Natural Language Generation and Parsing as Heuristic Planning Problems

Josefina Sierra-Santibáñez

Technical University of Catalonia, Spain, maria.josefina.sierra@upc.edu*

Abstract

This paper formulates the problems of natural language generation and parsing as particular instances of the classical planning problem. It assumes the existence of a Categorial Grammar lexicon from which the preconditions and effects of available actions are obtained. A declarative formalization of heuristics for action selection is used to guide the search for solutions. Heuristics for mapping formulas in the description logic DL-Lite into English sentences and backwards, and examples of application to Human Robot Interaction (HRI) are presented to illustrate the effectiveness of the approach.

Introduction

This paper studies the problem of human–robot communication in scenarios where a shared ontology should be built as a result of human–robot cooperation (e.g. map building). In particular, it addresses the issues of what language a human–robot team should use to communicate with each other, and how parsing and generation could be carried out flexibly, robustly and efficiently to facilitate cooperation.

We restrict attention to controlled languages (CL) (Huijsen, 1998; Kittredge, 2003) to deal with the first issue. The heuristic planning approach to parsing and generation proposed addresses the second issue. The logical language used for the declarative formalization of heuristics allows encoding a wide variety of information (syntactic, semantic, domain model and contextual) which can be exploited indistinctively by the planner for parsing and generation, to deal with anaphora, semantic/structural ambiguity, grammatical errors, or missing vocabulary in a flexible/robust manner.

The rest of the paper is organized as follows. First, we review basic concepts of λ-Calculus and Categorial Grammar (CG). Then, the formulation of natural language generation and parsing as planning problems is presented. Next, a declarative formalization of heuristics for action selection and the approach to planning associated with it are described. Finally, the approach is illustrated with heuristics that allow mapping DL-Lite(R,⊓) formulas into English sentences and backwards, and examples of situated word learning and word formation mechanisms in a HRI scenario.

The λ-Calculus and Categorial Grammar

We use the λ-Calculus (Church 1940) as a formalism to represent the meanings of the basic expressions of a language, and a compositional method to define the meaning of non-basic expressions from the meanings of basic expressions.

In the simply typed λ-Calculus there are infinitely many types of terms constructed from a finite set of basic types. A common choice of basic types in linguistics consists of a type Bool of Boolean values and a type Ind of individuals. The set of simple types is then built up by closing the set of basic types under the construction of total function types.

Terms in the λ-Calculus are built up out of variables and constants (Carpenter 1997). For each type τ we assume: 1. Varτ: a countably infinite set of variables of type τ. 2. Conτ: a collection of constants of type τ. The collections Termτ of λ-terms of type τ are defined as the smallest sets such that: 1) Varτ ⊆ Termτ; 2) Conτ ⊆ Termτ; 3) τ(β) ∈ Termτ if α ∈ Termσ→τ and β ∈ Termσ; 4) λx.α ∈ Termτ if τ = σ → ρ, x ∈ Varτ and α ∈ Termρ.

A term of the form α(β) is said to be a functional application, and one of the form λx.α a functional abstraction.

In categorial grammar (Ajdukiewicz 1935; Bar-Hillel 1950; Lambek 1958, 1961), every syntactic category corresponds to some λ-Calculus 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} of basic categories, which abbreviate noun phrase, noun, and sentence respectively, and are associated with the following λ-Calculus types Type(np) = Ind, Type(s) = Bool, and Type(n) = Ind → Bool. BasCat is used to generate an infinite set Cat of functional categories that is the smallest set such that: 1) BasCat ⊆ Cat; 2) (A\B) ∈ Cat if A, B ∈ Cat; 3) (B/A) ∈ Cat if A, B ∈ Cat.

A category B/A or A\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\B looks for its argument to the left.

The main operation in Applicative Categorial Grammar (ACG) is the concatenation of an expression of 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.
The correspondence Type: Cat → λ-Calculus Types for Cat is
(Type(A, B) = Type(B/A) = Type(A) → Type(B))
We assume we have a finite set BasExp of basic expressions,
e.g. some subset of English words or possibly complex sequences
of words that constitute a single lexical entry.

**Definition 1 (Categorial Lexicon)** A categorial lexicon is
a relation Lex ⊆ BasExp × Term × Cat such that if a lexical
category e ⇒ α: A ∈ Lex, then α ∈ Term

**Definition 2 (PSACG)** The phrase-structure rules of AC
are all instances of the following application schemes.

1. α: B/A ∈ Lex, e_1 ∈ Al, e_2 ∈ Al Forward Application
2. α: A/B ∈ Lex, e_1 ∈ Al, e_2 ∈ Al Backward Application

The forward scheme states that if e_1 is an expression of cat-
egory B/A with meaning α, and e_2 is an expression of cat-
egory A with meaning λ, then expression (e_1 concate-
nated to e_2) is an expression of category B with meaning
λ(β). The backward scheme is interpreted similarly.

**Phrase structure trees** formalize derivations where: the
leaves are lexical entries, and the internal nodes the result of
applying a phrase structure rule to their immediate succes-
sors. Let (α: C, e, (T_1, T_2)) denote the tree rooted at cate-
gory α: C, e with daughter trees T_1, T_2 (Carpenter 1997).

**Definition 3 (AdmTree)** The set T of admissible trees
is the least set such that:
1. (α: C, e) ∈ AdmTree if e ⇒ α: C ∈ Lex
2. (α: β: C, e, (T_1, T_2)) ∈ AdmTree if T_1, T_2 ∈ AdmTree

The language recognized by lexicon Lex is the expression
set LLex = {e | T ∈ AdmTree and Root(T) = (α: C, e)}.

**Parsing and Generation as Planning Problems**
This section proposes a formulation of generation (i.e. map-
ping λ-calculus into natural language expressions) and parsing
(i.e. the reverse mapping) as classical planning problems.

**Definition 4** A classical planning problem P is a tuple
(S, s_0, S_G, A, f), where: 1) S is a set of states; 2) s_0 ∈ S
is the known initial state; 3) S_G ⊆ S is the non-empty set of
goal states; 4) A(s) ⊆ A is the set of applicable actions in
state s; 5) f is the deterministic transition function. f(a, s)
is the state resulting from performing action a in state s.

**Domain model** of a planning problem is the set of states and
available actions. A problem instance is a pair of initial
and goal states. As we will see, the planning problems of
parsings and generation share the same domain model.

**Definition 5** Given a lexicon Lex, a construction is a pair
((e, B), α) such that there is a tree T ∈ AdmTree, with
Root(T) = (α: B, e). The expression and category (e, B)
form the syntactic part and the λ-term α the semantic part.

The notion of a construction (Goldberg 1995) has been for-
malized in Fluid Construction Grammar (Steele 2011) using
feature structure unification and merge for parsing and gen-
eration in the context of language evolution experiments.

**Definition 6** Parsing and Generation Planning Problems
A state s is a set of constructions Ψ. The initial state s_0 is
the empty set of constructions. The set of goal states S_G is
S_G = {ψ | ((e, B), β) ∈ Ψ} the set of states which contain a
construction ((e, B), β) for parsing problem (e, Lex); and
II) S_G = {ψ | ((e, B), α) ∈ Ψ} the set of states which contain a
construction ((e, B), α) for generation problem (α, Lex).
The available actions set A = {add_construction(C_1, C_2)}:
A(s), the set of applicable actions, and f(a, s), the transition function, are defined by
specifying the preconditions and effects of each action.

**Preconditions:** add_construction(C_1, C_2) can be executed
in state s if C_1, C_2 ∈ Ψ ∪ Lex, and if C_1 can be applied
to C_2 using the application schemes of AC (see below).

**Effects:** the result of executing add_construction(C_1, C_2) in
is s′ = Ψ ∪ C_1(C_2). C_1(C_2) denotes the result of
applying construction C_1 to construction C_2, and equals
• ((e_1, B), β(α)), if C_1 is of the form ((e_1, B/A), β)
and C_2 is of the form ((e_2, A), α), (forward app.);
or
• ((e_2, e_1, B), β(α)), if C_1 is of the form ((e_1, A/B), β)
and C_2 is of the form ((e_2, A), α), (backward app.).

Note the domain model (states and actions) of the planning
problems (α, Lex) generation and (e, Lex) parsing is deter-
ded by the CG lexicon Lex specified in its formulation.

**Heuristic Planning with Declarative Formalization of
Heuristics for Action Selection**
We propose solving the generation and parsing problems
using a heuristic planner, i.e. a planner that explores the
space of selectable states rather than that of reachable states.
A state is selectable (respectively, reachable) if it can be
generated by applying a sequence of selectable (resp. exec-
utable) actions to the initial state. (Sierra-Santibanez 1998) describes
a heuristic forward chaining planner which uses a
declarative formalization of heuristics for action selection
to circumscribe the set of states that should be considered
to those that are selectable according to a strategy for action
selection. The advantage of representing heuristics declar-
tively is that it is possible to refine the action selection strat-
yegy of the planner/robot by simple additions of better heuris-
tics, as proposed in the advice taker (McCarthy 1959).

An action selection strategy Σ is a set of action selec-
tion rules. An action selection rule is an implication
whose antecedent depends on a state s, and whose conse-
quent takes the following forms: Good(a, s), Bad(a, s) or
Better(a, b, s). The heuristic planner determines the set of
selectable actions for a state s by interpreting Σ non-
monotonically (Lifschitz 1995) and considering that an
executable action a is selectable in s if: (1) a is good for s; or
(2) if there are no good actions for a, and a is not bad for s.
Next, we present an action selection strategy that allows a
heuristic planner to map DL-Lite formulas into English
sentences, and a second strategy it can use for parsing them.

---

1See next sections for examples of constructions, actions, and their application to parse/generate sentences in a HRI scenario.
2The intuitive meaning of these predicates is that performing action a in state s is good, bad, or better than performing action b.
Generating English Sentences for DL-LiteR∈m

DL-Lite (Calvanese et al. 2007, 2011) is a family of Description Logics (DLs) (Baader et al. 2003) studied in the context of ontology-based access to relational databases. DL-LiteR∈m is a member of DL-Lite in which the Terminological Knowledge Base (TBox) consists of inclusion assertions of the form $CI \subseteq CR$. $CI$ and $CR$ denote concepts that may occur on the left and right-hand side, and can be of the form:

$CI \rightarrow B \mid \exists R \mid CI_1 \sqcap CI_2$

$CR \rightarrow B \mid \neg B \mid \exists R \mid \neg \exists R \mid \forall R.B$

where $B$ denotes an atomic concept and $R$ an atomic role.

Inclusion Assertions in DL-LiteR∈m can be translated into FOL formulations of the form $\forall x(\text{CI}(x) \rightarrow \text{CR}(x))$ such that $\text{CI}(x) \rightarrow \text{B}_0(x) \land \bigwedge_{i=1}^{m} A_i(x)$, $\text{CR}(x) \rightarrow B(x) \rightarrow \neg B(x) \mid \exists y R(x,y) \mid \neg \exists y R(x,y) \mid \exists y R(x,y) \land B(y)$ where each $A_i(x)$ can be of the form $B_i(x)$ or $\exists y R_i(x,y)$. $B, B_i$ denote unary predicates, and $R, R_i$ binary predicates.

Lexicon LexR∈m

every $\Rightarrow \lambda U.L.V.(U(x) \rightarrow V(x)) : (\text{nb}/(\text{nb},\text{sc})/\text{ncl})$

everyone $\Rightarrow \lambda U.L.V.(U(x) \rightarrow V(x)) : (\text{nb}/(\text{nb},\text{sc})/\text{ncl})$

is a $\Rightarrow \lambda U.L.x.(u) : (\text{nb}/(\text{nb},\text{sc})/\text{ncl})$

is not a $\Rightarrow \lambda U.L.\neg (u) : (\text{nb}/(\text{nb},\text{sc})/\text{ncl})$

does not $\Rightarrow \lambda U.L.\neg (u) : (\text{nb}/(\text{nb},\text{sc})/\text{ncl})$

robot $\Rightarrow \lambda x.\text{robot}(x) : n$

explores $\Rightarrow \lambda x.\text{explore}(x) : \text{np}/s$

scans $\Rightarrow \lambda x.\text{scan}(x,y) : (\text{np}/(\text{np},\text{sc})/\text{np})$

something $\Rightarrow \lambda U.L.y.\exists x. U(x,y) : (\text{np}/(\text{np},\text{sc})/\text{np})$

a $\Rightarrow \lambda U.L.y.\exists x. U(y,x) \land (U(x,y)) : (\text{np}/(\text{np},\text{sc})/\text{np})$

who2 $\Rightarrow \lambda U.L.\exists x. U(x) \land V(x) : (\text{ncl}/(\text{nb},\text{sc})/\text{np})$

and $\Rightarrow \lambda U.L.\lambda x. U(x) \land V(x) : (\text{np}/(\text{np},\text{sc})/\text{np})$

( Bernardi, Calvanese, and Thorne 2007) define a CG lexicon that captures a fragment of English consisting of sentences whose meanings belong to DL-LiteR∈m. They exploit the syntax-semantics interface provided by the Curry-Howard correspondence between the Lambek Calculus and the λ-Calculus to obtain DL-Lite representations of expressions in this English fragment compositionally while parsing (van Benthem 1987). Although we use the lexicon in ( Bernardi, Calvanese, and Thorne 2007) with slight variations (see Lexicon LexR∈m above), our work differs from theirs in using heuristic planning to solve the parsing problem rather than logical deduction, and in providing an algorithmic solution to the natural language generation problem.

Heuristics for Expressing $\text{CR}(x)$ of $\forall x(\text{CI}(x) \rightarrow \text{CR}(x))$

Given a basic expression $e$ of a lexicon $\text{Lex}$, $C_e$ denotes the construction associated with lexical entry $e \Rightarrow \alpha : A \in \text{Lex}^3$.

1. If $\text{CR}(x)$ is of the form $B(x)$ and there is a lexical entry $e \Rightarrow \lambda x B(x) : A \in \text{Lex}$ such that $A$ is
   (a) $(n)$ (noun), then $\text{good} (\text{add_construction}(C_r,C_{\text{Lex}}(A),s),s)$
   (b) $\text{np}/s$, intrans verb, $\text{good} (\text{add_construction}(C_r,C_{\text{Lex}}(A),s),s)$

2. If $\text{CR}(x)$ is of the form $\neg B(x)$ and there is a lexical entry $e \Rightarrow \lambda x B(x) : A \in \text{Lex}$ such that $A$ is
   (a) $(n)$ (noun), then $\text{good} (\text{add_construction}(C_{\text{Lex}}(A),C_r),s)$
   (b) $\text{np}/s$, then $\text{good} (\text{add_construction}(C_{\text{does_not}(C_r)}),s)$

3. If $\text{CR}(x)$ is of the form $\exists y B(x,y)$ and $e \Rightarrow \lambda y \lambda x B(x,y) : (\text{np}/s)/\text{np} \in \text{Lex}$ (transitive verb), then $\text{good} (\text{add_construction}(C_r,C_{\text{something}(C_{\text{Lex}}(A))},s))$

4. If $\text{CR}(x)$ is of the form $\neg \exists y B(x,y)$ and $e \Rightarrow \lambda y \lambda x B(x,y) : (\text{np}/s)/\text{np} \in \text{Lex}$ (transitive verb), then $\text{good} (\text{add_construction}(C_{\text{does_not}(C_r)}),s)$

5. If $\text{CR}(x)$ is of the form $\exists y B(x,y) \land B(y)$, $e_1 \Rightarrow \lambda y \lambda x B(x,y) : (\text{np}/s)/\text{np} \in \text{Lex}$ and $e_2 \Rightarrow \lambda x B(x) : n \in \text{Lex}$, then $\text{good} (\text{add_construction}(C_r,C_{\text{Lex}}(A),s),s)$

Heuristics for Expressing $\text{CI}(x)$ of $\forall x(\text{CI}(x) \rightarrow \text{CR}(x))$

These rules treat $A_1(x)$, the first conjunct in $\bigwedge_{i=1}^{m} A_i(x)$:

1. If $A_1(x)$ is of the form $B(x)$, and there is an entry $e \Rightarrow \lambda x B(x) : A \in \text{Lex}$ such that $A$ is
   (a) $B_0(x) \equiv \text{true, good} (\text{add_construction}(C_r,C_{\text{Lex}}(A),s),s)$
   (b) $\text{np}/s$ (intrans. verb), $\text{good} (\text{add_construction}(C_r,C_{\text{Lex}}(A),s),s)$

2. If $A_1(x)$ is of the form $\exists y B(x,y)$, and there is a lexical entry $e \Rightarrow \lambda y \lambda x B(x,y) : (\text{np}/s)/\text{np} \in \text{Lex}$, then $\text{good} (\text{add_construction}(C_{\text{Lex}}(A),C_{\text{something}(C_r)}),s))$

These rules treat the rest of the conjuncts in $\bigwedge_{i=1}^{m} A_i(x)$:

1. If there is a subformula $H(x) \land D(x)$ of $\bigwedge_{i=1}^{m} A_i(x)$ such that the current state $s$ contains construction $C_H=\text{Lex}(H(x))$ and lexical entry
   (a) $e_i \Rightarrow \lambda x D(x) : n \in \text{Lex}$, then $\text{good} (\text{add_construction}(C_{\text{condition}(C_{r})(C_{\text{Lex}}(A_i)),(C_{r})(H(x)),s)})$
   (b) $\text{np}/s$, then $\text{Lex} : s \in \text{Lex}$, then $\text{good} (\text{add_construction}(C_{\text{condition}(C_{r})(C_{\text{Lex}}(A_i)),s},s))$

2. If there is a subformula $H(x) \land \exists y B(x,y)$ of $\bigwedge_{i=1}^{m} A_i(x)$ such that the current state $s$ contains a construction $C_H=\text{Lex}(H(x))$ and there is a lexical entry $e_i \Rightarrow \lambda y \lambda x B(x,y) : (\text{np}/s)/\text{np} \in \text{Lex}$, then $\text{good} (\text{add_construction}(C_{\text{condition}(C_{r})(C_{\text{Lex}}(A_i)),(C_{r})(H(x)),s)})$

These rules add “every + noun + who” or “everyone who” to a conjunction of clauses.

1. If $\text{CI}(x)$ is of the form $\bigwedge_{i=1}^{m} A_i(x)$ with $m > 0$, there is no lexical entry of the form $e \Rightarrow \lambda x A_i(x) : n \in \text{Lex}$ for $i = 1,\ldots,m$ (i.e., $B_0(x) \equiv \text{true}$), and the current state $s$ contains a construction $C_{G_i}=\text{Lex}(G_i)$, then $\text{good} (\text{add_construction}(C_{\text{condition}(C_{r})(C_{\text{Lex}}(A_i)),s},s))$
2. If $Cl(x)$ is of the form $B_0(x) \land \bigwedge_{i=1}^{m} A_i(x)$ with\(^6\) $m > 0$, there is a lexical entry $e \Rightarrow \lambda x B_0(x) : n \in Lex$, and the current state $s$ contains a construction $C_G = ((e_G, np\{sl\}, \lambda x(\bigwedge_{i=1}^{m} A_i(x)))$, $\text{good}\_\text{add}\_\text{construction}(C_{every}/C_{who}(C_G) \cap C_t(s), s)$

Termination Heuristic
Given a language generation problem $\alpha$, with $\alpha$ of the form $\forall x (Cl(x) \rightarrow Cr(x))$, if the current state $s$ contains a construction $C_G = ((e_G, stb/(np\{scr\}), \lambda x_1 x_2 (Cl(x_2) \rightarrow X_1(x_2)))$ and another construction $C_H = ((e_H, np\{scr\}, \lambda x_3 Cr(x_3))$, then $\text{good}\_\text{add}\_\text{construction}(C_G, C_H, s)$.

Parsing Sentences in the English Fragment $L_{\text{Lex}_R, R}$

The following heuristics allow a heuristic planner to map English sentences in $L_{\text{Lex}_R, R}$, the language recognized by lexicon $L_{\text{Lex}_R, R}$, into DL-Lite$^R_\bot$ formulas. The set of heuristics has the same structure as that presented for the generation problem, except that the conditions of the rules depend on the form of $e$, the English sentence to be parsed. The action selection strategy for parsing requires adapting all the rules in the action selection strategy for generation. But we only indicate how to adapt one action selection rule in each of the main groups of the strategy for generation.

Heuristics for Parsing the Part of $e$ Expressing $Cr(x)$
(Rule 5) If there are lexical entries $e_1 \Rightarrow \lambda y \lambda x R(y, x) : (np\{s_2\})/np$ and $e_2 \Rightarrow \lambda x B_0(x) : n \in Lex$, and there is an expression $e_h$ such that $e = e_h \cdot e_1 \cdot "a" \cdot e_2$, then $\text{good}\_\text{add}\_\text{construction}(C_a, C_e, C_h, s)$.

Termination Heuristic
Given parsing problem $\alpha$, $Lex_R$, if state $s$ contains two constructions of the form $C_{cl} = ((e_c, stb/(np\{scr\}), \beta)$ and $C_{cr} = ((e_c, np\{scr\}, \alpha)$, and $e_c \cdot e_{cr} = e$, then $\text{good}\_\text{add}\_\text{construction}(C_{cl}, C_{cr}, s)$.

Related Work
(Lierler and Schiller 2012; Schüller 2013) use Planning and the CYK algorithm (Kasami 1965), Answer Set Programming (ASP) (Gelfond 1988; Baral 2003), and Combinatory Categorial Grammar (CCG) (Steedman 2000) for parsing NL. Our work uses a smaller set of combinatorial rules than (Schüller 2013), and requires the introduction of heuristics for action selection for efficient parsing, but allows parsing sentences into semantic representations as (Blackburn and Bos 2005; Bernardi, Calvanese, and Thorne 2007; Steels 2011) do, and addresses the problem of NL generation as (Steele 2011) does.

(Reib 2016) uses the same CG-based formalism for planning and plan recognition (as we do for NL generation and parsing), and specifies action preconditions, effects, causally prior tasks and causally subsequent tasks in CCG.

(Koller and Petrick 2011) translate the sentence generation problem into a planning problem, using tree-joining grammars (Joshi and Schabes 1997), and defining actions as operations that add a single elementary tree to the derivation.

---

Application Examples

Although a pattern matching approach can be used to map formulas in DL-Lite$^R_\bot$ into English sentences, and backwards, such an approach does not build semantic and syntactic representations for each component of a formula or of an English sentence, nor can it exploit domain dependent or contextual knowledge to fill in missing information, learn associations between expressions and meanings in context, or extend a robot’s lexicon using word formation mechanisms. The following examples illustrate the formalization proposed allows implementing such abilities.

Consider a scenario where a human-robot team is trying to build a map of a damaged area after an earthquake. Suppose a human team member asks a supervisor robot: **Does every robot examine a region?** But the robot does not understand the word “examine”.

If the robot had a lexical entry for “examine”, it could apply heuristic 5 for parsing, adding the construction $C_\alpha(C_{region}) = (("\text{examine}\_\text{a}\_\text{region}\_\text{np}\{s_2\}), \lambda x_0 \exists y (\lambda R(x_0, y) \land region(y)))$ to its current state. The problem is that lexical entry “examine” $\Rightarrow \lambda y.\lambda R(x_0, y) : (np\{s_3\})/np$ does not belong to the robot’s lexicon. However, the position of expression “examine” in the sentence allows inferring its syntactic category, and therefore the $\lambda$-Calculus type of the $\lambda$-term that could constitute its meaning. Using such syntactic and semantic information, the heuristic planner described in this paper could hypothesize the existence of lexical entry “examine” $\Rightarrow \lambda y.\lambda x.\lambda R(x, y) : (np\{s_3\})/np$ where variable $R$, ranging over binary predicate symbols, represents the indeterminate part of its semantic component.

Using the hypothesized lexical entry, the planner could apply rule 5, and add the following construction to its current state $\forall x (\lambda R(x_0, y) \land region(y))$. Afterwards, the planner would apply rule 2 for parsing “every+noun” (case $m = 0$), add construction $C_{every}(C_{robot}) = (("\forall\_\text{robot}\_\text{np}\{s_2\}), \lambda x_1 \forall x_2 (\lambda R(x_2) \rightarrow X_1(x_2)))$, use termination rule “(Every robot examines a region, stb), $\forall x (\lambda R(x_0, y) \land region(y))$” and parse the question.

Finally, the robot could unify the semantic part of this construction and formula $Formula \forall x (\lambda R(x_0, y) \rightarrow \exists y (\lambda R(x, y) \land region(y)))$, obtained by sensing the current situation; guess the sentence’s meaning should be $F$, and add lexical entry “examine” $\Rightarrow \lambda y.\lambda x.\lambda R(x, y) : (np\{s_3\})/np$ to its lexicon through this situated word learning process.

Suppose a human says “Is Robot_5 approachable from where you are?” to Robot_2, who has lexical entries “read” $\Rightarrow \lambda y.\lambda x.\lambda R(x, y) : (np\{s_3\})/np$ “readable” $\Rightarrow \lambda x.\lambda y.\lambda R(x, y) : (np\{s_3\})/np$ “approach” $\Rightarrow \lambda y.\lambda x.\lambda R(x, y) : (np\{s_3\})/np$.

Robot_2 could build lexical entry “approachable” $\Rightarrow \lambda y.\lambda x.\lambda R(x, y) : (np\{s_3\})/np$ from that for “readable”, applying the word formation scheme $\lambda x.\lambda y.\lambda R(x, y) : (np\{s_3\})/np$ to Stem “approach”. This example illustrates how a robot could formalise word formation mechanisms, such as derivation or compounding.

---

\(^6\)If $B_0(x) \neq \text{false}$, $m = 0$, $\text{good}\_\text{add}\_\text{construction}(C_{every}, C_t(s))$.

\(^7\)Category $q$ for question, lex entry $\text{Does} \Rightarrow \lambda U.\lambda x.\lambda y.\lambda U(x) : q/\text{stb}$.

\(^8\)Category $pn$ for proper noun, and Robot_5 $\Rightarrow$ robot_5 : pn.
References


