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$_{R,\sqcap}$ 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$_{\{R,\sqcap\}}$ formulas into English sentences and backwards, and examples of situated word learning and word formation mechanisms in a HRI scenario.

*Partially supported by MCIN/AEI/10.13039/501100011033 under grant PID2020-112581GB-C21.

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. Var_τ: a countably infinite set of variables of type τ.
2. Con_τ: a collection of constants of type τ.
The collections Term_τ of **λ-terms** of type τ are defined as the smallest sets such that: 1) $\mathrm{Var}_\tau \subseteq \mathrm{Term}_\tau$; 2) $\mathrm{Con}_\tau \subseteq \mathrm{Term}_\tau$; 3) $\alpha(\beta) \in \mathrm{Term}_\tau$ if $\alpha \in \mathrm{Term}_{\sigma\to\tau}$ and $\beta \in \mathrm{Term}_\sigma$; 4) $\lambda x.\alpha \in \mathrm{Term}_\tau$ if $\tau = \sigma \to \rho,\ x \in \mathrm{Var}_\sigma$ and $\alpha \in \mathrm{Term}_\rho$

A term of the form $\alpha(\beta)$ is said to be a *functional application*, and one of the form $\lambda x.\alpha$ 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 $Type(n) = Ind \to Bool$. *BasCat* is used to generate an infinite set **Cat** of functional categories that is the smallest set such that: 1) $BasCat \subseteq Cat$; 2) $(A\backslash B) \in Cat$ if $A, B \in Cat$; 3) $(B/A) \in Cat$ 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 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* $\rightarrow$ *λ-Calculus Types* for *Cat* is $Type(A\backslash B) = Type(B/A) = Type(A) \rightarrow Type(B)$

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* $\text{Lex} \subseteq \text{BasExp} \times \textit{Term} \times \text{Cat}$ *such that if a lexical entry* $e \Rightarrow \alpha : A \in \text{Lex}$*, then* $\alpha \in \textit{Term}_{\text{Type}(A)}$.

Definition 2 *(*$\mathbf{PS_{ACG}}$*) The* phrase-structure rules *of* **ACG** *are all instances of the following* **application schemes.**

$\alpha : B/A, e_1,\ \beta : A, e_2 \Rightarrow \alpha(\beta) : B, e_1 \cdot e_2$ *Forward Application*

$\alpha : A, e_1,\ \beta : A\backslash B, e_2 \Rightarrow \beta(\alpha) : B, e_1 \cdot e_2$ *Backward Application*

The forward scheme states that if e_1 is an expression of category B/A with meaning α, and e_2 is an expression of category A with meaning β, then expression $e_1 \cdot e_2$ (e_1 concatenated to e_2) is an expression of category B with meaning $\alpha(\beta)$. 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 successors. Let $\langle \alpha : C, e, \langle T_1, T_2 \rangle \rangle$ denote the tree *rooted* at category $\alpha : C, e$ with *daughter* trees T_1, T_2 (Carpenter 1997).

Definition 3 *(*$\mathbf{AdmTree_{Lex}}$*) The set of* **admissible trees** *associated with a lexicon* **Lex** *is the least set such that:*

1. $\langle \alpha : C, e \rangle \in \mathbf{AdmTree}_{Lex}$ **if** $e \Rightarrow \alpha : C \in \mathbf{Lex}$

2. $\langle \alpha(\beta) : C, e, \langle T_1, T_2 \rangle \rangle \in \mathbf{AdmTree}$ **if** $T_1, T_2 \in \mathbf{AdmTree}$ *and* $Root(T_1), Root(T_2) \Rightarrow \alpha(\beta) : C, e \in \mathbf{PS_{ACG}}$.

The **language recognized** *by lexicon* **Lex** *is the expression set* $\mathbf{L_{Lex}} = \{e \mid T \in AdmTree_{Lex} \text{ and } Root(T) = (\alpha : C, e)\}$.

Parsing and Generation as Planning Problems

This section proposes a formulation of *generation* (i.e. mapping λ-terms 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)* $s_0 \in S$ *is the* known initial state*; 3)* $S_G \subseteq S$ *is the non-empty set of* goal states*; 4)* $A(s) \subseteq A$ *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*.*

A solution to P *is an action sequence* $a_0, \ldots, a_n$ *that generates a state sequence* $s_0, \ldots, s_{n+1}$ *where* $s_{n+1} \in S_G$ *is a goal state, each* $a_i \in A(s_i)$ *is applicable in state* s_i*, and each* $s_{i+1} = f(a, s_i)$ *follows* s_i *(Geffner and Bonet 2013).*

The **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 **parsing and generation share the same domain model**.

Definition 5 *Given a lexicon* **Lex***, a* **construction** *is a pair* $((e, B), \alpha)$ *such that there is a tree* $T \in \mathbf{AdmTree_{Lex}}$ *with* $Root(T) = (\alpha : B, e)$*. 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 formalized in *Fluid Construction Grammar* (Steels 2011) using feature structure *unification* and *merge* for parsing and generation 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** $\mathbf{s_0}$ *is the empty set of constructions. The* **set of goal states** $\mathbf{S_G}$ *is I)* $\mathbf{S_G} = \{\Psi \mid ((e, B), \beta) \in \Psi\}$ *the set of states which contain a construction* $((e, B), \beta)$ *for* **parsing problem (e, Lex)***; and II)* $\mathbf{S_G} = \{\Psi \mid ((e', B), \alpha) \in \Psi\}$ *the set of states which contain a construction* $((e', B), \alpha)$ *for* **generation problem (**α**,Lex)**. *The* **available actions** *set* $\mathbf{A} = \{add\_construction(C_1, C_2) \mid$ *each* $C_{i=1,2}$ *is a construction*$\}$. $\mathbf{A(s)}$*, the set of* **applicable actions***, and* $\mathbf{f(a, s)}$*, the* **transition function***, are defined by specifying the* preconditions *and* effects *of each action*[1].

Preconditions: add_construction(C_1, C_2) *can be executed in state s if* $C_1, C_2 \in \Psi \bigcup Lex$*, and if* C_1 *can be applied to* C_2 *using the application schemes of* **ACG** *(see below).*

Effects: *the result of executing* add_construction(C_1, C_2) *in s is* $s' = \Psi \bigcup \{C_1(C_2)\}$. $C_1(C_2)$ *denotes the result of applying construction* C_1 *to construction* C_2*, and equals*

- $((e_1 \cdot e_2, B), \beta(\alpha))$*, if* C_1 *is of the form* $((e_1, B/A), \beta)$ *and* C_2 *is of the form* $((e_2, A), \alpha)$*, (forward app.); or*
- $((e_2 \cdot e_1, B), \beta(\alpha))$*, if* C_1 *is of the form* $((e_1, A\backslash B), \beta)$ *and* C_2 *is of the form* $((e_2, A), \alpha)$*, (backward app.).*

Note the domain model (states and actions) of the planning problems (α, Lex) *generation* and (e, Lex) *parsing* is determined 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. executable) 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 declaratively is that it is possible to refine the action selection strategy of the planner/robot by simple additions of better heuristics, as proposed in *the advice taker* (McCarthy 1959).

An *action selection strategy* Σ is a set of action selection rules. An *action selection rule* is an implication whose antecedent depends on a state s, and whose consequent takes the following forms: $Good(a, s)$, $Bad(a, s)$ or $Better(a, b, s)$[2]. The heuristic planner determines the set of selectable actions for a state s by interpreting Σ nonmonotonically (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 s, and a is *not bad* for s. Next, we present an action selection strategy that allows a heuristic planner to map DL-Lite$_{R,\sqcap}$ formulas into English sentences, and a second strategy it can use for parsing them.

[1]See next sections for examples of constructions, actions, and their application to parse/generate sentences in a HRI scenario.

[2]The 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-Lite$_{\mathbf{R},\sqcap}$

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-Lite$_{\mathbf{R},\sqcap}$ is a member of DL-Lite in which the *Terminological Knowledge Base* (**TBox**) consists of **inclusion assertions** of the form $Cl \sqsubseteq Cr$. Cl and Cr denote concepts that may occur on the left and right-hand side, and can be of the form:

$Cl \quad \longrightarrow \quad B \mid \exists R \mid Cl_1 \sqcap Cl_2$
$Cr \quad \longrightarrow \quad B \mid \neg B \mid \exists R \mid \neg\exists R \mid \exists R.B$

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

Inclusion Assertions in DL-Lite$_{R,\sqcap}$ can be translated into FOL formulas of the form $\forall x(Cl(x) \rightarrow Cr(x))$ such that[3]

$Cl(x) \quad \longrightarrow \quad B_0(x) \wedge \bigwedge_{i=1}^{m} A_i(x)$
$Cr(x) \quad \longrightarrow \quad B(x) \mid \neg B(x) \mid \exists y R(x,y) \mid \neg\exists y R(x,y) \mid \exists y(R(x,y) \wedge B(y))$

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 Lex$_{\mathbf{R},\sqcap}$

every	$\Rightarrow$	$\lambda U.\lambda V.\forall x.(U(x) \rightarrow V(x))$: $(stb/(np\backslash scr))/ncl$
everyone	$\Rightarrow$	$\lambda U.\lambda V.\forall x.(U(x) \rightarrow V(x))$: $(stb/(np\backslash scr))/(np\backslash swho)$
who$_1$	$\Rightarrow$	$\lambda U.\lambda x.U(x) : (np\backslash swho)/(np\backslash scl)$
is a	$\Rightarrow$	$\lambda U.\lambda x.U(x) : (np\backslash s)/n$
is not a	$\Rightarrow$	$\lambda U.\lambda x.\neg U(x) : (np\backslash scr)/n$
does not	$\Rightarrow$	$\lambda U.\lambda x.\neg U(x) : (np\backslash scr)/(np\backslash s)$
robot	$\Rightarrow$	$\lambda x.robot(x) : n$
explores	$\Rightarrow$	$\lambda x.explore(x) : np\backslash s$
scans	$\Rightarrow$	$\lambda y.\lambda x.scan(x,y) : (np\backslash s_\exists)/np$
something	$\Rightarrow$	$\lambda U.\lambda y.\exists x.U(y,x)$: $((np\backslash s_\exists)/np)\backslash(np\backslash s)$
a	$\Rightarrow$	$\lambda U.\lambda V.\lambda y.\exists x.(V(y,x) \wedge U(x))$: $(((np\backslash s_\exists)/np)\backslash(np\backslash scr))/n$
who$_2$	$\Rightarrow$	$\lambda U.\lambda V.\lambda x.(U(x) \wedge V(x))$: $(ncl\backslash ncl)/(np\backslash scl)$
and	$\Rightarrow$	$\lambda U.\lambda V.\lambda x.(U(x) \wedge V(x))$: $((np\backslash scl)\backslash(np\backslash scl))/(np\backslash scl)$

(Bernardi, Calvanese, and Thorne 2007) define a CG lexicon that captures a fragment of English consisting of sentences whose meanings belong to DL-Lite$_{R,\sqcap}$. 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 Lex$_{\mathbf{R},\sqcap}$** above[4]), 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 $\mathbf{Cr(x)}$ of $\forall\mathbf{x}(\mathbf{Cl(x)} \rightarrow \mathbf{Cr(x)})$ Given a basic expression e of a lexicon *Lex*, C_e denotes the construction associated with lexical entry $e \Rightarrow \alpha : A \in Lex$[5].

1. If $Cr(x)$ is of the form $B(x)$ and there is a lexical entry $e \Rightarrow \lambda x B(x) : A \in Lex$ such that A is
   (a) n (noun), then *good(add_construction(C_r,$C_{is\ a}$(C_e)),s)*
   (b) $np\backslash s$, intrans verb, *good(add_construction(C_r,C_e),s)*
2. If $Cr(x)$ is of the form $\neg B(x)$ and there is a lexical entry $e \Rightarrow \lambda x B(x) : A \in Lex$ such that A is
   (a) n (noun), then *good(add_construction($C_{is\ not\ a}$,C_e),s)*
   (b) $np\backslash s$, then *good(add_construction($C_{does\ not}$,C_e),s)*
3. If $Cr(x)$ is of the form $\exists y R(x,y)$ and $e \Rightarrow \lambda y \lambda x R(x,y) : (np\backslash s_\exists)/np \in Lex$ (transitive verb), then *good(add_construction(C_r,$C_{something}$(C_e)),s)*
4. If $Cr(x)$ is of the form $\neg\exists y R(x,y)$ and $e \Rightarrow \lambda y \lambda x R(x,y) : (np\backslash s_\exists)/np \in Lex$ (transitive verb), then *good(add_construction($C_{does\ not}$,$C_{something}$(C_e)),s)*
5. If $Cr(x)$ is of the form $\exists y(R(x,y) \wedge B(y))$, $e_1 \Rightarrow \lambda y \lambda x R(x,y) : (np\backslash s_\exists)/np \in Lex$ and $e_2 \Rightarrow \lambda x B(x) : n \in Lex$, then *good(add_construction(C_a(C_{e_2}),C_{e_1}),s)*

Heuristics for Expressing $\mathbf{Cl(x)}$ of $\forall\mathbf{x}(\mathbf{Cl(x)} \rightarrow \mathbf{Cr(x)})$ 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 $e \Rightarrow \lambda x B(x) : A \in Lex$ such that A is
   (a) *n*, $B_0(x) \not\equiv true$, *good(add_construction(C_s,$C_{is\ a}$(C_e)),s)*
   (b) $np\backslash s$ (intr. verb), *good(add_construction(C_s,C_e),s)*
2. If $A_1(x)$ is of the form $\exists y R(x,y)$, and there is a lexical entry $e \Rightarrow \lambda y \lambda x R(x,y) : (np\backslash s_\exists)/np \in Lex$, then *good(add_construction(C_s,$C_{something}$(C_e)),s)*

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

1. If there is a subformula $H(x) \wedge D(x)$ of $\bigwedge_{i=1}^{m} A_i(x)$ such that the current state s contains construction C_H=$((e_H, np\backslash scl)), \lambda x H(x))$ and lexical entry
   (a) $e_i \Rightarrow \lambda x D(x) : n \in Lex$, then *good(add_construction(C_{and}(C_s($C_{is\ a}$(C_{e_i}))),C_H),s)*
   (b) $e_i \Rightarrow \lambda x D(x) : np\backslash s \in Lex$, then *good(add_construction(C_{and}(C_s(C_{e_i})),C_H),s)*
2. If there is a subformula $H(x) \wedge \exists y R(x,y)$ of $\bigwedge_{i=1}^{n} A_i(x)$ such that the current state s contains a construction $C_H = ((e_H, np\backslash scl)), \lambda x H(x))$ and there is a lexical entry $e_i \Rightarrow \lambda y \lambda x R(x,y) : (np\backslash s_\exists)/np \in Lex$, then *good(add_construction(C_{and}(C_s($C_{something}$(C_{e_i}))),C_H),s)*

These rules add *"every + noun + who"* or *"everyone who"* to a conjunction of clauses.

1. If $Cl(x)$ is of the form $\bigwedge_{i=1}^{m} A_i(x)$ with $m > 0$, there is no lexical entry of the form $e \Rightarrow \lambda x A_i(x) : n \in Lex$ for $i = 1, \ldots, m$ (i.e. $B_0(x) \equiv true$), and the current state s contains a construction $C_G = ((e_G, np\backslash scl, \lambda x(\bigwedge_{i=1}^{m} A_i(x)))$, then *good(add_construction($C_{everyone}$, C_{who_1}(C_G)),s)*

---

[3] We assume without loss of generality that $B_0(x)$ is one of the conjuncts in $Cl(x)$ that can be expressed by a noun in the lexicon, if there is any; otherwise, $B_0(x) \equiv$ *true* and $Cl(x) \equiv \bigwedge_{i=1}^{n} A_i(x)$.

[4] The λ-Calculus type of subcategories *stb, scr, scl, swho,* $s_\exists$ of *s* is *Bool*; and that of subcategory *ncl* of *n* is *Ind*$\rightarrow$*Bool*. The following lexical entries implement the inference rules $(np\backslash s) \Rightarrow (np\backslash scr)$, $(np\backslash s) \Rightarrow (np\backslash scl)$, and $n \Rightarrow ncl$.

$$C_r : \quad \emptyset \quad \Rightarrow \quad \lambda U \lambda x U(x) : (np\backslash scr)/(np\backslash s)$$
$$C_s : \quad \emptyset \quad \Rightarrow \quad \lambda U \lambda x U(x) : (np\backslash scl)/(np\backslash s)$$
$$C_l : \quad \emptyset \quad \Rightarrow \quad \lambda U \lambda x U(x) : ncl/n$$

[5] We assume all rules of the form *Condition(s)*$\rightarrow$ *good(add_construction(C_1,C_2),s)* satisfy the condition $C_1(C_2)) \notin$s.

2. If $Cl(x)$ is of the form $B_0(x) \wedge \bigwedge_{i=1}^{m} A_i(x)$ 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 $C_G = ((e_G, np\backslash scl), \lambda x(\bigwedge_{i=1}^{m} A_i(x)))$, *good(add_construction(C_{every},(C_{who_2}(C_G))(C_l(C_e))),s)*

Termination Heuristic Given a language generation problem (α, Lex) with α of the form $\forall x(Cl(x) \rightarrow Cr(x))$, if the current state s contains a construction $C_G = ((e_G, stb/(np\backslash scr)), \lambda X_1 \forall x_2(Cl(x_2) \rightarrow X_1(x_2)))$ and another construction $C_H = ((e_H, np\backslash scr), \lambda x_3 Cr(x_3))$, then *good(add_construction(C_G,C_H),s)*.

Parsing Sentences in the English Fragment $\mathbf{L_{Lex_{R,\sqcap}}}$

The following heuristics allow a heuristic planner to map English sentences in $L_{Lex_{R,\sqcap}}$, the language recognized by lexicon $Lex_{R,\sqcap}$, into DL-Lite$_{R,\sqcap}$ 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 $\mathbf{e}$, 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 $e_1 \Rightarrow \lambda y \lambda x R(x,y) : (np\backslash s_{\exists})/np$ and $e_2 \Rightarrow \lambda x B(x) : n$ in *Lex*, and there is an expression e_h such that $e = e_h \cdot e_1 \cdot$ "a" $\cdot\, e_2$, then *good(add_construction(C_a(C_{e_2}),C_{e_1}),s)*.

Termination Heuristic Given parsing problem $(\mathbf{e}, Lex)$, if state s contains two constructions of the form $C_{cl} \equiv ((e_{cl}, stb/(np\backslash scr)), \beta)$ and $C_{cr} \equiv ((e_{cr}, np\backslash scr), \alpha)$, and $e_{cl} \cdot e_{cr} = \mathbf{e}$, then *good(add_construction(C_{cl},C_{cr}),s)*.

Related Work

(Lierler and Schüller 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 combinatory 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 (Steels 2011) does.

(Geib 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 *the sentence generation problem* into a planning problem, using tree-adjoining 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$_{R,\sqcap}$ 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 how 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 $(C_a(C_{region}))(C_{examine}) =$ (("examine a region", $np\backslash scr$), $\lambda x_0.\exists y.(examine(x_0,y) \wedge region(y))$) to its current state. The problem is that lexical entry "examine" $\Rightarrow \lambda y.\lambda x.examine(x,y) : (np\backslash s_{\exists})/np$ 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 λ-Calculus type of the λ-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" $\Rightarrow \lambda y.\lambda x.\lambda R.R(x,y) : (np\backslash s_{\exists})/np$ 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 (("examine a region", $np\backslash scr$), $\lambda x_0 \exists y(\lambda R.R(x_0,y) \wedge region(y))$). Afterwards, the planner would apply rule 2 for parsing *"every+noun"* (case $m = 0$), add construction $C_{every}(C_l(C_{robot})) =$ (("Every robot", $stb/(np\backslash scr)$), $\lambda X_1.\forall x_2.(robot(x_2) \rightarrow X_1(x_2))$), use termination rule (("Every robot examines a region", stb), $\forall z(robot(z) \rightarrow \exists y(\lambda R.R(z,y) \wedge region(y)))$) and parse the question[7].

Finally, the robot could unify the semantic part of this construction and formula $F \equiv \forall x(robot(x) \rightarrow \exists y(scan(x,y) \wedge region(y)))$, obtained by sensing the current situation; guess the sentence's meaning should be F, and add lexical entry "examines"$\Rightarrow \lambda y.\lambda x.scan(x,y) : (np\backslash s_{\exists})/np$ 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"$\Rightarrow \lambda y.\lambda x.read(x,y) : (np\backslash s_{\exists})/np$
"readable"$\Rightarrow \lambda x.\exists y.read(y,x) : pn\backslash s$
"approach"$\Rightarrow \lambda y.\lambda x.approach(x,y) : (np\backslash s_{\exists})/np$

Robot_2 could build lexical entry "approachable" $\Rightarrow \lambda x.\exists y.approach(y,x) : pn\backslash s$ from that for "readable", applying the word formation scheme $\lambda Stem(Stem \cdot$"able"$\Rightarrow \lambda x.\exists y.Stem(y,x)) : pn\backslash s)$ to $Stem$ *"approach"*[8].
This example illustrates how a robot could formalise word formation mechanisms, such as *derivation* or *compounding*.

[6] If $B_0(x) \not\equiv$ true, m=0, *good(add_construction(C_{every},C_l(C_e)),s)*
.

[7] Category q for question, lex entry $\mathbf{Does} \Rightarrow \lambda U \lambda x U(x) : q/stb$.

[8] Category pn for proper noun, and $\mathbf{Robot\_5} \Rightarrow robot\_5 : pn$.

References

Ajdukiewicz, K. 1935. Die Syntaktische Konnexitat. *Studia Philosophica*, 1: 1–27.

Baader, F.; Calvanese, D.; McGuinness, D.; Nardi, D.; and Patel-Schneider, P. F., eds. 2003. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

Bar-Hillel, Y. 1950. On Syntactical Categories. *Journal of Symbolic Logic*, 15: 1–16.

Baral, C. 2003. *Knowledge Representation, Reasoning, and Declarative Problem Solving*. Cambridge University Press.

Bernardi, R.; Calvanese, D.; and Thorne, C. 2007. Lite Natural Language. In *Proc. of the 7th Int. Workshop on Computational Semantics (IWCS 2007)*.

Blackburn, P.; and Bos, J. 2005. *Representation and Inference for Natural Language. A First Course in Computational Semantics*. CSLI Publications.

Calvanese, D.; De Giacomo, G.; Lembo, D.; Lenzerini, M.; Poggi, A.; Rodriguez-Muro, M.; Rosati, R.; Ruzzi, M.; and Savo, D. F. 2011. The MASTRO system for ontology-based data access. *Semantic Web*, 2: 43–53.

Calvanese, D.; De Giacomo, G.; Lembo, D.; Lenzerini, M.; and Rosati, R. 2007. Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family. *Journal of Automated Reasoning*, 39: 385–429.

Carpenter, B. 1997. *Type-Logical Semantics*. MIT Press.

Church, A. 1940. A formulation of a simple theory of types. *Journal of Symbolic Logic*, 5: 56–68.

Geffner, H.; and Bonet, B. 2013. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan and Claypool Publishers.

Geib, C. 2016. Lexicalized Reasoning about Actions. *Advances in Cognitive Systems (ACS)*, 4: 187–206.

Gelfond, L.-V., M. 1988. The stable model semantics for logic programming. In *International Logic Programming Conference and Symposium (ICLP)*, 1070–1080.

Goldberg, A. E. 1995. *Constructions: A Construction Grammar Approach to Argument Structure*. Chicago : University of Chicago Press. ISBN 978-0-226-30086-3. Originally presented as the author's thesis (Ph.D.)–University of California, Berkeley, 1992.

Joshi, A.; and Schabes, Y. 1997. *Tree-Adjoining Grammars*.

Kasami, T. 1965. An efficient recognition and syntax analysis algorithm for context-free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Laboratory.

Koller, A.; and Petrick, R. 2011. Experiences with planning for natural language generation. *Computational Intelligence*, (27): 23–40.

Lambek, J. 1958. The mathematics of sentence structure. *American Mathematical Monthly*, (65): 154–169.

Lambek, J. 1961. On the calculus of syntactic types. In *Structure of Language and its Mathematical Aspects: Proceedings of Symposia in Applied Mathematics*, 166–178. American Mathematical Society.

Lierler, Y.; and Schüller, P. 2012. *Parsing Combinatory Categorial Grammar via planning in Answer Set Programming*, volume 7265 of *LNCS*, 436–453. Springer.

Lifschitz, V. 1995. Nested abnormality theories. *Artificial Intelligence*, 74: 1262–1277.

McCarthy, J. 1959. Programs with common sense. In *In Mechanization of Thought Processes, Proceedings of the Symposium of the National Physics Laboratory*, 77–84.

Schüller, P. 2013. Flexible combinatory categorial grammar parsing using the CYK algorithm and Answer Set Programming. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, 499–511. Springer Berlin Heidelberg.

Sierra-Santibanez, J. 1998. Declarative formalization of strategies for action selection. In *Proceedings of the Seventh International Workshop on Non-monotonic Reasoning*.

Steedman, M. 2000. *The syntactic process*. MIT Press.

Steels, L. 2011. Introducing Fluid Construction Grammar. In Steels, L., ed., *Design Patterns in Fluid Construction Grammar*, 3–30. Amsterdam: John Benjamins.

van Benthem, J. 1987. *Categorial Grammar and Lambda Calculus*, 39–60. Boston, MA: Springer US. ISBN 978-1-4613-0897-3.