarrow \lambda\alpha\lambda\beta\ \text{and}(\alpha)(\beta) : (pr\backslash pr)/pr$$

an instance of $y \Rightarrow \mathbf{Coor}_\sigma(\mathbf{and}) : A\backslash A/A$ with $\sigma = Ind \rightarrow Bool$ (see section 3.2).

The type of simplification illustrated in the above example, which transforms association $yizq \Rightarrow \lambda\beta_{Ind \rightarrow Bool}\ \texttt{and}(\texttt{le})(\beta) : pr\backslash pr$ into association $y \Rightarrow \lambda\alpha\lambda\beta\ \text{and}(\alpha)(\beta) : (pr\backslash pr)/pr$ by using lexical entry $izq \Rightarrow \text{le} : pr : 1.0$, gives rise to the second simplification rule introduced in this paper (defined as follows).

Rule 2 *(Forward Simplification)* An agent can simplify an association of the form $e_1 \Rightarrow \alpha : c_1$ into association $e_3 \Rightarrow \lambda x_\tau \alpha' : c_1/c_2$, if there is a proper subterm γ of α such that
1. expression e_γ of syntactic category c_2 can be generated to communicate the higher-order logic term γ using the agent's lexicon;
2. there exists a non-empty expression e_3 such that $e_3 \cdot e_\gamma$ (i.e. the concatenation of expressions e_3 and e_γ) is e_1; and
3. α' is the result of replacing term γ with variable x_τ in α, where τ is the type of γ and x_τ is a variable of type τ.

Thus, when an agent constructs an association $E \Rightarrow F : C$ as a result of an invention or adoption step in a language game, and the meaning F of such an association is either a

conjunction or a disjunction of basic properties, or the negation of a basic property[20], the agent applies simplification rules 1 and 2 to such an association until it cannot be further simplified. During the simplification process *backward simplification* has precedence over *forward simplification*. Finally, the association resulting from this simplification process is added to the agent's lexicon as a new lexical entry, so that the agent can use it in succeeding language games to parse/generate other sentences. The initial score μ of new lexical entries constructed in this way is set to 0.5 in the experiments described in this paper.

In [41,42], simplification is performed by applying two *induction operations*, called *simplification* and *chunk*, to the grammar rules constructed by an agent. These operations adapt the induction rules proposed in [27,56] to the type of grammar rules used in [41, 42]. The *forward* and *backward simplification rules* defined in the present paper differ from those used in [41,42] in applying *deduction* to infer the meaning and syntactic categories of unknown subexpressions (i.e. expressions which cannot be generated or parsed by known grammar rules), instead of inducing new grammar rules from previous ones and inventing syntactic categories for unknown subexpressions. Furthermore, *forward* and *backward simplification* eliminate the need to introduce the induction rules *chunk I* and *chunk II* proposed in [27], since the application of a *forward (or backward) simplification* rule achieves in a single step the combined effect of applying the rules *simplification* and *chunk* in [41,42].

The rule of *simplification* in [41,42] achieves generalisation by replacing subterms with new variables at the semantic level and known subexpressions with non-terminals of their syntactic categories at the syntactic level. However, it does not allow inferring the syntactic categories of unknown subexpressions, as forward (or backward) simplification rules do. The two rules *chunk I* and *chunk II* in [41,42] are therefore needed to allow generalisation of grammar rules involving unknown subexpressions. They achieve such generalization by inventing new syntactic categories for unknown subexpressions, and using non-terminals of such categories in the generalised versions of grammar rules involving unknown subexpressions. The *forward (backward) simplification* rules proposed in the present paper, instead of inventing new syntactic categories for unknown subexpressions, deduce the meanings and syntactic categories of such subexpressions from the meanings and syntactic categories of the subexpressions surrounding them in a given utterance, in accordance with the Lambek Calculus' algebraic interpretation of operators $/$ and $\backslash$ (see section 3.2); and construct the meanings and syntactic categories of such unknown subexpressions by applying functional abstraction at both the semantic level and the syntactic level. In particular, *forward (or backward) simplification* generalises an association *at the semantic level* by replacing a term γ with a variable x_τ of the same type and applying λ-abstraction to x_τ, and it generalises the same association *at the syntactic level* by replacing syntactic category c_1 with functor category c_1/c_2 (or $c_2\backslash c_1$), which is more general than c_1 because it depends on category c_2.

The repair meta-operators *build-hierarchy* and *build-or-extend-group* proposed in [52] are different from the *simplification rules* defined in the present paper, those used in [41,42], or the repair operators described in the next subsection. They build group-units by combining existing units, or extend the set of constituents of an existing group-unit with another unit. This is automatically done to some extent by the categorial grammar phrase-structure schemes implemented in the parsing and production algorithms used in the present paper, and by the parsing and production algorithm of Definite Clause Grammars built in Prolog and used in [41,42]. But the meta-operators *build-hierarchy* and *build-or-extend-group* also

[20] In the experiments, the agents do not try to simplify associations between expressions and more complex meanings, except in the case of learning the expressions used by other agents to refer to the grouping operator `id`, which we discuss in the next subsection in the context of *repair operations*.

establish co-reference relations between the variables occurring in the semantic and syntactic parts of the component units of a group-unit. This is necessary in [52], because the meanings the agents communicate to each other consist of a conjunction of atomic formulas containing only n-ary predicate symbols and free variables which may occur in more than one atomic formula, and the role of syntax is to specify the co-reference (i.e. equality) relations of the arguments (i.e. variables) of the different atomic formulas in a given meaning. The simplification rules proposed in the present paper do not deal with co-reference relations between variables, because there are no free variables in the higher-order logic terms the agents use to represent the meanings they communicate to each other, nor in the propositional logic formulas used to represent the meanings of expressions in [41,42].

The *build-hierarchy* meta-operator in [52] also decides which of the arguments involved in a group-unit is going to be the referent of the group-unit, and determines on that basis the syntactic and the semantic categories of the group-unit. In the experiments described in the present paper, the agents also build group-units when they parse or produce an utterance, because the phrase-structure application schemes build higher order logic terms of the form $\alpha(\beta)$ to express the meaning of expressions constructed by concatenating two subexpressions whose meanings are α and β respectively, and they also assign syntactic categories to the expressions resulting from concatenating two subexpressions, using the syntactic categories of the subexpressions. However, the meaning and syntactic category of a group-unit constructed in this manner are determined by the meanings and syntactic categories of its constituents in the present paper. So if the agents need to explore the space of possible compositions of a given pair of subexpressions, they must assign different meanings and syntactic categories to such subexpressions, and the phrase structure application schemes do the rest of the work automatically. In [41,42], the meanings and syntactic categories of group-units are also determined by the meanings and syntactic categories of its constituents, but in this case the Definite Clause Grammar rules built by the agents during the simulations may specify different ways of constructing the meanings and syntactic categories of group-units from those of their constituents.

Finally, the meta-operator *coercion* in [52] plays a similar role to rule *chunk II* in [41, 42], which is also subsumed by the *forward* and *backward simplification* rules defined in the present paper. The difference is that the meta-operator *coercion* in [52] adds a lexical category to a word unit, and the simplification rules proposed in the present paper add a new lexical entry for that word to the agent's lexicon, which associates the word with a new syntactic category in [41,42], and with an inferred syntactic category in the present paper.

4.4 Repair

In the first step of a language game, the speaker tries to generate an expression to communicate a higher-order logic term `F` that allows discriminating the subset of objects that constitutes the topic of the language game from the rest of the objects in the context. As mentioned earlier, if the speaker can generate an expression for term `F` using its lexicon, it selects the expression E with the highest score from the set of all expressions it can generate for `F`. However, if it realises that expression E is ambiguous, for example sentence *'noizqy-sup'*, generated using lexical entries 2, 4, 8 and 9, can be parsed as $\texttt{not}(\texttt{and}(\texttt{up})(\texttt{le}))$ or as $\texttt{and}(\texttt{up})(\texttt{not}(\texttt{le}))$, it applies a *repair operation* to lexical entry 9, replacing it with 12, which introduces a new syntactic category gr_{pr} for expressions constructed by applying a coordination operator to expressions of category pr.

$$y \Rightarrow \texttt{Coor}_{\texttt{Ind}\rightarrow\texttt{Bool}}(\texttt{and}) : pr\backslash gr_{pr}/pr : 0.93 \tag{12}$$

Apparently this contradicts the principle that any pair of categories (e.g. *pr*) can coordinate to produce a result of the same category (e.g. *pr* instead of gr_{pr}). But it only prevents the direct application of a negation or coordination operator to the expression resulting from coordinating a pair of categories, because a new word playing the same role as *'both'* or *'either'* in the examples of section 3.3 is also introduced in order to enable the transformation of the expression resulting from coordinating a pair of expressions into an expression of the same syntactic category as the coordinate expressions, and therefore the application of negation or coordination operators. In section 3.3, the new word introduced by the agent was *'grupo',* and we will also use this word in the examples of this section. But it is important to note that in the experiments the agents invent different words, which play the same role as *'grupo'*.

Continuing with the example, if the speaker's lexicon does not contain a lexical entry for a word that transforms expressions of syntactic category gr_{pr} into expressions of syntactic category *pr*, the speaker, in addition to replacing lexical entry 9 with lexical entry 12, adds lexical entry 13 to its lexicon. Constant `id`, used in the semantic part of 13, represents the identity function, which maps every higher-order logic term into itself. Therefore, from the semantic point of view, we could say that `id` plays the same role as placing a pair of parentheses around the term to which `id` is applied.

$$\text{grupo} \Rightarrow \texttt{id} : pr/gr_{pr} : 0.5 \tag{13}$$

Note that it is possible that the speaker had created already a lexical entry similar to 13 in a previous language game. In this case this step is skipped. For example, if the speaker tries to repair lexical entry 10 after having applied a repair operation to lexical entry 9, it will only replace lexical entry 10 with 14, where gr_{pr} is the syntactic category of the expression resulting from coordinating two expressions of category *pr*, which was introduced when lexical entry 12 was created. That is, it does not create a lexical entry that can be used to transform an expression of syntactic category gr_{pr} into an expression of syntactic category *pr*, because a lexical entry that can be used for that purpose (i.e. 13) already exists in its lexicon[21].

$$o \Rightarrow \text{Coor}_{Ind \rightarrow Bool}(or) : pr\backslash gr_{pr}/pr : 0.75 \tag{14}$$

It is easy to check that the refined lexicon (consisting of entries 2 to 8, 12, 13 and 14) generates unambiguous expressions for the examples considered in section 3.3, i.e. it produces expression *'nogrupoizqysup'* to communicate term `not(id(and(up)(le)))`, expression *'noizqysup'* for term `and(up)(not(le))`, and expression *'supogrupoizqyinf'* for term `or(id(and(do)(le)))(up)`. The score of lexical entries created by repair operations (e.g. lexical entry 13) is initially set to 0.5. But the score of lexical entries which are modifications of lexical entries which already belonged to the agent's lexicon is the score of the original lexical entry (e.g. lexical entry 14 modifies lexical entry 10).

Thus, repair operations may involve two steps: 1) the replacement of a lexical entry for a coordination operator (e.g. lexical entry 9) by a repaired lexical entry for the same coordination operator (e.g. lexical entry 12); and 2) the addition of a lexical entry for a new

[21] On the other hand, if a repair operation had been applied to a sentence that contained the logical operator 'or' before any repair operation had been applied to a sentence that contained the logical operator 'and', lexical entry 10 would have been replaced with lexical entry 14 in step 1, and lexical entry 13 would have been added to the agent's lexicon in step 2. Later on, if the agent applies a repair operation to a sentence containing 'and', it would replace lexical entry 9 with lexical entry 12 and step 2 would not be applied.

word whose meaning is `id` and which plays the same role as *'both'* or *'either'* in the English sentences used as examples in section 3.3.

In step 1, a lexical entry for a coordination operator (`and, or`) is replaced with a lexical entry for the same coordination operator whose syntactic category has the same functional structure as that of the original entry, except that it produces a result of syntactic category gr_c instead of c. Syntactic category gr_c is constructed by the agent[22], unless there is a lexical entry in its lexicon that already uses this category. There are three possible cases depending on the position in the sentence of the word associated with the coordination operator:

- *Infix:* If the word associated with the coordination operator is placed *between* the expressions associated with its arguments, the syntactic category of the original lexical entry is $c\backslash c/c$, and the syntactic category of the repaired lexical entry is $c\backslash gr_c/c$, where c is the syntactic category of the coordinate expressions.
- *Prefix:* If the word associated with the coordination operator is placed *before* the expressions associated with its arguments, the syntactic category of the original lexical entry is $c/c/c$, and the syntactic category of the repaired lexical entry is $gr_c/c/c$.
- *Suffix:* If the word associated with the coordination operator is placed *after* the expressions associated with its arguments, the syntactic category of the original lexical entry is $c\backslash c\backslash c$, and the syntactic category of the repaired lexical entry is $c\backslash c\backslash gr_c$

In step 2, a lexical entry that introduces a new word which plays the same role as *'both'* or *'either'* in English is created (e.g. lexical entry 13), if the agent's lexicon does not have another word that can be used for that purpose. We have already explained that the meaning of the new word is `id`, the identity function, and that its syntactic category must be a functional category that transforms an expression of syntactic category gr_c into an expression of syntactic category c, where c is the syntactic category of the coordinate expressions. The direction of such a functional category (forward or backward) depends on the direction of the functional category of the lexical entry for `not` that was used in the generation of the ambiguous expression that triggered the application of the repair operation.

- *Prefix:* If the syntactic category of the lexical entry used for `not` is c/c (i.e. the word associated with the negation operator is placed *before* the expression associated with its argument), the syntactic category of the new lexical entry should be c/gr_c.
- *Suffix:* If the syntactic category of the lexical entry used for `not` is $c\backslash c$ (i.e. the word associated with the negation operator is placed *after* the expression associated with its argument), the syntactic category of the new lexical entry should be $gr_c\backslash c$.

Repair operations are therefore triggered by the generation of ambiguous expressions to communicate meanings of the form $\texttt{not}(\texttt{and}(\alpha)(\beta))$ or $\texttt{not}(\texttt{or}(\alpha)(\beta))$[23]. It should be noted, however, that once an agent has applied a repair operation to the lexical entry of a coordination operator (`and,or`), it cannot use the repaired lexical entry to generate expressions for higher-logic terms of the form $\texttt{not}(\texttt{and}(\alpha)(\beta))$, $\texttt{not}(\texttt{or}(\alpha)(\beta))$, $\texttt{or}(\texttt{and}(\texttt{do})(\texttt{le}))(\texttt{up})$ or $\texttt{and}(\texttt{up})(\texttt{or}(\texttt{do})(\texttt{le}))$. It must *reconceptualise* these terms in such a way that they use the grouping operator `id`, which is required to apply negation or coordination operators to expressions resulting from coordinating a pair of constituents of the same Boolean type. Thus, instead of using the previous higher-order logic terms as meanings, the agent must reconceptualise such meanings as $\texttt{not}(\texttt{id}(\texttt{and}(\alpha)(\beta)))$, $\texttt{not}(\texttt{id}(\texttt{or}(\alpha)(\beta)))$, $\texttt{or}(\texttt{id}(\texttt{and}(\texttt{do})(\texttt{le})))$

[22] We describe repair operations for arbitrary syntactic categories c, because they are applicable to lexical entries for coordinators of expressions of any Boolean type. However, in the experiments, c is always pr.

[23] They could have been triggered by the generation of ambiguous expressions for other types of meanings, e.g. $\texttt{and}(\alpha)(\texttt{not}(\beta))$ or $\texttt{and}(\alpha)(\texttt{or}(\beta)(\gamma))$. But in the present experiments this is not the case.

(up) and and(up)id(or(do)(le))) respectively before trying to communicate them to other agents.

In the previous section we saw how simplification operations applied to associations constructed during the invention or adoption steps of a language game allow the agents to expand their lexicons by adding new lexical entries for coordination and negation operators. These entries correspond to words and syntactic categories for coordination (and,or) and negation (not) operators that a given agent may invent when it acts as a speaker in a language game or that it learns from other agents when it acts as a hearer. However, we have not explained yet how the agents learn the expressions used by other agents to refer to the grouping operator id. This is done as follows.

Suppose an agent acting as speaker tries to communicate to the hearer a higher-order logic term F of the form not(id($\otimes(\alpha)(\beta)$))), where $\otimes$ is a dyadic coordination operator (and, or), and α and β are terms of the same Boolean type. If the speaker can generate an unambiguous expression E for F, it communicates E to the hearer. If the hearer cannot parse E, but it can understand the expressions used by the speaker for not and $\otimes$, and it has applied a repair operation to the lexical entry for $\otimes$ in past language games, then the hearer constructs an association of the form $E' \Rightarrow \texttt{id}(\otimes(\alpha)(\beta)) : c$ as follows.

1. If the word e_{not} used by the speaker to communicate operator not is associated with a lexical entry of the form $e_{not} \Rightarrow \texttt{not} : c/c : s$, then E' will be the result of removing prefix e_{not} from E, i.e. the concatenation $e_{not} \cdot E'$ will be equal to E.
2. If the word e_{not} used by the speaker to communicate operator not is associated with a lexical entry of the form $e_{not} \Rightarrow \texttt{not} : c\backslash c : s$, then E' will be the result of removing suffix e_{not} from E, i.e. the concatenation $E' \cdot e_{not}$ will be equal to E,

Next, the hearer applies *simplification* to association $E' \Rightarrow \texttt{id}(\otimes(\alpha)(\beta)) : c$, using subterm $\otimes(\alpha)(\beta)$ of $\texttt{id}(\otimes(\alpha)(\beta))$ and subexpression $e_{\otimes(\alpha)(\beta)}$ of E', where $e_{\otimes(\alpha)(\beta)}$ is the expression generated by the hearer to communicate $\otimes(\alpha)(\beta)$ using the same lexical entries for $\otimes, \alpha$ and β that were used by the speaker. In particular, the result of applying simplification to association $E' \Rightarrow \texttt{id}(\otimes(\alpha)(\beta)) : c$, using subterm $\otimes(\alpha)(\beta)$ and subexpression $e_{\otimes(\alpha)(\beta)}$, may be of one the following forms[24]

1. $e_{\texttt{id}} \Rightarrow \texttt{id} : c/gr_c$, if the lexical entry used by the speaker for id was a prefix; or
2. $e_{\texttt{id}} \Rightarrow \texttt{id} : gr_c\backslash c$, if the lexical entry used by the speaker for id was a suffix.

The main difference is that in the former case *forward simplification* is applied, because $e_{\otimes(\alpha)(\beta)}$ is a *suffix* of E', i.e. concatenation $e_{id} \cdot e_{\otimes(\alpha)(\beta)}$ is equal to E'; whereas in the latter *backward simplification* should be used, because $e_{\otimes(\alpha)(\beta)}$ is a *prefix* of E'. It should be noted that simplification of a term α using a subterm γ_1 takes precedence over simplification of α using another subterm γ_2, if γ_2 is a proper subterm of γ_1. This explains why $E' \Rightarrow \texttt{id}(\otimes(\alpha)(\beta)) : c$ is simplified using subterm $\otimes(\alpha)(\beta)$, instead of subterms α or β.

And finally, the hearer adds the association resulting from the simplification process described above to its lexicon as a new lexical entry.

Although both simplification operations (section 4.3) and repair operations (section 4.4) modify an agent's lexicon, there are important differences between them. The former are applied when a new association is created in the invention or adoption steps of a language game. They try to generalise the association so that it can be used to express and parse more

[24] We use again an arbitrary syntactic category c to describe the adoption of lexical entries constructed during the application of repair operations, because the mechanisms proposed are applicable to lexical entries for coordinators of expressions of any Boolean type. However, in the experiments, c is always pr.

meanings. Repair operations, on the other hand, specialise lexical entries which are too general and can therefore generate ambiguous expressions. Simplification rules operationalise the *pattern-finding* ability children develop according to *the usage-based theory of language acquisition* [54]. Whereas repair operations implement cognitive processes associated with a particular type of grammar invention and overgeneralisation examples reported in studies of children language acquisition [55].

4.5 Adaptation

Coordination of the agents' preferences for lexical entries is necessary, because different agents may invent different words to refer to the same Boolean operator, and they may place these words in different positions (prefix, infix or suffix) with respect to the positions of the expressions associated with the arguments of such a Boolean operator. Although [46, 27, 19] also study the acquisition of word-order based syntax, they do not address the issue of coordination of the agents' preferences for lexical entries, because the populations in their experiments consist only of two agents, which necessarily share the same history of linguistic interactions. In the experiments discussed in the present paper *coordination* is achieved through a process of self-organisation of the agents' linguistic interactions, which takes place when these agents update the scores of their lexical entries in order to adapt their preferences for lexical entries to those they observe are used more often by other agents.

The agents update the scores of their lexical entries at the last step of a language game, when the speaker communicates the meaning it had in mind to the hearer, and only in the case in which the speaker can generate at least one expression for the meaning it is trying to communicate using its lexicon and the hearer can parse the expression communicated by the speaker. If an agent acting as speaker can generate several expressions to communicate a given meaning, it chooses the expression with the highest score, and temporarily stores the rest of the expressions in a set called *competing expressions*; similarly, if an agent acting as hearer can obtain several meanings by parsing an expression, it selects the meaning with the highest score and stores the rest in a set called *competing meanings*. In a language game only the agent playing the role of hearer updates the scores of its lexical entries. However, as all the agents in the population play both the role of speaker and that of hearer in different language games, all of them have ample opportunity to update the scores of their lexical entries during a simulation.

If the meaning interpreted by the hearer is logically equivalent to the meaning the speaker had in mind, the language game succeeds, and the hearer adjusts the scores of its lexical entries both at the level of interpretation and at that of generation: 1) It increases the scores of the lexical entries it used for obtaining the meaning the speaker had in mind and decreases the scores of the lexical entries it used for obtaining competing meanings. 2) It tries to simulate what it would have said if it had been in the speaker's place, i.e. it tries to express the meaning the speaker had in mind using its own lexicon; and it increases the scores of the lexical entries that generate the expression used by the speaker, and decreases the scores of the lexical entries that generate competing expressions.

If the meaning interpreted by the hearer is not logically equivalent to the meaning the speaker had in mind, the language game fails, and the hearer decreases the scores of the lexical entries it used for obtaining its interpretation of the expression communicated by the speaker.

The scores of lexical entries are updated by using the following formulas. The original score s of a lexical entry is replaced with the result of evaluating expression 15 if the score

is *increased*, and with the result of evaluating expression 16 if the score is *decreased*.

$$minimum(1,\ s+\delta_h^+) \tag{15}$$

$$maximum(0,\ s-\delta_h^-) \tag{16}$$

In our experiments, the following values for parameters δ_h^+, δ_h^- and μ have been used[25]: $\delta_h^+ = 0.1, \delta_h^- = 0.1$ and $\mu = 0.5$. This choice is based on the research described in [17], which proposes a formal framework for studying the evolution of conventions in multi-agent systems and a general method for analysing convention problems in multi-agent systems; and applies such a method to the study of agent-based models of the emergence and evolution of language, focusing on the *naming game* [45] and specifically on several strategies proposed in the literature for updating the scores of associations between words and meanings.

An additional feature of the adaptation mechanism used in the experiments described in the present paper is that if the score of a lexical entry becomes zero after a language game, such a lexical entry is deleted from the agent's lexicon.

In [17] the dynamics of a multi-agent system consisting of a population of autonomous agents playing the naming game is described by means of a system of stochastic equations, and its evolution is investigated by studying the properties of a deterministic system which approximates the average dynamics of the stochastic system. In particular, in this framework an agent design is said to solve a convention problem if: (1) the deterministic dynamical multi-agent system it induces converges to a state of agreement for almost all initial conditions, i.e. the set of initial conditions for which no agreement is reached has zero measure; and (2) agreement is reached in a reasonable amount of time for any population, i.e. the time required to reach it does not grow too fast as a function of the size of the population.

With respect to the question whether an agent solves a convention problem, the important properties of such a deterministic dynamical system are: (i) if all the states in which the agents agree on a convention are stable stationary states of the deterministic system; and (2) if the rest of the stationary states of the deterministic system are unstable. [17] argues that such properties can be investigated by studying the behaviour of the so-called agent's *response function*, because the stationary states of the deterministic system correspond to the fixed points of the response function, and the stability of such stationary states can be also determined by analysing the behaviour of such a function.

Specifically, for a population of autonomous agents playing the naming game, where the agents do not generate homonyms, update the scores of their lexical entries only when they play the role of hearer, and delete those lexical entries whose scores become zero after a language game, [17] shows that the agent's response function is well defined, and that if $\delta_h^+ = 0.1, \delta_h^- = 0.1$ and $\mu = 0.5$, then such a response function is sufficiently adaptive and amplifying. Intuitively, this means that the population will always succeed in establishing a common language in a reasonable amount of time, because if the response function is sufficiently *adaptive* then the equilibrium states are reached fast enough, and if the response function is *amplifying* then the population will never get stuck in a suboptimal behaviour[26].

The formal framework and results described in [17] are applicable to the convention problem studied in the present paper, because the language game the agents play in our

[25] μ is the initial score agents assign to the lexical entries they create during simplification or repair.

[26] Adaptation captures the speed at which an agent can adapt to a new situation, which determines the time it takes for a population to agree on a common language. Amplification relates to the extent to which an agent is able to escape from a behaviour which represents a suboptimal fixed point of its response function. This is achieved by amplifying small deviations from suboptimal equilibrium states, thus making them unstable.

experiments is an instance of the naming game, and the agents in the populations we use: (1) do not generate homonyms; (2) update the scores of their lexical entries only when they play the role of hearer; and (3) delete those lexical entries whose scores become zero after a language game. In the language game studied in our experiments, the agents should agree on a lexicon for naming three Boolean operators `and,or,not` and one grouping operator `id`. However, the linguistic expression of such operators not only requires specifying the word (or name) that should be assigned to each operator, but also the position where such word should be placed with respect to the expressions associated with its arguments (i.e. prefix, infix or suffix). Therefore, a "name" in the naming game played by the agents in our experiments consists of two parts: (1) the word which is associated with an operator; and (2) a specification of whether such a word should be placed before, between or after the expressions associated with the arguments of such an operator in the sentence.

5 Experiments

The agent-based model proposed in this paper has been implemented in Prolog, and tested by conducting a series of experiments which study both the emergence of a language system for Boolean coordination and its transmission across generations.

The first goal of the experiments we have carried out is to show that the conceptual and linguistic systems proposed, as well as the simplification and repair operators defined in previous sections can be operationalised, and that they allow a single agent to effectively construct and learn a language system for Boolean coordination. The second goal is to show that such simplification and repair operators, coupled with the adaptation mechanisms described in section 4.5, enable a population of autonomous agents to construct a shared language system for Boolean coordination and to transmit it to succeeding generations.

In order to provide empirical evidence that a language system for Boolean coordination actually emerges in a population of autonomous agents, we have chosen a score update strategy (i.e. *only hearer*), and some parameter values (i.e. $\delta_h^+ = 0.1, \delta_h^- = 0.1$ and $\mu = 0.5$) for which the population will always succeed in establishing a common language in a reasonable amount of time according to [17]. A detailed discussion of the effects of using different strategies for updating the scores of lexical entries (e.g. *speaker and hearer* versus *only hearer*), choosing different values for the initial score of lexical entries (μ) and update parameters (δ_h^+, δ_h^-), or allowing homonymy to be introduced in the language constructed by the population falls outside the scope of this paper, but it can be found in research works focusing on the study of the emergence and evolution of conventions in multi-agent systems such as [28], [1] and [17].

In all the experiments described in this section, the agents start with a common lexicon for 600 basic properties[27]. Then they play a series of language games about higher-order logic terms which recursively combine basic properties using negation, conjunction and disjunction. Such higher-order logic terms, which are randomly generated, may have between one and five nested Boolean operators (`and,or,not`). The depth of their syntactic trees ranges from two to six. For example, term `[[and,ri],le]`, which represents propositional formula *right* $\wedge$ *left*, is an example of an easy term with syntactic tree of depth two, which contains a single Boolean operator. On the other hand, term `[[or,[[and,li],up]], [not,[[or,li],up]]]`, which represents formula *light* $\leftrightarrow$ *upper*, is an example of a term

[27] Basic properties are propositional constants representing properties such as being light, dark, in an upper or lower position, or on the right or left side of the whiteboard. [21,8,19] also use simulations with software agents and initialise the agents with a common vocabulary for basic concepts.

of medium difficulty with syntactic tree of depth four and four Boolean operators. In order to allow the agents to have enough opportunity to learn the language constructed by others, 45% of the terms generated during a simulation contain a single Boolean operator, 20% have two Boolean operators and syntactic trees of depth three, 18% have three Boolean operators and syntactic trees of depth three or four, 12% have four operators, and 5% have five Boolean operators and syntactic trees of depth five or six.

We monitor the evolution over time of four measures, in order to observe the evolution of the population's global performance in the experiments studying the emergence and evolution of a language system for Boolean coordination. Time in this context should be understood as the number of games played by the population. Thus, the value of variable t at a given moment in a simulation is the number of games played so far by the population.

- *Communicative success* $S(t)$ is the average of successful language games in a sliding window of consecutive language games containing the games in the interval $[t-w+1,t]$ if $w < t$, or the games in the interval $[1,t]$ if $t \leq w$.
- *Lexical variability* $V(t)$ is the average of language games in a sliding window of consecutive language games containing the games in the interval $[t-w+1,t]$ if $w < t$, or the games in the interval $[1,t]$ if $t \leq w$, such that either: (1) the hearer does not understand correctly the expression communicated by the speaker, or (2) the hearer does understand correctly the expression communicated by the speaker, but the hearer would use an expression different from the expression used by the speaker to communicate the meaning the speaker had in mind. This measure captures the dissimilarity between the agents' internal languages (i.e. the sets of preferred lexical entries of the form *expression* $\Rightarrow$ *meaning: syntactic_category: score* stored by each individual agent). In previous papers [41] and [42], we monitored the evolution of *coherence* $C(t)$ rather than *lexical variability* $V(t)$. The relation between both measures is $V(t) = 1 - C(t)$. We use lexical variability in the present paper, because the curve associated with lexical variability can be clearly distinguished from the curve associated with communicative success on the graphs.
- *Invention* $I(t)$ is the average number of inventions per agent in past language games, i.e. it is the total number of words invented by all the agents in the population in the language games contained in the interval $[1,t]$ divided by the size of the population.
- *Adoption* $A(t)$ is the average number of adoptions per agent in past language games, i.e. it is the total number of games in which an agent adopts an expression used by another agent in the games contained in the interval $[1,t]$ divided by the size of the population.

The first series of experiments we have conducted study *language emergence* for different population sizes. Figures 1 to 4 show the evolution over time of *communicative success, lexical variability, invention* and *adoption* for such experiments. In all the experiments described in this paper, the length w of the sliding window used for computing *communicative success* and *lexical variability* is of fifty games[28]. For each population size, the curves describing the evolution of communicative success, lexical variability, invention and adoption on the corresponding graph show the average of fifty simulation runs with different random seeds[29]. Standard deviations have also been computed and are represented by the shaded region drawn around each curve.

[28] The length of the sliding window used in the experiments described in [41,42] was of ten games, and that used in the experiments described in [8] of fifty games.

[29] The results described in [41,42] are the average of ten simulation runs, whereas those reported in [8] are the average of fifty simulation runs.

It can be observed that for the population sizes considered in this paper the curves describing the evolution of communicative success and lexical variability follow the *S-shaped curve* typically observed in the spreading of new language conventions in human populations [1]. This is also the case for the experiments described in [41] and [42].

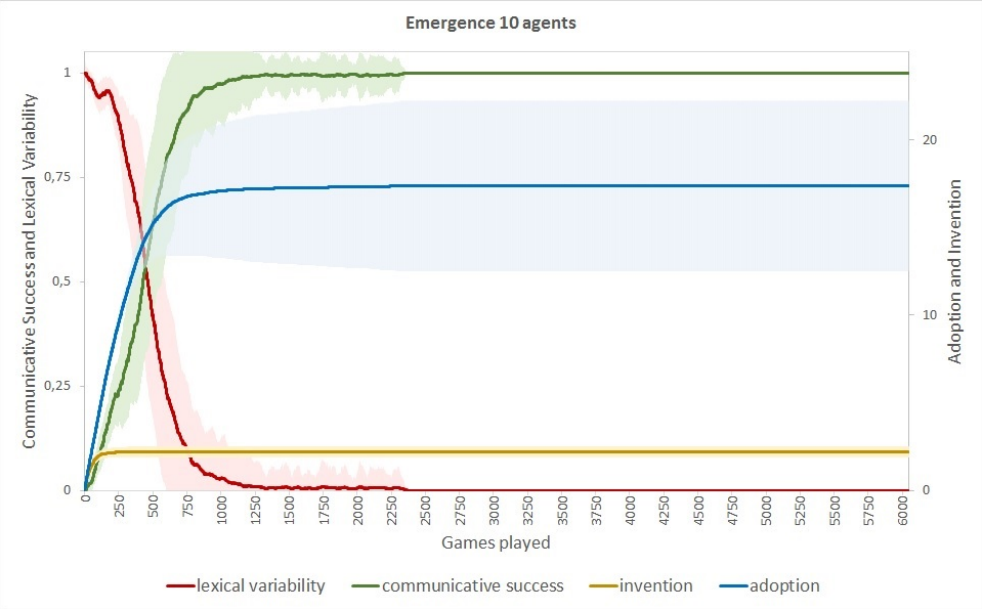

Fig. 1 Evolution of communicative success, lexical variability, invention and adoption in an experiment with ten agents. The results shown are the average of fifty simulation runs with different random seeds.

The results of the four experiments we have carried out to study the emergence of a language system for Boolean coordination with different population sizes are summarised in table 1. Each row describes the results of one experiment. The population size of the experiment is shown in the first column, the maximum number of games required to reach *full communicative success* (i.e. $S(t) = 1.0$) in the fifty simulation runs of that experiment in the second column, the minimum number of games required to reach full communicative success in the fifty simulation runs in the third column, the maximum number of games required to reach *null lexical variability*[30] (i.e. $V(t) = 0.0$) in the fifty simulation runs of that experiment in the fourth column, and the minimum number of games required to reach null lexical variability in the fifty simulation runs in the fifth column.

For the experiments we have carried out, it can be observed that the time to convergence (to reach full communicative success) does not grow too fast with respect to the population size. In particular, for the experiments shown in table 1, the time to convergence is less than $6 \cdot n^2 \cdot \ln(n)$, where n is the population size[31].

[30] Note that null lexical variability means that all the agents in the population prefer the same expression (both word and position) for naming the Boolean operators (`and`,`or`,`not`) and the grouping operator (`id`).

[31] In [17], the number of words invented per object in the naming game is estimated to be in $O(n)$, where n is the population size, and the time for $n/2$ words to spread in the population in $O(n^2 \cdot \ln(n))$. The asymptotic

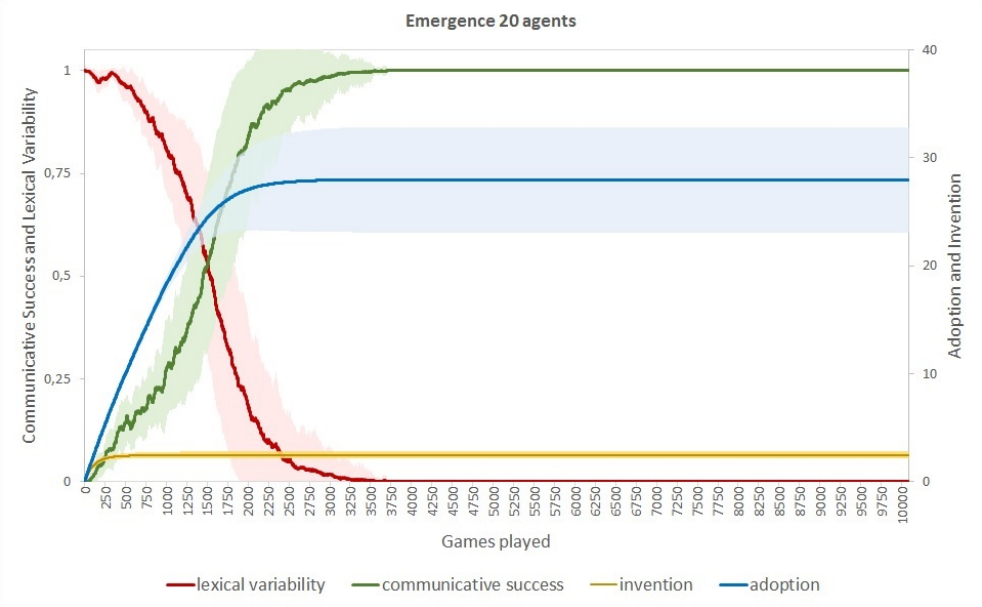

Fig. 2 Evolution of communicative success, lexical variability, invention and adoption in an experiment with twenty agents. The results shown are the average of fifty simulation runs with different random seeds.

Number of games to reach full communicative success and null variability in emergence experiments				
Pop. size	Max number of games to reach $S(t) = 1.0$	Min number of games to reach $S(t) = 1.0$	Max number of games to reach $V(t) = 0.0$	Min number of games to reach $V(t) = 0.0$
10	1280	349	1292	386
20	3703	1231	3704	1234
40	7941	4020	7957	4075
80	21230	10583	21232	10587

Table 1 Maximum and minimum number of games to reach full communicative success ($S(t) = 1.0$) and null lexical variability ($V(t) = 0.0$) in the fifty simulation runs of each experiment studying language emergence. Each row displays the results of one particular experiment, which consists of fifty simulation runs with different random seeds and a population of the size indicated in the first column.

The minimum and maximum average number of games played by each agent until full communicative success and null lexical variability are reached in the experiments studying language emergence are shown in table 2. These results are computed by dividing the corresponding values in table 1 by the population size of each experiment.

[42] describes an experiment in which ten agents[32] play language games about propositional logic formulas that represent *recursive combinations of basic concepts using negation, conjunction and disjunction*. In this experiment, full communicative success is reached in 1050 games and full coherence (i.e. null lexical variability) in 1250 games. The main qualitative difference between the results in [42] and those obtained in the present paper is that the time span between reaching full communicative success and reaching null lexical vari-

notation $O(g(n))$ is defined as follows. For a given function $g(n)$, we denote by $O(g(n))$ the set of functions $O(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq c \cdot g(n) \text{ for all } n \geq n_0\}$.

[32] The experiments described in [56, 8] also use a population of ten agents.

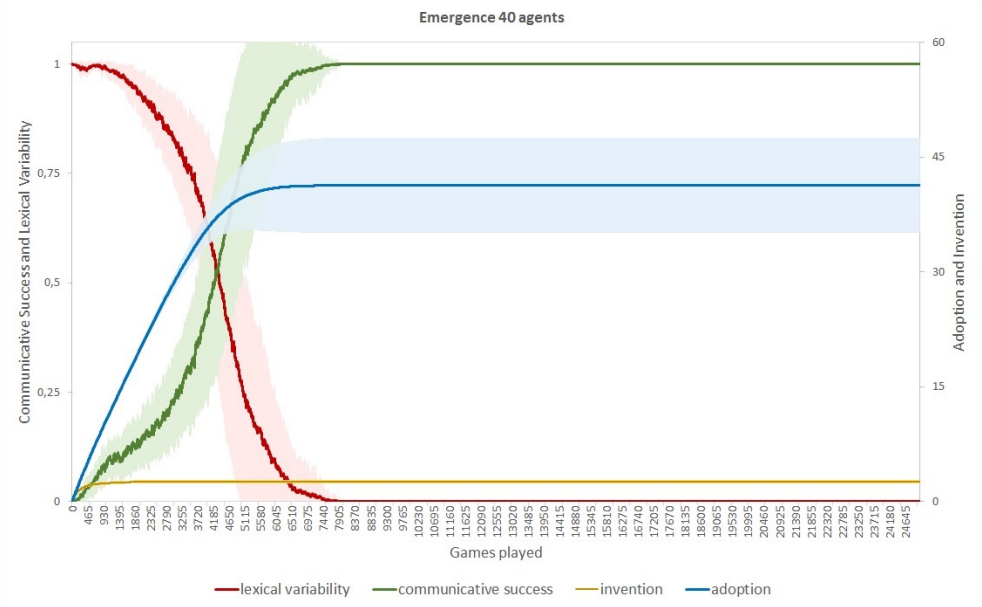

Fig. 3 Evolution of communicative success, lexical variability, invention and adoption in an experiment with forty agents. The results shown are the average of fifty simulation runs with different random seeds.

Games per agent to reach full communicative success and null variability in emergence experiments				
Pop. size	Max number of games played per agent to reach $S(t) = 1.0$	Min number of games played per agent to reach $S(t) = 1.0$	Max number of games played per agent to reach $V(t) = 0.0$	Min number of games played per agent to reach $V(t) = 0.0$
10	128	34.90	129.20	38.60
20	185.15	61.55	185.20	61.70
40	198.52	100.50	198.92	101.87
80	265.37	132.29	265.40	132.34

Table 2 Maximum and minimum average number of games played by each agent until full communicative success and null lexical variability are reached in the fifty simulation runs of each experiment. Each row displays the results of one experiment, which consists of fifty simulation runs with different random seeds and a population of the size indicated in the first column.

ability is larger in [42] than in the experiments studying language emergence described in the present paper. This might be due to the fact grammar rules whose scores reach zero are not eliminated in [42]. In a similar experiment described in [41], in which ten agents play language games only about *non-recursive logical formulas* constructed from basic concepts using negation and ten binary Boolean functions, full communicative success is reached in 1950 games, and full coherence (i.e. null lexical variability) in 4600 games[33]. As it can be observed, the time span between reaching full communicative success and reaching null lexical variability is greater than in [42]. A plausible explanation for this might be the higher number of Boolean operators (eleven versus three) used in the semantic representations of

[33] The number of games required to reach *full communicative success* (or *full coherence*) reported in [41, 42] is the *maximum* number of games required in the ten simulation runs carried out in those experiments.

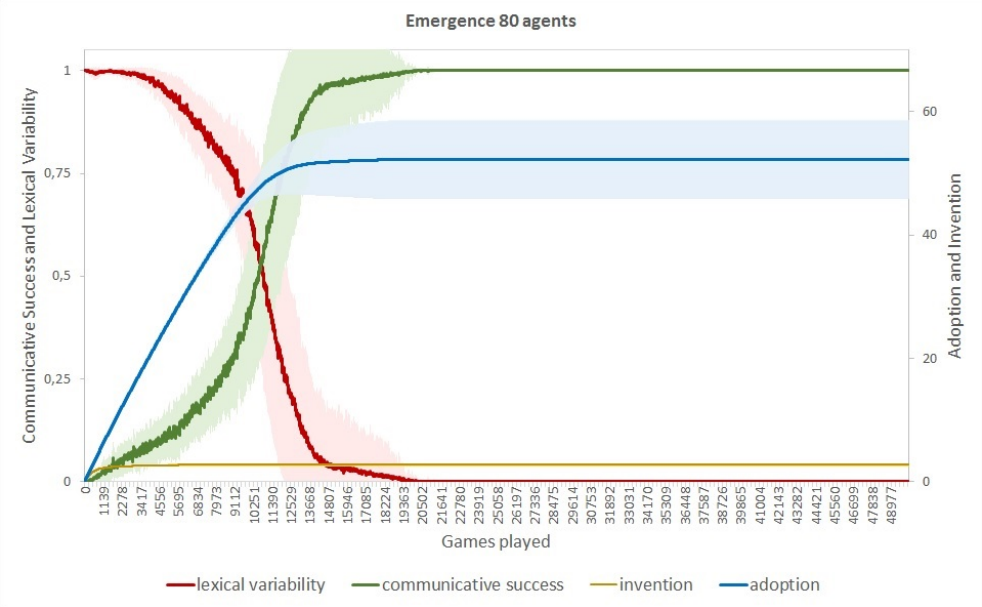

Fig. 4 Evolution of communicative success, lexical variability, invention and adoption in an experiment with eighty agents. The results shown are the average of fifty simulation runs with different random seeds.

this experiment, and the facts that most of these Boolean operators are not commutative, and that grammar rules whose scores reach zero are not eliminated either in [41].

It should also be noted that the score update strategy used in [41] and [42] is similar to that used in the present paper (i.e. *only hearer* with score update parameter values $\delta_h^+ = 0.1$ and $\delta_h^- = 0.1$). However, the parameter value used for initializing the scores of new lexical entries in [41] and [42] was $\mu = 0.1$, whereas in the present paper is $\mu = 0.5$. This difference, together with the facts that the results reported in [41,42] correspond to the average of ten independent simulation runs instead of fifty simulation runs and that the length of the sliding window used in these papers was of ten games instead of fifty games, may also account for the differences between the results obtained in those papers and the present one.

Table 3 shows the average number of inventions per agent $I(t)$ at the end of a simulation run and its standard deviation $\sigma_{I(t)}$ for the fifty simulation runs conducted for each experiment, the average total number of words invented by all the agents in the population in the fifty simulation runs of each experiment (column four), and the average number of adoptions per agent $A(t)$ at the end of a simulation run and its standard deviation $\sigma_{A(t)}$ for the fifty simulation runs carried out for each experiment. It can be observed that each agent adopts only a subset of the total set of words invented by all the agents in the population, and that the total number of words invented by the population is less than $6 \cdot n$ for the experiments we have carried out, where n is the population size. Furthermore, the percentage of words adopted per agent of the set of all the words invented by the population (column seven in table 3) decreases as the population size of the experiment is increased.

Table 4 shows the maximum and minimum period of time during which the population keeps inventing new words in the fifty simulation runs carried out in each experiment, and the maximum and minimum period of time during which the agents keep adopting words invented by other agents in the fifty simulation runs of each experiment. In particular, for each

Average number of inventions per agent and adoptions per agent at the end of the simulation runs						
Pop. size	Average number of inventions per agent $I(t)$	$\sigma_{I(t)}$	Average number of words invented by all agents	Average number of adoptions per agent $A(t)$	$\sigma_{A(t)}$	Percentage of words adopted per agent from words invented by all
10	2.14	0.29	21.4	16.29	3.33	76%
20	2.48	0.25	49.6	27,99	4.85	56%
40	2.61	0.19	104.4	41.28	6.16	39%
80	2.73	0.13	218.4	52.23	6.38	24%

Table 3 Average number of inventions and adoptions per agent at the end of a simulation run. Each row describes one experiment with a population of the size specified in the first column. Columns two and three show the average number of inventions per agent and its standard deviation for the fifty simulation runs carried out in each experiment. Columns five and six give similar information for the average number of adoptions per agent and its standard deviation. Column four displays the average total number of words invented by the population in the fifty simulation runs carried out in each experiment. Column seven shows the average percentage of words adopted per agent of the set of all the words invented by the population in the fifty simulation runs of each experiment.

simulation run s_i we compute $games^{s_i}_{max(I)}$, the number of games it takes for the population to reach the maximum number of inventions in that simulation run, and $games^{s_i}_{max(A)}$, the number of games it takes for the population to reach the maximum number of adoptions in that simulation run. The values in the second and third column of row j are the maximum and the minimum of the set of numbers $\{games^{s_i}_{max(I)}\}_{i=1...50}$ obtained for the fifty simulation runs of the experiment described in row j. Similarly, the values in the fourth and fifth column of row j are the maximum and the minimum of the set of numbers $\{games^{s_i}_{max(A)}\}_{i=1...50}$ obtained for the fifty simulation runs of the experiment described in row j.

In general, as it can be observed in table 4 and figures 1 to 4, invention grows rapidly reaching its maximum early on in each simulation run, whereas adoption reaches its maximum close to the time at which full communicative success is reached (see table 1). This behavior is also observed in the experiments described in previous papers. For example, in [42] the agents invent new words during the first 700 games, and adopt words invented by others during the first 950 games; and in [41] the agents invent new words during the first 550 games, and adopt words invented by others during the first 1900 games[34]. The maximum numbers of games during which the agents adopt words in the experiments described in [41, 42] are also close to the maximum number of games required to reach full communicative success reported for those experiments (1050 games in [42] and 1950 games in [41]).

Figures 5 to 16 show the results of several experiments studying language transmission across generations. The agents in the population are divided into three groups: the elder, the adults and the young. Every t_r games, the elder (approximately one third of the population[35]) are replaced with new agents which have no lexical entries for Boolean operators, the previous adults become the elder, the young the adults, and the new agents the younger generation. As a consequence the population is completely renewed every $3 \cdot t_r$ games, i.e. every three *turnover intervals*.

In particular, we have carried out experiments for populations of ten, twenty, forty and eighty agents, and turnover intervals of 500 to 3000 games. In all the experiments, *commu-*

[34] The number of games during which the agents invent or adopt words reported in [41,42] is the *maximum* for the ten simulation runs carried out in those experiments.

[35] The initial groups of elder, adult and young agents contain $\frac{n}{3}, \frac{n}{3}+n\ mod(3)$ and $\frac{n}{3}$ agents respectively, where n is the population size, $\frac{n}{3}$ is the integer division of n by 3 and n mod(3) the remainder.

Maximum and minimum number of games during which the agents keep inventing and adopting				
Pop. size	Max number of games to reach $\max(I(t))$	Min number of games to reach $\max(I(t))$	Max number of games to reach $\max(A(t))$	Min number of games to reach $\max(A(t))$
10	354	75	1225	291
20	1668	280	3578	1181
40	4899	1253	7799	3861
80	11735	4227	20860	10523

Table 4 Maximum and minimum number of games during which the agents keep inventing new words in the fifty simulation runs of each experiment (second and third columns). Maximum and minimum number of games during which the agents keep adopting words invented by other agents in the fifty simulation runs of each experiment (fourth and fifth columns). The population size of each experiment is in the first column.

nicative success, invention and *adoption* decrease significantly when new agents are introduced into the population, and *lexical variability* increases.

In the experiments studying language transmission for a population of ten agents and turnover intervals of lengths 1500 and 1000 (figures 5 and 6), we can observe that full communicative success and null lexical variability are reached during a large part of most turnover intervals. There is also a steady decrease in the average number of adoptions per agent in succeeding generations, which can be explained by the fact that new agents learn a language already established in the population (i.e. the language that is transmitted from generation to generation), which uses fewer variations for expressing each Boolean operator.

In the experiment studying language transmission for ten agents and turnover interval of length 500 games (figure 7), full communicative success and null lexical variability are reached during the last part of most turnover intervals, although there are a few turnover intervals where communicative success reaches values over 0.98 for a very short period of time before the next generation of agents is introduced. The population size and length of the turnover interval used in this experiment are similar to those used in the experiments described in [41,42]. In [42], each generation reached communicative success and coherence values over 0.99 and 0.98 respectively, and in [41] over 0.96 and 0.94 respectively.

The experiments studying language transmission for a population of twenty agents and turnover intervals of lengths $t_r = 1500$ and $t_r = 1000$ games (figures 8 and 9) show roughly similar patterns of behaviour to those produced by a population of ten agents and turnover intervals of lengths 1000 and 500 respectively, except that a language system for Boolean coordination takes more than one generation to emerge[36]. But once such a language system is established in one generation, it is reliably transmitted to succeeding generations.

In the experiment studying language transmission for a population of twenty agents and turnover interval of 500 games (figure 10), it seems that a shared language system for Boolean coordination does not emerge until the seventh generation. From that moment on communicative success reaches values between 0.95 and 0.99 in succeeding generations, which means that the agents in such generations share a language system for Boolean coordination which allows the unambiguous communication of at least 95% of the higher order logic terms constructed from the common set of basic concepts communicated by the agents.

In figures 7 and 10 to 16, we can also observe that the average number of adoptions per agent (i.e. $A(t)$) reaches its minimum after a certain number of generations. In these graphs, $A(t)$ stabilizes after reaching its minimum, and it does not decrease anymore.

[36] Note that in figures 6 and 9, the length of the first turnover interval is different from 1000 games, but the length of the rest of the turnover intervals is of 1000 games.

The experiments with a population of forty agents and turnover interval of 1500 games (figure 11), and with a population of eighty agents and turnover interval of 3000 games (figure 14) follow a roughly similar pattern of behaviour, where full communicative success is reached at the end of each turnover interval, the main difference being the number of games (and hence generations) a shared language system for Boolean coordination takes to emerge. But the number of games a shared language system takes to emerge in the experiment studying language transmission with eighty agents and turnover interval of 3000 games (see figure 14) is consistent with (i.e. approximately equal to) the number of games a shared language system takes to emerge in the experiment studying language emergence for a population of eighty agents (see figure 4 and table 1). This relation also holds for the experiment studying language transmission with forty agents and turnover interval of 1500 games described in figure 11 and the experiment studying language emergence for a population of forty agents described in figure 3. The number of games a shared language system for Boolean coordination takes to emerge in any experiment studying language transmission for a population of ten or twenty agents is also consistent with the number of games a shared language system takes to emerge in the experiment studying language emergence for the corresponding population (see figures 1, 2, 5, 6, 7, 8, 9, 10, and table 1).

An index that might help understand the results of the experiments studying language transmission is *the number of games each agent can play during one turnover interval* $g_r^a = t_r/n$, that is, the length of the turnover interval t_r divided by the population size n of the experiment. For example, $g_r^a = 50$ for the experiments described in figures 7 and 9, which show a similar pattern of behaviour. Similarly, $g_r^a = 37.5$ for the experiments described in figures 11 and 14, which also exhibit a similar pattern of behaviour. A third group of experiments we have not discussed previously consists of the experiments described in figures 10, 12 and 15, which show a similar behaviour pattern where communicative success forms a pointed shape and reaches values between 0.95 and 0.98 in most of the turnover intervals that take place after a language system for Boolean coordination had emerged in the population. The number of games each agent can play in the turnover interval of the experiments belonging to this third group of experiments is $g_r^a = 25$.

We have also conducted two experiments where the number of games each agent can play in the turnover interval g_r^a is less than 25. The first experiment studies language transmission for a population of forty agents and a turnover interval of 800 games (see figure 13). In this experiment, where $g_r^a = 20$, communicative success forms a pointed shape and reaches values between 0.90 and 0.94 in the last turnover intervals, which means that the language system shared by the agents in the last generations allows the unambiguous communication of at most 94% of the higher order logic terms the agents try to communicate to each other. The second experiment studies language transmission for a population of eighty agents and a turnover interval of 1500 games (see figure 16). In this experiment, where $g_r^a = 18.75$, a shared language system for Boolean coordination does not emerge in the 49500 games played by the agents during the whole simulation. However, we can observe that communicative success keeps increasing during the simulations, reaching values close to 0.78 in the last two turnover intervals, and that lexical variability and adoption decrease.

The above classification of the experiments we have carried out in terms of the number of games each agent can play in the turnover interval (i.e. g_r^a) suggests that, if the agents are allowed to play at least a certain number of games during the turnover interval, one third of the population (i.e. the new generation) will be able to learn the language of the other two thirds (i.e. the previous generations) before the next generation of agents is introduced into the population. The particular number of games per agent in the turnover interval required to ensure language transmission (e.g. $g_r^a = 37$ or $g_r^a = 25$) is not important. The interesting

fact is that we observe similar behaviour patterns in experiments with identical values of g_r^a independently of the population size of each experiment. This is so, because language transmission only depends on whether the new agents are able to learn the language of the rest of the population (i.e. the language that is transmitted across generations) before the next generation of agents is introduced, so that the agents that know the language transmitted across generations always outnumber those that must learn it.

Thus the minimum number of games per agent in the turnover interval required to guarantee language transmission in a given experiment is determined by two factors: (1) the probability that a new agent will play a language game with another agent that already knows the language that is transmitted across generations (i.e. the population percentage that is not replaced with new agents during turnover, which is approximately 66.66% in our experiments); and (2) the intrinsic difficulty of the language system the new agents must learn during the turnover interval, which depends on the number of different meanings that can be expressed, and on the complexity of the linguistic constructions used to express them.

6 Conclusions

This paper proposes an agent-based model of the emergence and transmission of a language system for Boolean coordination. The model has been implemented in Prolog, and tested by conducting a series of experiments in which a population of autonomous agents tries to communicate about subsets of objects characterised by higher-order logic terms constructed by recursively combining basic properties with higher-order logic operators such as negation, conjunction or disjunction. The results of the experiments we have performed show that a language system for Boolean coordination emerges as a result of a process of self-organisation of the agents' linguistic interactions when these agents adapt their preferences for vocabulary, syntactic categories and word order to those they observe are used more often by other agents. Such a language system uses linguistic devices such as syntactic categories, word order and function words; and it can be reliably transmitted across generations.

The language systems the agents build in the experiments we have carried out allow the unambiguous communication of higher-order logic terms representing logical combinations of basic properties with non-trivial recursive structure. Therefore, from a semantic point of view, they are as expressive as the language systems built in the experiments described in [42][37], and more expressive than the language systems constructed in the experiments reported in [27,41]. However, the conceptual system, linguistic system, and simplification and repair operations of the agent-based model proposed in the present paper are more general than those defined in [42], because they do not only allow the simulation of the emergence and evolution of a language system for the *Boolean coordination* of *basic properties*, but also for the *Boolean coordination of higher-order logic terms of any Boolean type*, which can represent the meaning of simple sentences but also of constituents such as nouns, sentences, verbs, adjectives, adverbs, prepositions, prepositional phrases and others, taking advantage of the expressiveness of the λ-calculus and categorial grammar.

The agent-based model proposed improves previous work on the emergence and evolution of grammar by introducing an elaborate conceptual system based on the λ-calculus and higher-order logic, which allows the formal representation of the meaning of practically any constituent of natural language expressions. It also puts forward the use of categorial

[37] See appendix A of [42] for a discussion of the expressiveness of the language systems constructed in the experiments reported in that paper, and a proof of the ability of such language systems to unambiguously express every propositional logic formula, which are represented in [42] using Lisp-like notation [35,34].

grammar, which allows formalising a wide range of instances of coordination of Boolean categories, including both traditional constituents and categories not traditionally analysed as forming constituents. Furthermore, it defines simplification and repair operations that infer the meanings and syntactic categories of unknown subexpressions from the meanings and syntactic categories of the subexpressions surrounding them in a given utterance, rather than inducing new grammar rules from previous ones and inventing syntactic categories for unknown subexpressions. In particular, the simplification operators defined in the present paper are more general than those proposed in [27,56,41,42], and they can be formulated in a simpler manner by applying functional abstraction both at the semantic level and at the syntactic level of an association.

Finally, more experiments have been conducted with respect to [41,42], including different population sizes, turnover intervals, and fifty simulation runs with different random seeds per experiment. The experiments studying language emergence we have carried out show that the time to convergence (i.e. to reach full communicative success) and the number of words invented and adopted per agent do not grow too fast with respect to the population size. In particular, in the experiments we have conducted with population sizes between ten and eighty agents, the time to convergence is less than $6 \cdot n^2 \cdot \ln(n)$, and the total number of words invented by the population less than $6 \cdot n$, where n is the population size. In fact, the percentage of words adopted per agent of the set of all the words invented by the population decreases as the population size of the experiment is increased. All this indicates that emergence experiments based on the agent-based model proposed could scale for larger populations, given that neither the computational time nor the memory space requirements grow too fast with the size of the population. Furthermore, the experiments studying language transmission we have performed suggest that language transmission depends on the number of games each agent can play during one turnover interval, which is equal to the length of the turnover interval divided by the population size. In particular, the minimum number of games per agent in one turnover interval required to guarantee language transmission can be determined by: (1) the probability that a new agent will play a language game with another agent that already knows the existing language (i.e. the population percentage that is not replaced with new agents during one turnover interval); and (2) the intrinsic difficulty of the language system the new agents must learn during the turnover interval.

In future work, we would like to study the emergence and evolution of *language strategies* rather than *language systems*. The language strategy used in [41] differs from that used in [42] and from the language strategy proposed in the present paper. However, given that all of them represent different approaches to solve the problem of Boolean coordination, they could be better compared to each other within a framework that considers the emergence and evolution of different language strategies co-existing in a given population and competing with each other. In particular, such a framework may enable direct experimental comparison between the approach proposed in the present paper and those used in [41] and [42]. Competition between such language strategies can be modeled using agent-based models, but it can also be studied using ecosystem simulation and grammatical evolution techniques [3,4].

A second line of research is the study and development of new language strategies based on the co-evolution of syntax and semantics, i.e. on the coordination of the conceptual and linguistic systems built by the individual agents, and not only on the coordination of their linguistic systems. This may include the emergence and evolution of language systems for Boolean coordination, non-Boolean coordination, or other aspects of language such as case-systems or phrase-structure.

Acknowledgements I should like to express my gratitude to Manuel Alfonseca for reading several versions of this paper and providing valuable comments.

References

1. Baroncheli A., Felici M., Caglioti E., Loreto V., and Steels L. Sharp transition towards shared vocabularies in multi-agent systems. *Journal of Statistical Mechanics*, P06014, 2006.
2. K. Ajdukiewicz. Die syntaktische konnexitat. *Studia Philosophica*, 1:1–27, 1935.
3. M. Alfonseca and F.J. Soler-Gil. Evolving an ecology of mathematical expressions with grammatical evolution. *Biosystems*, 111(2):111–119, 2013.
4. M. Alfonseca and F.J. Soler-Gil. Evolving a predator-prey ecosystem of mathematical expressions with grammatical evolution. *Complexity*, 20(3):66–83, 2015. Published online DOI 10.1002/cplx.21507.
5. J. Allen. *Natural Language Understanding (second edition)*. The Benjamin/Cummings Publishing Company, 1995.
6. Y. Bar-Hillel. On syntactical categories. *Journal of Symbolic Logic*, 15:1–16, 1950.
7. J. Batali. Computational simulations of the emergence of grammar. In *Approaches to the Evolution of Language: Social and Cognitive Bases*, pages 405–426. Cambridge Univ Press, 1998.
8. K. Beuls and L. Steels. Agent-based models of strategies for the emergence and evolution of grammatical agreement. *PLoS ONE*, 8(3):e58960, 2013.
9. T. Briscoe, editor. *Linguistic Evolution through Language Acquisition: Formal and Computational Models*. Cambridge University Press, Cambridge, 2002.
10. F. Bueno, D. Cabeza, M. Carro, M. Hermenegildo, P. López-García, and G. Puebla. The Ciao Prolog system. reference manual. Technical Report CLIP3/97.1, School of Computer Science, Technical University of Madrid (UPM), August 1997. Available from http://www.clip.dia.fi.upm.es/.
11. B. Carpenter. *Type-Logical Semantics*. MIT Press, 1997.
12. E.A. Cartmill, S. Roberts, H. Lyn, and H. Cornish, editors. *Evolution of Language, Proceedings of the Tenth International Conference EVOLANG*. World Scientific, 2014.
13. A. Church. A formulation of a simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.
14. W.F. Clocksin and C.S. Mellish, editors. *Programming in Prolog*. Springer, fourth edition, 1996.
15. A. Colmerauer, H. Kanoui, R. Pasero, and P. Roussel. Un système de Communication Homme-machine en Francais, Research Report. Technical report, Groupe Intelligence Artificielle, Université Aix-Marseille II, France, 1973.
16. B. de Boer. *The Origins of Vowels Systems*. Oxford University Press, 2001.
17. B. de Vylder. *The Evolution of Conventions in Multi-Agent Systems*. PhD thesis, Artificial Intelligence Lab, Free University of Brussels, 2008.
18. M. Di Sciullo and C. Boeckx, editors. *The Biolinguistic Enterprise. New perspectives on the Evolution and Nature of the Human Language Faculty*. Oxford University Press, 2011.
19. E. Garcia-Casademont and L. Steels. Insight grammar learning. *Journal of Cognitive Science*, 17(1):27–62, 2016.
20. G. Gazdar. A cross-categorial semantics for coordination. *Linguistics and Philosophy*, 3:407–410, 1980.
21. K. Gerasymova, M. Spranger, and K. Beuls. A language strategy for aspect: Encoding aktionsarten through morphology. In *Experiments in Cultural Language Evolution*, pages 257–276. Springer, 2012.
22. B. Grosz, K. Jones, and B. Webber, editors. *Readings in Natural Language Processing*. Morgan Kaufmann, 1986.
23. J. Hurford, M. Studdert-Kennedy, and C. Kight, editors. *Approaches to the Evolution of Language: Social and Cognitive Bases*. Edinburgh University Press, 1998.
24. M. Kay. The MIND system. In *Natural Language Processing*, pages 155–188. Algorithmics Press, 1973.
25. M. Kay. Algorithm schemata and data structures in syntactic processing. Technical Report CSL-80-12, Xerox Corporation, 1980. Reprinted in [22].
26. E. L. Keenan and L. M. Faltz. *Boolean Semantics for Natural Language*. Synthese Language Library, no. 23. Dordrecht: Reidel, 1985.
27. S. Kirby. Learning, bottlenecks and the evolution of recursive syntax. In *Linguistic Evolution through Language Acquisition: Formal and Computational Models*, pages 96–109. Cambridge University Press, 2002.
28. Dall'Asta L., Baronchelli A., Barrat A., and Loreto V. Non-equilibrium dynamics of language games on complex networks. *Physical Review*, 74(3):036105, 2006.
29. J. Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, (65):154–169, 1958.
30. J. Lambek. On the calculus of syntactic types. In *Structure of Language and its Mathematical Aspects: Proceedings of Symposia in Applied Mathematics*, pages 166–178. American Mathematical Society, 1961.
31. J. Lara and M. Alfonseca. Some strategies for the simulation of vocabulary agreement in multi-agents communities. *Journal of Artificial Societies and Social Simulation*, 3(4), 2000. http://jasss.soc.surrey.ac.uk/3/4/2.html.

32. J. Lara and M. Alfonseca. The role of oblivion, memory size and spatial separation in dynamic language games. *Journal of Artificial Societies and Social Simulation*, 5(2), 2002. http://jasss.soc.surrey.ac.uk/5/2/1.html.
33. C. Lyon, C. Nehaniv, and A. Cangelosi, editors. *Emergence of Language and Communication*. Lecture Notes in Computer Science. Springer-Verlag, Berlin, 2007.
34. J. McCarthy. *Formalizing Common Sense. Papers by John McCarthy*. Ablex. Edited by Vladimir Lifschitz, 1990.
35. John McCarthy. Recursive functions of symbolic expressions and their computation by machine, part I. *Communications of the ACM*, 3(4):184–195, 1960.
36. JW. Minett and WSY. Wang, editors. *Language Acquisition, Change and Emergence: Essays in Evolutionary Linguistics*. City University of Hong Kong Press, 2005.
37. R. Montague. The proper treatment of quantification in ordinary English. In *Approaches to Natural Language: Proceedings of the 1970 Stanford Workshop on Grammar and Semantics*. Dordrech: Reidel, 1973. Reprinted in R. Thomanson, editor, Formal Philosophy, 247–270. New Haven: Yale University Press.
38. J. Piaget. *The Equilibration of Cognitive Structures: the Central Problem of Intellectual Development*. University of Chicago Press, 1985.
39. J. Sierra-Santibáñez. Grounded models as a basis for intuitive reasoning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI-2001*, pages 401–406, 2001.
40. J. Sierra-Santibáñez. Grounded models as a basis for intuitive and deductive reasoning: The acquisition of logical categories. In *Proceedings of the European Conference on Artificial Intelligence, ECAI-2002*, pages 93–97, 2002.
41. J. Sierra-Santibáñez. An agent-based model studying the acquisition of a language system of logical constructions. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, AAAI-2014*, pages 350–357. AAAI Press, 2014.
42. J. Sierra-Santibáñez. An agent-based model of the emergence and transmission of a language system for the expression of logical combinations. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, AAAI-2015*, pages 492–499. AAAI Press, 2015.
43. M. Steedman. Dependency and coordination in the grammar of Dutch and English. *Language*, 61:523–568, 1985.
44. L. Steels. A self-organizing spatial vocabulary. *Artificial Life*, 2(3):319–332, 1995.
45. L. Steels. The origins of ontologies and communication conventions in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 1(2):169–194, 1998.
46. L. Steels. The origins of syntax in visually grounded robotic agents. *Artificial Intelligence*, 103(1-2):133–156, 1998.
47. L. Steels. Modeling the cultural evolution of language. *Physics of Life Reviews*, 8:339–356, 2011.
48. L. Steels, editor. *Experiments in Cultural Language Evolution*. John Benjamins, 2012.
49. L. Steels. *The Talking Heads experiment: Origins of words and meanings (Computational Models of Language Evolution) (Volume 1)*. Language Science Press, 2015.
50. L. Steels. Agent-based models for the emergence and evolution of grammar. *Phil. Trans. R. Soc. B*, 371:20150447:1–9, 2016.
51. L. Steels and T. Belpaeme. Coordinating perceptually grounded categories through language: a case study for colour. *Behavioral and Brain Sciences*, 28:469–529, 2005.
52. L. Steels and E. Garcia-Casademont. How to play the syntax game. In *Proceedings of the European Conference on Artificial Life*, pages 479–486. MIT Press, 2015.
53. L. Steels and P. Vogt. Grounding adaptive language games in robotic agents. In *Proceedings of the European Conference on Artificial Life*. MIT Press, 1997.
54. M. Tomasello. *Constructing a Language: a Usage-Based Theory of Language Acquisition*. Harvard Univ Press, 2003.
55. M. Tomasello. Acquiring linguistic constructions. In *Handbook of child psychology*. Wiley Online Library, 2006.
56. P. Vogt. The emergence of compositional structures in perceptually grounded language games. *Artificial Intelligence*, 167(1-2):206–242, 2005.
57. M.A. Williams, J. McCarthy, P. Gardenfors, C. Stanton, and A. Karol. A grouding framework. *Autonomous Agents and Multi-Agent Systems*, 19:272–296, 2009.
58. L. Wittgenstein. *Philosophical Investigations*. Macmillan, New York, 1953.
59. W. Zuidema and B. de Boer. Multi-agent simulations of the evolution of combinatorial phonology. *Adaptive Behavior*, 18(2):141–154, 2010.

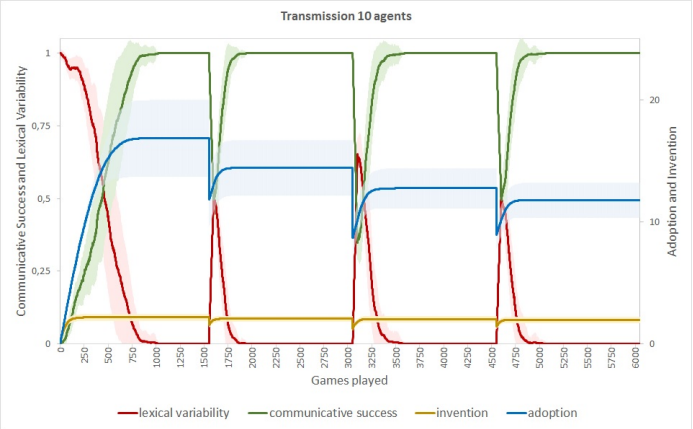

Fig. 5 Experiment studying language transmission: 10 agents and turnover interval of 1500 games.

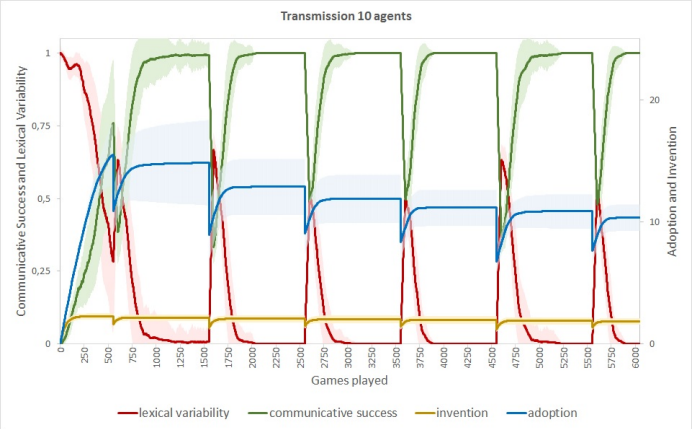

Fig. 6 Experiment studying language transmission: 10 agents and turnover interval of 1000 games.

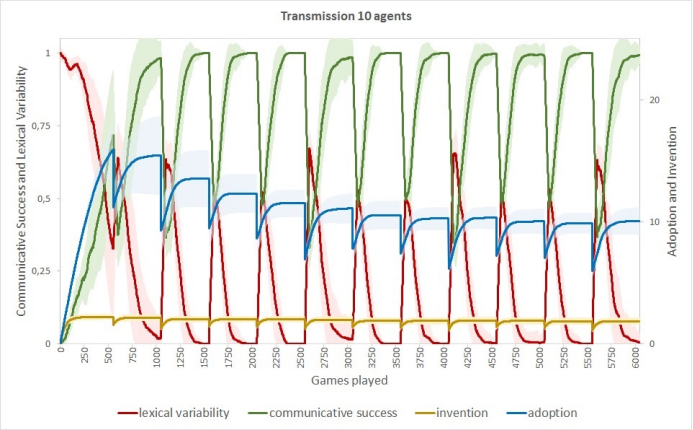

Fig. 7 Experiment studying language transmission: 10 agents and turnover interval of 500 games.

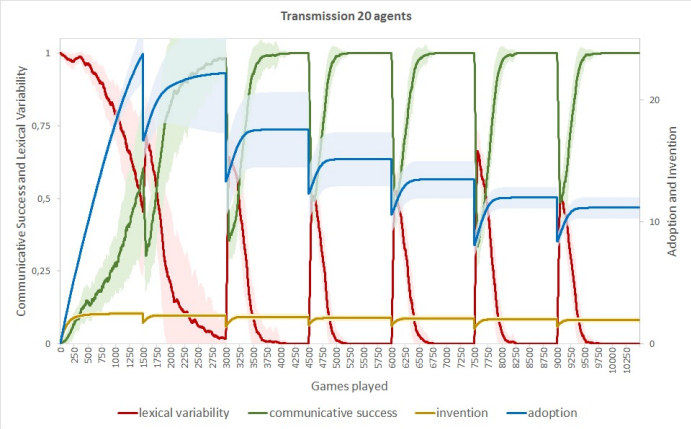

Fig. 8 Experiment studying language transmission: 20 agents and turnover interval of 1500 games.

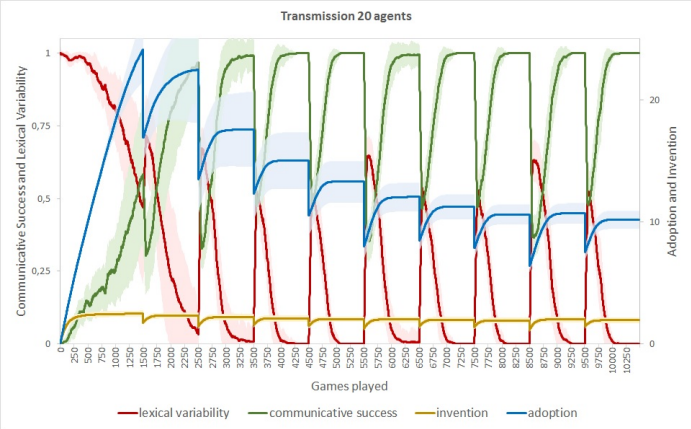

Fig. 9 Experiment studying language transmission: 20 agents and turnover interval of 1000 games.

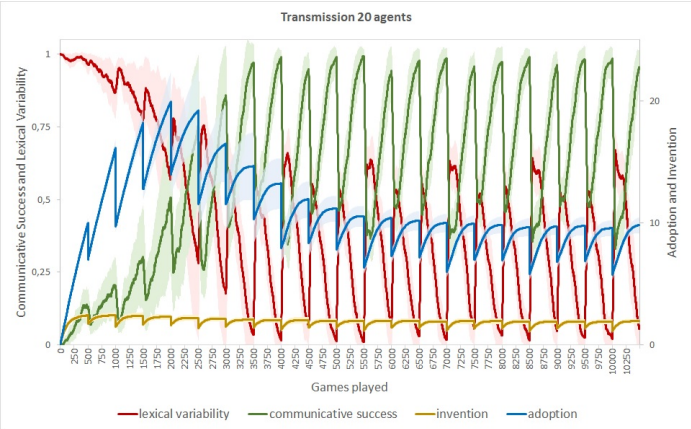

Fig. 10 Experiment studying language transmission: 20 agents and turnover interval of 500 games.

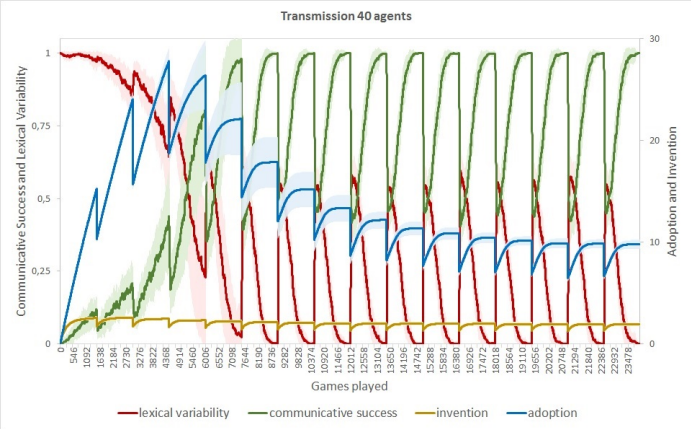

Fig. 11 Experiment studying language transmission: 40 agents and turnover interval of 1500 games.

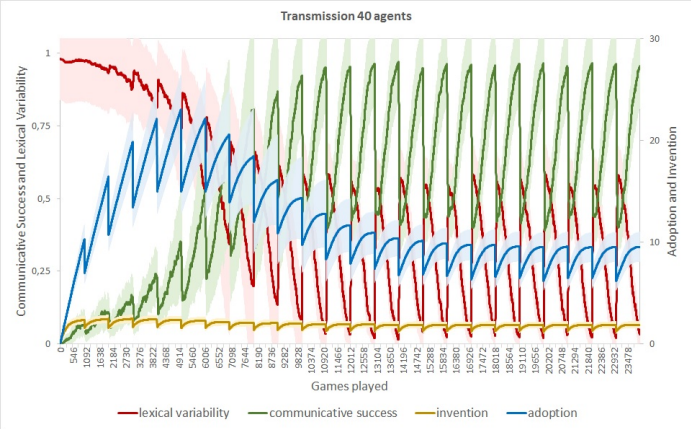

Fig. 12 Experiment studying language transmission: 40 agents and turnover interval of 1000 games.

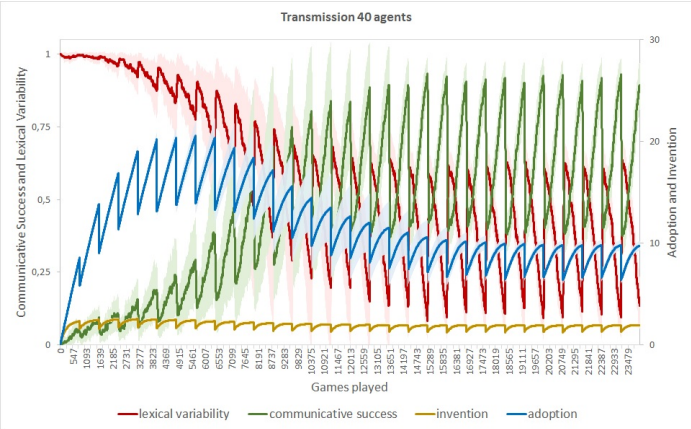

Fig. 13 Experiment studying language transmission: 40 agents and turnover interval of 800 games.

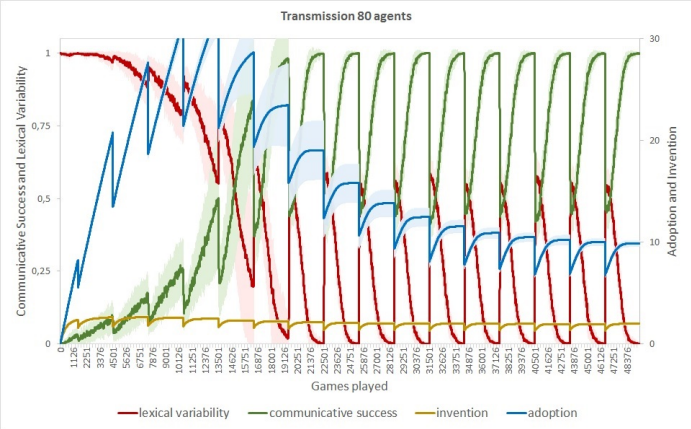

Fig. 14 Experiment studying language transmission: 80 agents and turnover interval of 3000 games.

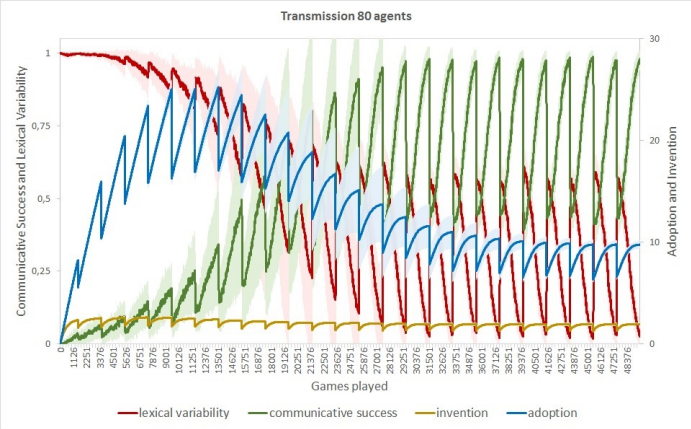

Fig. 15 Experiment studying language transmission: 80 agents and turnover interval of 2000 games.

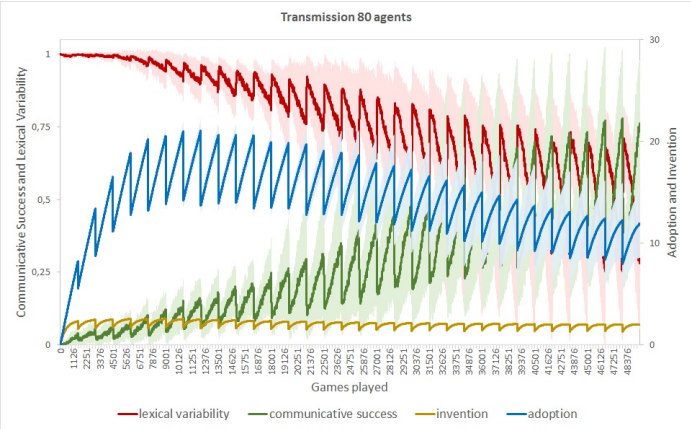

Fig. 16 Experiment studying language transmission: 80 agents and turnover interval of 1500 games.