Generating effective referring expressions using charts

Nikos Engonopoulos and **Alexander Koller**
University of Potsdam, Germany
{engonopo|akoller}@uni-potsdam.de

Abstract

We present a novel approach for generating effective referring expressions (REs). We define a synchronous grammar formalism that relates surface strings with the sets of objects they describe through an abstract syntactic structure. The grammars may choose to require or not that REs are distinguishing. We then show how to compute a chart that represents, in finite space, the complete (possibly infinite) set of valid REs for a target object. Finally, we propose a probability model that predicts how the listener will understand the RE, and show how to compute the most effective RE according to this model from the chart.

1 Introduction

The fundamental challenge in the generation of referring expressions (REG) is to compute an RE which is effective, i.e. understood as intended by the listener. Throughout the history of REG, we have approximated this as the problem of generating *distinguishing* REs, i.e. REs that are only satisfied by a unique individual in the domain. This has been an eminently successful approach, as documented e.g. in the overview article of Krahmer and van Deemter (2012) and a variety of recent shared tasks involving RE generation (Gatt and Belz, 2010; Belz et al., 2008; Koller et al., 2010).

Nonetheless, reducing effectiveness to uniqueness is limiting in several ways. First, in complex, real-world scenes it may not be feasible to generate fully distinguishing REs, or these may have to be exceedingly complicated. It is also not necessary to generate distinguishing REs in such situations, because listeners are very capable of taking the discourse and task context into account to resolve even ambiguous REs. Conversely, listeners can misunderstand even a distinguishing RE, so uniqueness is no guarantee for success. We propose instead to define and train a probabilistic RE resolution model $P(a|t)$, which directly captures the probability that the listener will resolve a given RE t to some object a in the domain. An RE t will then be "good enough" if $P(a^*|t)$ is very high for the intended target referent a^*.

Second, in an interactive setting like the GIVE Challenge (Koller et al., 2010), the listener may behave in a way that offers further information on how they resolved the generated RE. Engonopoulos et al. (2013) showed how an initial estimate of the distribution $P(a|t)$ can be continuously updated based on the listener's behavior, and that this can improve a system's ability to detect misunderstandings. It seems hard to achieve this in a principled way without an explicit model of $P(a|t)$.

In this paper, we present an algorithm that generates the RE t that maximizes $P(a^*|t)$, i.e. the RE that has the highest chance to be understood correctly by the listener according to the probabilistic RE resolution model. This is a challenging problem, since the algorithm must identify that RE from a potentially infinite set of valid alternatives. We achieve this by using a *chart-based* algorithm, a standard approach in parsing and realization, which has (to our knowledge) never been used in REG.

We start by defining a synchronous grammar formalism that relates surface strings to their interpretations as sets of objects in a given domain (Section 3). This formalism integrates REG with surface realization, and allows us to specify in the grammar whether REs are required to be distinguishing. We then show how to compute a chart for a given grammar and target referent in Section 4. Section 5 defines a log-linear model for $P(a|t)$, and presents a Viterbi-style algorithm for computing the RE t from the chart that maximizes $P(a^*|t)$. Section 6 concludes by discussing how to apply our algorithm to the state-of-the-art approaches of Krahmer et al. (2003) and Golland et al. (2010), and how to address a particular challenge involving cycles that arises when dealing

Proceedings of the INLG and SIGDIAL 2014 Joint Session, pages 162–171,
Philadelphia, Pennsylvania, 19 June 2014.

with probabilistic listener models.

2 Related Work

RE generation is the task of generating a natural-language expression that identifies an object to the listener. Since the beginnings of modern REG (Appelt, 1985; Dale and Reiter, 1995), this problem has been approximated as generating a *distinguishing* description, i.e. one which fits only one object in the domain and not any of the others. This perspective has made it possible to apply search-based (Kelleher and Kruijff, 2006), logic-based (Areces et al., 2008) and graph-based (Krahmer et al., 2003) methods to the problem, and overall has been one of the success stories of NLG.

However, in practice, human speakers frequently *overspecify*, i.e. they include information in an RE beyond what is necessary to make it distinguishing (Wardlow Lane and Ferreira, 2008; Koolen et al., 2011). An NLG system, too, might include redundant information in an RE to make it easier to understand for the user. Conversely, an RE that is produced by a human can often be easily resolved by the listener even if it is ambiguous. Here we present an NLG system that directly uses a probabilistic model of RE resolution, and is capable of generating ambiguous REs if it predicts that the listener will understand them.

Most existing REG algorithms focus on generating distinguishing REs, and then select the one that is *best* according to some criterion, e.g. most human-like (Krahmer et al., 2003; FitzGerald et al., 2013) or most likely to be understood (Garoufi and Koller, 2013). By contrast, Mitchell et al. (2013) describe a stochastic algorithm that computes human-like, non-relational REs that may not be distinguishing. Golland et al. (2010) are close to our proposal in spirit, in that they use a log-linear probability model of RE resolution to compute a possibly non-distinguishing RE. However, they use a trivial REG algorithm which is limited to grammars that only permit a (small) finite set of REs for each referent. This is in contrast to general REG, where there is typically an infinite set of valid REs, especially when relational REs ("the button *to the left of* the plant") are permitted.

Engonopoulos et al. (2013) describe how to update an estimate for $P(a|t)$ based on a log-linear model based on observations of the listener's behavior. They use a shallow model based on a string t and not an RE derived from a grammar, and they do not discuss how to generate the best t. The algorithm we develop here fills this gap.

Our formalism for REG can be seen as a *synchronous* grammar formalism; it simultaneously derives strings and their interpretations, connecting the two by an abstract syntactic representation. This allows performing REG and surface realization with a single algorithm, along the lines of SPUD (Stone et al., 2003) and its planning-based implementation, CRISP (Koller and Stone, 2007). Probabilistic synchronous grammars are widely used in statistical machine translation (Chiang, 2007; Graehl et al., 2008; Jones et al., 2012) and semantic parsing (Zettlemoyer and Collins, 2005; Wong and Mooney, 2007). Lu and Ng (2011) have applied such grammars to surface realization. Konstas and Lapata (2012) use related techniques for content selection and surface realization (with simple, non-recursive grammars).

Charts are standard tools for representing a large space of possible linguistic analyses compactly. Next to their use in parsing, they have also been applied to surface realization (Kay, 1996; Carroll et al., 1999; Kaplan and Wedekind, 2000). To our knowledge, ours is the first work using charts for REG. This is challenging because the input to REG is much less structured than in parsing or realization.

3 Grammars for RE generation

We define a new grammar formalism that we use for REG, which we call *semantically intepreted grammar* (SIG). SIG is a synchronous grammar formalism that relates natural language strings with the sets of objects in a given domain which they describe. It uses *regular tree grammars (RTGs)* to describe languages of *derivation trees*, which then project to strings and sets.

3.1 Derivation trees

We describe the abstract syntax of an RE by its *derivation tree*, which is a tree over some ranked signature Σ of symbols representing lexicon entries and grammatical constructions. A *(ranked) signature* is a finite set of symbols $r \in \Sigma$, each of which is assigned an *arity* $\mathsf{ar}(r) \in \mathbb{N}_0$. A tree over the signature Σ is a term $r(t_1, \ldots, t_n)$, where $r \in \Sigma$, $n = \mathsf{ar}(r)$, and $t_1, \ldots, t_n$ are trees over Σ. We write T_Σ for the set of all trees over Σ.

Fig. 1b shows an example derivation tree for the RE "the square button" over the signature $\Sigma = \{def|_1, square|_1, button|_0\}$, where $r|_n$ indicates that the symbol r has arity n. In term nota-

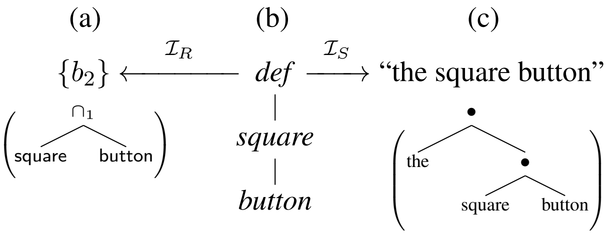


Figure 1: A SIG derivation tree (b) with its interpretations (a, c).

tion, it is *def*(*square*(*button*)).

String interpretation. We *interpret* derivation trees simultaneously as strings and sets. First, let Δ be a finite alphabet, and let Δ^* be the string algebra over Δ. We define a *string interpretation* over Δ as a function $\mathcal{I}_S$ that maps each $r|_n \in \Sigma$ to a function $\mathcal{I}_S(r) : (\Delta^*)^n \to \Delta^*$. For instance, we can assign string interpretations to our example signature Σ as follows; we write $w_1 \bullet w_2$ for the concatenation of the strings w_1 and w_2.

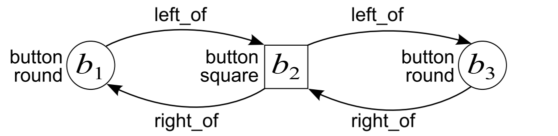
$$\begin{aligned} \mathcal{I}_S(\mathit{def})(w_1) &= \text{the} \bullet w_1 \\ \mathcal{I}_S(\mathit{square})(w_1) &= \text{square} \bullet w_1 \\ \mathcal{I}_S(\mathit{button}) &= \text{button} \end{aligned}$$

Since the arity of $\mathcal{I}_S(r)$ is the same as the arity of r for any $r \in \Sigma$, we can use $\mathcal{I}_S$ to recursively map derivation trees to strings. Starting at the leaves, we map the tree $r(t_1, \ldots, t_n)$ to the string $\mathcal{I}_S(r)(\mathcal{I}_S(t_1), \ldots, \mathcal{I}_S(t_n))$, where $\mathcal{I}_S(t_i)$ is the string that results from recursively applying $\mathcal{I}_S$ to the subtree t_i. In the example, the subtree *button* is mapped to the string "button". We then get the string for the subtree *square*(*button*) by concatenating this with "square", obtaining the string "square button" and so on, as shown in Fig. 1c.

Relational interpretation. We further define a *relational interpretation* $\mathcal{I}_R$, which maps each $r|_n \in \Sigma$ to a function $\mathcal{I}_R(r) : R(U)^n \to R(U)$, where $R(U)$ is a class of *relations*. We define $\mathcal{I}_R$ over some first-order *model* structure $M = \langle U, L\rangle$, where U is a finite *universe* U of individuals and L interprets a finite set of predicate symbols as relations over U. We let $R(U)$ be the set of all k-place relations over U for all $k \geq 0$. The subsets of U are the special case of $k = 1$. We write $k(R)$ for the arity of a relation $R \in R(U)$.

For the purposes of this paper, we construct $\mathcal{I}_R$ by combining the following operations:

- The denotations of the atomic predicate symbols of M; see Fig. 2 for an example.



$U = \{b_1, b_2, b_3\}$ $\quad$ $\mathsf{button} = \{b_1, b_2, b_3\}$
$\mathsf{round} = \{b_1, b_3\}$ $\quad$ $\mathsf{square} = \{b_2\}$
$\mathsf{left\_of} = \{\langle b_1, b_2\rangle, \langle b_2, b_3\rangle\}$
$\mathsf{right\_of} = \{\langle b_2, b_1\rangle, \langle b_3, b_2\rangle\}$

Figure 2: A simple model, illustrated as a graph.

- $\mathsf{proj}_i(R) = \{a_i \mid \langle a_1, \ldots, a_{k(R)}\rangle \in R\}$ is the projection to the i-th component; if $i > k(R)$, it evaluates to $\emptyset$.
- $R_1 \cap_i R_2 = \{\langle a_1, \ldots, a_{k(R_1)}\rangle \in R_1 \mid a_i \in R_2\}$ is the intersection on the i-th component of R_1; if $i > k(R_1)$, it evaluates to $\emptyset$.
- For any $a \in U$, $\mathsf{uniq}_a(R)$ evaluates to $\{a\}$ if $R = \{a\}$, and to $\emptyset$ otherwise.
- For any $a \in U$, $\mathsf{member}_a(R)$ evaluates to $\{a\}$ if $a \in R$, and to $\emptyset$ otherwise.

For the example, we assume that we want to generate REs over the scene shown in Fig. 2; it consists of the universe $U = \{b_1, b_2, b_3\}$ and interprets the atomic predicate symbols button, square, round, left_of, and right_of. Given this, we can assign a relational interpretation to the derivation tree in Fig. 1b using the following mappings:

$$\begin{aligned} \mathcal{I}_R(\mathit{def})(R_1) &= R_1 \\ \mathcal{I}_R(\mathit{square})(R_1) &= \mathsf{square} \cap_1 R_1 \\ \mathcal{I}_R(\mathit{button}) &= \mathsf{button} \end{aligned}$$

We evaluate a derivation tree to a relation as we did for strings (cf. Fig. 1a). The subtree *button* maps to the denotation of the symbol button, i.e. $\{b_1, b_2, b_3\}$. The subtree *square*(*button*) evaluates to the intersection of this set with the set of square individuals, i.e. $\{b_2\}$; this is also the relational interpretation of the entire derivation tree. We thus see that "the square button" is an RE that describes the individual b_2 uniquely.

3.2 Semantically interpreted grammars

Now we define grammars that describe relations between strings and relations over U. We achieve this by combining a *regular tree grammar* (RTG, (Gécseg and Steinby, 1997; Comon et al., 2007)), describing a language of derivation trees, with a string interpretation and a relational interpretation. An RTG $G = (N, \Sigma, S, P)$ consists of a finite set N of nonterminal symbols, a ranked signature Σ, a start symbol $S \in N$, and a finite set P of production rules $\mathrm{A} \to r(B_1, ..., B_n)$, where

$A, B_1, \ldots, B_n \in N$ and $r|_n \in \Sigma$. We say that a tree $t_2 \in T_\Sigma$ can be *derived in one step* from $t_1 \in T_\Sigma$, $t_1 \Rightarrow t_2$, if it can be obtained by replacing an occurrence of B in t_1 with t and P contains the rule $B \rightarrow t$. A tree $t_n \in T_\Sigma$ can be *derived* from t_1, $t_1 \Rightarrow^* t_n$, if there is a sequence $t_1 \Rightarrow \ldots \Rightarrow t_n$ of length $n \geq 0$. For any nonterminal A, we write $L_A(G)$ for the set of trees $t \in T_\Sigma$ with $A \Rightarrow^* t$. We simply write $L(G)$ for $L_S(G)$ and call it the *language* of G.

We define a *semantically interpreted grammar* (SIG) as a triple $\mathcal{G} = (G, \mathcal{I}_S, \mathcal{I}_R)$ of an RTG G over some signature Σ, together with a string interpretation $\mathcal{I}_S$ over some alphabet Δ and a relational interpretation $\mathcal{I}_R$ over some universe U, both of which interpret the symbols in Σ. We assume that every terminal symbol $r \in \Sigma$ occurs in at most one rule, and that the nonterminals of G are pairs A_b of a syntactic category A and a *semantic index* $b = \mathsf{ix}(A_b)$. A semantic index indicates the individual in U to which a given constituent is meant to refer, see e.g. (Kay, 1996; Stone et al., 2003). Note that SIGs can be seen as specific Interpreted Regular Tree Grammars (Koller and Kuhlmann, 2011) with a set and a string interpretation.

We ignore the start symbol of G. Instead, we say that given some individual $b \in U$ and syntactic category A, the set of *referring expressions* for b is $\mathsf{RE}_\mathcal{G}(A, b) = \{t \in L_{A_b}(G) \mid \mathcal{I}_R(t) = \{b\}\}$, i.e. we define an RE as a derivation tree that G can derive from A_b and whose relational interpretation is $\{b\}$. From t, we can read off the string $\mathcal{I}_S(t)$.[1]

3.3 An example grammar

Consider the SIG $\mathcal{G}$ in Fig. 3 for example. The grammar is written in template form. Each rule is instantiated for all semantic indices specified in the line above; e.g. the symbol round denotes the set $\{b_1, b_3\}$, therefore there are rules $N_{b_1} \rightarrow \mathit{round}_{b_1}(N_{b_1})$ and $N_{b_3} \rightarrow \mathit{round}_{b_3}(N_{b_3})$. The values of $\mathcal{I}_R$ and $\mathcal{I}_S$ for each symbol are specified below the RTG rule for that symbol.

We can use $\mathcal{G}$ to generate NPs that refer to the target referent b_2 given the model shown in Fig. 2 by finding trees in $L_{\mathrm{NP}_{b_2}}(G)$ that refer to $\{b_2\}$. One such tree is $t_1 = \mathit{def}_{b_2}(\mathit{square}_{b_2}(\mathit{button}_{b_2}))$, a more detailed version of the tree in Fig. 1b. It can be derived by $\mathrm{NP}_{b_2} \Rightarrow \mathit{def}_{b_2}(\mathrm{N}_{b_2}) \Rightarrow \mathit{def}_{b_2}(\mathit{square}_{b_2}(\mathrm{N}_{b_2})) \Rightarrow t_1$. Because $\mathcal{I}_R(t_1) = \{b_2\}$, we see that $t_1 \in \mathsf{RE}_\mathcal{G}(\mathrm{NP}, b_2)$; it represents

for all $a \in U$:
$\mathrm{NP}_a \rightarrow \mathit{def}_a(\mathrm{N}_a)$
$\mathcal{I}_S(\mathit{def}_a)(w_1) = \text{the} \bullet w_1$
$\mathcal{I}_R(\mathit{def}_a)(R_1) = \mathsf{member}_a(R_1)$

for all $a \in \mathsf{button}$:
$\mathrm{N}_a \rightarrow \mathit{button}_a$
$\mathcal{I}_S(\mathit{button}_a) = \text{button}$
$\mathcal{I}_R(\mathit{button}_a) = \mathsf{button}$

for all $a \in \mathsf{round}$:
$\mathrm{N}_a \rightarrow \mathit{round}_a(\mathrm{N}_a)$
$\mathcal{I}_S(\mathit{round}_a)(w_1) = \text{round} \bullet w_1$
$\mathcal{I}_R(\mathit{round}_a)(R_1) = \mathsf{round} \cap_1 R_1$

for all $a \in \mathsf{square}$:
$\mathrm{N}_a \rightarrow \mathit{square}_a(\mathrm{N}_a)$
$\mathcal{I}_S(\mathit{square}_a)(w_1) = \text{square} \bullet w_1$
$\mathcal{I}_R(\mathit{square}_a)(R_1) = \mathsf{square} \cap_1 R_1$

for all $a, b \in \mathsf{left\_of}$:
$\mathrm{N}_a \rightarrow \mathit{leftof}_{a,b}(\mathrm{N}_a, \mathrm{NP}_b)$
$\mathcal{I}_S(\mathit{leftof}_{a,b})(w_1, w_2) = w_1 \bullet \text{to} \bullet \text{the} \bullet \text{left} \bullet \text{of} \bullet w_2$
$\mathcal{I}_R(\mathit{leftof}_{a,b})(R_1, R_2) = \mathsf{proj}_1((\mathsf{left\_of} \cap_1 R_1) \cap_2 R_2)$

for all $a, b \in \mathsf{right\_of}$:
$\mathrm{N}_a \rightarrow \mathit{rightof}_{a,b}(\mathrm{N}_a, \mathrm{NP}_b)$
$\mathcal{I}_S(\mathit{rightof}_{a,b})(w_1, w_2) = w_1 \bullet \text{to} \bullet \text{the} \bullet \text{right} \bullet \text{of} \bullet w_2$
$\mathcal{I}_R(\mathit{rightof}_{a,b})(R_1, R_2) = \mathsf{proj}_1((\mathsf{right\_of} \cap_1 R_1) \cap_2 R_2)$

Figure 3: An example SIG grammar.

the string $\mathcal{I}_S(t_1) =$ “the square button”.

A second derivation tree for b_2 is $t_2 = \mathit{def}_{b_2}(\mathit{square}_{b_2}(\mathit{square}_{b_2}(\mathit{button}_{b_2})))$, corresponding to $\mathcal{I}_S(t_2) =$ “the square square button”. It derives from NP_{b_2} in four steps, and has $\mathcal{I}_R(t_2) = \{b_2\}$. Even the small grammar $\mathcal{G}$ licences an infinite set of REs for b_2, all of which are semantically correct. Avoiding the generation of nonsensical REs like “the square square button” is a technical challenge to which we will return in Section 6. $\mathcal{G}$ can also derive relational REs; for instance, the derivation tree in Fig. 6 for the string “the button to the left of the square button” is in $\mathsf{RE}_\mathcal{G}(\mathrm{NP}, b_1)$.

Finally, $\mathcal{G}$ considers the non-distinguishing $t_3 = \mathit{def}_{b_2}(\mathit{button}_{b_2})$ (for “the button”) a valid RE for b_2. This is because member_{b_2} will quietly project the set $\{b_1, b_2, b_3\}$ (to which button_{b_2} refers) to $\{b_2\}$. As discussed in previous sections, we want to allow such non-unique REs and delegate the judgment about their quality to the probability model. It would still be straightforward, however, to impose a hard uniqueness constraint, by simply changing $\mathcal{I}_R(\mathit{def}_a)(R_1)$ to $\mathsf{uniq}_a(R_1)$ in Fig. 3. This would yield $\mathcal{I}_R(t_3) = \emptyset$, i.e. t_3 would no longer be in $\mathsf{RE}_\mathcal{G}(\mathrm{NP}, b_2)$.

4 Chart-based RE generation

We now present a chart-based algorithm for generating REs with SIG grammars. Charts allow us to represent all REs for a target referent compactly, and can be computed efficiently. We show in Section 5 that charts also lend themselves well to computing the most effective RE.

[1]Below, we will often write the RE as a string when the derivation tree is clear.

$$
\begin{array}{l}
\mathrm{N}_{b_1}/\{b_1,b_2,b_3\} \rightarrow \mathit{button}_{b_1} \\
\mathrm{N}_{b_2}/\{b_1,b_2,b_3\} \rightarrow \mathit{button}_{b_2} \\
\mathrm{N}_{b_3}/\{b_1,b_2,b_3\} \rightarrow \mathit{button}_{b_3} \\
\mathrm{N}_{b_1}/\{b_1,b_3\} \rightarrow \mathit{round}_{b_1}(\mathrm{N}_{b_1}/\{b_1,b_2,b_3\}) \\
\mathrm{N}_{b_3}/\{b_1,b_3\} \rightarrow \mathit{round}_{b_3}(\mathrm{N}_{b_3}/\{b_1,b_2,b_3\}) \\
\mathrm{N}_{b_1}/\{b_1,b_3\} \rightarrow \mathit{round}_{b_1}(\mathrm{N}_{b_1}/\{b_1,b_3\}) \\
\mathrm{N}_{b_3}/\{b_1,b_3\} \rightarrow \mathit{round}_{b_3}(\mathrm{N}_{b_3}/\{b_1,b_3\}) \\
\mathrm{N}_{b_2}/\{b_2\} \rightarrow \mathit{square}_{b_2}(\mathrm{N}_{b_2}/\{b_1,b_2,b_3\}) \\
\mathrm{N}_{b_2}/\{b_2\} \rightarrow \mathit{square}_{b_2}(\mathrm{N}_{b_2}/\{b_2\}) \\
\mathrm{NP}_{b_2}/\{b_2\} \rightarrow \mathit{def}_{b_2}(\mathrm{N}_{b_2}/\{b_1,b_2,b_3\}) \\
\mathrm{NP}_{b_2}/\{b_2\} \rightarrow \mathit{def}_{b_2}(\mathrm{N}_{b_2}/\{b_2\}) \\
\mathrm{N}_{b_1}/\{b_1\} \rightarrow \mathit{leftof}_{b_1,b_2}(\mathrm{N}_{b_1}/\{b_1,b_2,b_3\}, \mathrm{NP}_{b_2}/\{b_2\}) \\
\mathrm{N}_{b_1}/\{b_1\} \rightarrow \mathit{leftof}_{b_1,b_2}(\mathrm{N}_{b_1}/\{b_1,b_3\}, \mathrm{NP}_{b_2}/\{b_2\}) \\
\mathrm{N}_{b_1}/\{b_1\} \rightarrow \mathit{leftof}_{b_1,b_2}(\mathrm{N}_{b_1}/\{b_1\}, \mathrm{NP}_{b_2}/\{b_2\}) \\
\mathrm{N}_{b_1}/\{b_1\} \rightarrow \mathit{round}_{b_1}(\mathrm{N}_{b_1}/\{b_1\}) \\
\mathrm{NP}_{b_1}/\{b_1\} \rightarrow \mathit{def}_{b_1}(\mathrm{N}_{b_1}/\{b_1,b_2,b_3\}) \\
\mathrm{NP}_{b_1}/\{b_1\} \rightarrow \mathit{def}_{b_1}(\mathrm{N}_{b_1}/\{b_1,b_3\}) \\
\mathrm{NP}_{b_1}/\{b_1\} \rightarrow \mathit{def}_{b_1}(\mathrm{N}_{b_1}/\{b_1\}) \\
\mathrm{N}_{b_3}/\{b_3\} \rightarrow \mathit{rightof}_{b_3,b_2}(\mathrm{N}_{b_3}/\{b_1,b_2,b_3\}, \mathrm{NP}_{b_2}/\{b_2\}) \\
\mathrm{N}_{b_3}/\{b_3\} \rightarrow \mathit{rightof}_{b_3,b_2}(\mathrm{N}_{b_3}/\{b_1,b_3\}, \mathrm{NP}_{b_2}/\{b_2\}) \\
\mathrm{N}_{b_3}/\{b_3\} \rightarrow \mathit{rightof}_{b_3,b_2}(\mathrm{N}_{b_3}/\{b_3\}, \mathrm{NP}_{b_2}/\{b_2\}) \\
\mathrm{N}_{b_3}/\{b_3\} \rightarrow \mathit{round}_{b_3}(\mathrm{N}_{b_3}/\{b_3\}) \\
\mathrm{NP}_{b_3}/\{b_3\} \rightarrow \mathit{def}_{b_3}(\mathrm{N}_{b_3}/\{b_1,b_2,b_3\}) \\
\mathrm{NP}_{b_3}/\{b_3\} \rightarrow \mathit{def}_{b_3}(\mathrm{N}_{b_3}/\{b_1,b_3\}) \\
\mathrm{NP}_{b_3}/\{b_3\} \rightarrow \mathit{def}_{b_3}(\mathrm{N}_{b_3}/\{b_3\}) \\
\mathrm{N}_{b_2}/\{b_2\} \rightarrow \mathit{leftof}_{b_2,b_3}(\mathrm{N}_{b_2}/\{b_1,b_2,b_3\}, \mathrm{NP}_{b_3}/\{b_3\}) \\
\mathrm{N}_{b_2}/\{b_2\} \rightarrow \mathit{rightof}_{b_2,b_1}(\mathrm{N}_{b_2}/\{b_1,b_2,b_3\}, \mathrm{NP}_{b_1}/\{b_1\}) \\
\mathrm{N}_{b_2}/\{b_2\} \rightarrow \mathit{leftof}_{b_2,b_3}(\mathrm{N}_{b_2}/\{b_2\}, \mathrm{NP}_{b_3}/\{b_3\}) \\
\mathrm{N}_{b_2}/\{b_2\} \rightarrow \mathit{rightof}_{b_2,b_1}(\mathrm{N}_{b_2}/\{b_2\}, \mathrm{NP}_{b_1}/\{b_1\})
\end{array}
$$

Figure 4: The chart for the grammar in Fig. 3.

4.1 RE generation charts

Generally speaking, a chart is a packed data structure which describes how larger syntactic representations can be recursively built from smaller ones. In applications such as parsing and surface realization, the creation of a chart is driven by the idea that we consume some input (words or semantic atoms) as we build up larger structures. The parallel to this intuition in REG is that "larger" chart entries are more precise descriptions of the target, which is a weaker constraint than input consumption. Nonetheless, we can define REG charts whose entries are packed representations for large sets of possible REs, and compute them in terms of these entries instead of RE sets.

Technically, we represent charts as RTGs over an extended set of nonterminals. A chart for generating an RE of syntactic category A for an individual $b \in U$ is an RTG $C = (N', \Sigma, S', P')$, where $N' \subseteq N \times R(U)$ and $S' = A_b/\{b\}$. Intuitively, the nonterminal $A_b/\{a_1, \dots, a_n\}$ expresses that we *intend* to generate an RE for b from A, but each RE that we can derive from the nonterminal *actually* denotes the referent set $\{a_1, \dots, a_n\}$.

A chart for the grammar in Fig. 3 is shown in Fig. 4. To generate an NP for b_2, we let its start symbol be $S' = \mathrm{NP}_{b_2}/\{b_2\}$. The rule $\mathrm{N}_{b_2}/\{b_1, b_2, b_3\} \rightarrow \mathit{button}_{b_2}$ says that we can generate an RE t with $\mathcal{I}_R(t) = \{b_1, b_2, b_3\}$ from the nonterminal symbol N_{b_2} by expanding this symbol with the grammar rule $\mathrm{N}_{b_2} \rightarrow \mathit{button}_{b_2}$. Similarly,

$$
\frac{\begin{array}{c} A \rightarrow r(B_1, ..., B_n) \text{ in } G \\ B_1' = B_1/R_1, ..., B_n' = B_n/R_n \text{ in } N' \end{array}}{\begin{array}{c} \text{Add } A' = A/\mathcal{I}_R(r)(R_1, ..., R_n) \text{ to } N' \\ \text{Add rule } A' \rightarrow r(B_1', ..., B_n') \text{ to } P' \end{array}}
$$

Figure 5: The chart computation algorithm.

the rule $\mathrm{N}_{b_2}/\{b_2\} \rightarrow \mathit{square}_{b_2}(\mathrm{N}_{b_2}/\{b_1, b_2, b_3\})$ expresses that we can generate an RE with $\mathcal{I}_R(t) = \{b_2\}$ by expanding the nonterminal symbol N_{b_2} into $\mathit{square}_{b_2}(t')$, where t' is any tree that the chart can generate from $\mathrm{N}_{b_2}/\{b_1, b_2, b_3\}$.

4.2 Computing a chart

Given a SIG $\mathcal{G}$, a syntactic category A, and a target referent b, we can compute a chart C for $\mathsf{RE}_{\mathcal{G}}(A, b)$ using the parsing schema in Fig. 5. The schema assumes that we have a rule $A \rightarrow r(B_1, \dots, B_n)$ in G; in addition, for each $1 \leq i \leq n$ it assumes that we have already added the nonterminal $B_i' = \mathrm{B}_i/R_i$ to the chart, indicating that there is a tree t_i with $B_i \Rightarrow^* t_i$ and $\mathcal{I}_R(t_i) = R_i$. Then we know that $t = r(t_1, \dots, t_n)$ can be derived from A and that $R' = \mathcal{I}_R(t) = \mathcal{I}_R(r)(R_1, \dots, R_n)$. We can therefore add the nonterminal $A' = A/R'$ and the production rule $A' \rightarrow r(B_1', \dots, B_n')$ to the chart; this rule can be used as the first step in a derivation of t from A'. We can optimize the algorithm by adding A' and the rule only if $R' \neq \emptyset$.

The algorithm terminates when it can add no more rules to the chart. Because U is finite, this always happens after a finite number of steps, even if there is an infinite set of REs. For instance, the chart in Fig. 4 describes an infinite language of REs, including "the square button", "the button to the left of the round button", "the button to the left of the button to the right of the square button", etc. Thus it represents relational REs that are nested arbitrarily deeply through a finite number of rules.

After termination, the chart contains all rules by which a nonterminal can be decomposed into other (productive) nonterminals. As a result, $L(C)$ contains exactly the REs for b of category A:

Theorem 1 *If C is a chart for the SIG $\mathcal{G}$, the syntactic category A, and the target referent b, then $L(C) = \mathsf{RE}_{\mathcal{G}}(A, b)$.*

5 Computing best referring expressions

The chart algorithm allows us to compactly represent *all* REs for the target referent. We now show how to compute the *best* RE from the chart. We present a novel probability model $P(b|t)$ for RE resolution, and take the "best" RE to be the

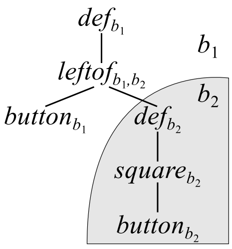


Figure 6: The derivation tree for "the button to the left of the square button".

one with the highest chance to be understood as intended. Next to the best RE itself, the algorithm also computes the entire distribution $P(b|t)$, to support later updates in an interactive setting.

Nothing in our algorithm hinges on this particular model; it can also be used with any other scoring model that satisfies a certain monotonicity condition which we spell out in Section 5.2.

5.1 A log-linear model for effective REs

We model the probability $P(b|t)$ that the listener will resolve the RE t to the object b using a log-linear model with a set of *feature functions* $f(a, t, M)$, where a is an object, t is a derivation tree, and M is the relational interpretation model.

We focus on features that only look at information that is local to a specific subtree of the RE, such as the label at the root. For instance, a feature $f_{round}(a, t', M)$ might return 1 if the root label of t' is $round_a$ and a is round in M, and 0 otherwise. Another feature $f_{def}(a, t', M)$ might return $1/k$ if t' is of the form $def_b(t'')$, $R = \mathcal{I}_R(t'')$ has k elements, and $a \in R$; and 0 otherwise. This feature counterbalances the ability of the grammar in Fig. 3 to say "the w" even when w is a non-unique description by penalizing descriptions with many possible referents through lower feature values.

When generating a relational RE, the derivation tree naturally splits into separate *regions*, each of which is meant to identify a specific object. These regions are distinguished by the semantic indices in the nonterminals that derive them; e.g., in Fig. 6, the subtree for "the square button" is an attempt to refer to b_2, whereas the RE as a whole is meant to refer to b_1. To find out how effective the RE is as a description of b_1, we evaluate the features at all nodes in the region $\mathsf{top}(t)$ containing the root of t.

Each feature function f_i is associated with a *weight* w_i. We obtain a *score tuple* $\mathsf{sc}(t')$ for some subtree t' of an RE as follows:

$$\mathsf{sc}(t') = \langle s(a_1, t', M), \ldots, s(a_m, t', M) \rangle,$$

t	b_1	b_2	b_3
"the button"	0.33	0.33	0.33
"the round button"	0.45	0.10	0.45
"the button to the left of the square button"	0.74	0.14	0.12

Figure 7: Probability distributions for some REs t.

where $U = \{a_1, \ldots, a_m\}$ and $s(a, t', M) = \sum_{i=1}^{n} w_i \cdot f_i(a, t', M)$. We then combine these into a score tuple $\mathsf{score}(t) = \sum_{u \in \mathsf{top}(t)} \mathsf{sc}(t.u)$ for the whole RE t, where $t.u$ is the subtree of t below the node u. Finally, given a score tuple $\mathsf{s} = \langle s_1, \ldots, s_m \rangle$ for t, we define the usual log-linear probability distribution as

$$P(a_i|t) = \mathsf{prob}(a_i, \mathsf{s}) = \frac{e^{s_i}}{\sum_{j=1}^{m} e^{s_j}}.$$

The *best* RE for the target referent b is then

$$\mathsf{best}_{\mathcal{G}}(A, b) = \operatorname*{arg\,max}_{t \in \mathsf{RE}_{\mathcal{G}}(A,b)} \mathsf{prob}(b, \mathsf{sc}(t)).$$

For illustration, we consider a number of REs for b_1 in our running example. We use f_{round} and f_{def} and let $w_{round} = w_{def} = 1$. In this case, the RE "the button" has a score tuple $\langle 1/3, 1/3, 1/3 \rangle$, which is the sum of the tuple $\langle 0, 0, 0 \rangle$ for f_{round} (since the RE does not use the "round" rule) and the tuple $\langle 1/3, 1/3, 1/3 \rangle$ for f_{def} (since "button" is three-way ambiguous in M). This yields a uniform probability distribution over U (see Fig. 7). By contrast, "the round button" gets $\langle 3/2, 0, 3/2 \rangle$, resulting in the distribution in the second line of Fig. 7. This RE is judged better than "the button" because it assigns a higher probability to b_1.

Relational REs involve derivation trees with multiple regions, only the top one of which is directly counted for $P(b|t)$ (see Fig. 6). We incorporate the quality of the other regions through appropriate features. In the example, we use a feature $f_{leftof}(a, t', M) = \sum_{b : \langle a, b \rangle \in \mathsf{left\_of}} P(b|t'')$, where t'' is the second subtree of t'. This feature computes the probability that the referent to which the listener resolves t'' is actually to the right of a, and will thus take a high value if t'' is a good RE for b_2. Assuming a probability distribution of $P(b_2|t') = 0.78$ and $P(b_1|t') = P(b_3|t') = 0.11$ for t' ="the square button", we get the tuple $\langle 0.78, 0.11, 0 \rangle$ for f_{leftof}, yielding the third line of Fig. 7 for $w_{leftof} = 1$.

5.2 Computing the best RE

We compute $\mathsf{best}_{\mathcal{G}}(A, b)$ from the chart by adapting the Viterbi algorithm. Our key data structure assigns a score tuple $\mathbf{is}(A')$ to each nonterminal A' in the chart. Intuitively, if the semantic index of A' is b, then $\mathbf{is}(A')$ is the score tuple $\mathsf{sc}(t)$ for the tree $t \in L_{A'}(C)$ which maximizes $P(b|t)$. We also record this best tree as $\mathsf{bt}(A')$. Thus the algorithm is correct if, after running it, we obtain $\mathsf{best}_{\mathcal{G}}(A, b) = \mathsf{bt}(\mathrm{A}_b/\{b\})$.

As is standard in chart algorithms, we limit our attention to features whose values can be computed bottom-up by local operations. Specifically, we assume that if $A' \to r(B'_1, \ldots, B'_n)$ is a rule in the chart and t_i is the best RE for B'_i for all i, then the best RE for A' that can be built using this rule is $r(t_1, \ldots, t_n)$. This means that features must be *monotonic*, i.e. that the RE that seemed locally best for B'_i leads to the best RE overall.

Under this assumption, we can compute $\mathbf{is}(A')$ and $\mathsf{bt}(A')$ bottom-up as shown in Fig. 8. We iterate over all nonterminals A' in the chart in a fixed linear order, which we call the *evaluation order*. Then we compute $\mathbf{is}(A')$ and $\mathsf{bt}(A')$ by maximizing over the rules for A'. Assume that the best RE for A' can be constructed using the rule $A' \to r(B'_1, \ldots, B'_n)$. Then if, at the time we evaluate A', we have fully evaluated all the B'_i in the sense that $\mathsf{bt}(B'_i)$ is actually the best RE for B'_i, the algorithm will assign the best RE for A' to $\mathsf{bt}(A')$, and its score tuple to $\mathbf{is}(A')$. Thus, if we call an evaluation order *exact* if the nonterminals on the right-hand side of each rule in the chart come before the nonterminal on the left-hand side, we can inductively prove the following theorem:

Theorem 2 *If the evaluation order is exact, then for every nonterminal A' in the chart, we obtain* $\mathsf{bt}(A') = \arg\max_{t \in L_{A'}(C)} P(\mathsf{ix}(A')|t)$ *and* $\mathbf{is}(A') = \mathsf{sc}(\mathsf{bt}(A'))$.

In other words, the algorithm is correct if the evaluation order is exact. If it is not, we might compute a sub-optimal RE as $\mathsf{bt}(A')$, which underestimates $\mathbf{is}(A')$. The choice of evaluation order is thus crucial.

for nonterminals A' in evaluation order **do**
    for rules $\mathbf{r}$ of the form $A' \to r(B'_1, \ldots, B'_n)$ **do**
        $a = \mathsf{ix}(A')$
        $t' = r(\mathsf{bt}(B'_1), \ldots, \mathsf{bt}(B'_n))$
        $\mathsf{s} = \mathsf{sc}(t') + \sum_{\substack{i=1 \\ \mathsf{ix}(B'_i)=a}}^{n} \mathbf{is}(B'_i)$
        if $\mathsf{prob}(a, \mathsf{s}) > \mathsf{prob}(a, \mathbf{is}(A'))$ **then**
            $\mathbf{is}(A') = \mathsf{s}$
            $\mathsf{bt}(A') = t'$

Figure 8: Computing the best RE.

6 Evaluating charts with cycles

It remains to show how we can determine an exact evaluation order for a given chart. One way to think about the problem is to consider the *ordering graph* $O(C)$ of the chart C (see Fig. 9 for an example). This is a directed graph whose nodes are the nonterminals of the chart; for each rule $A' \to r(B'_1, \ldots, B'_n)$ in C, it has an edge from B'_i to A' for each i. If this graph is acyclic, we can simply compute a topological sort of $O(C)$ to bring the nodes into a linear order in which each B'_i precedes A'. This is enough to evaluate charts using certain simpler models. For instance, we can apply our REG algorithm to the log-linear model of Golland et al. (2010). Because they only generate REs with a bounded number of relations, their grammars effectively only describe finite languages. In such a case, our charts are always acyclic, and therefore a topological sort of $O(C)$ yields an exact evaluation order.

This simple approach will not work with grammars that allow arbitrary recursion, as they can lead to charts with cycles (indicating an infinite set of valid REs). E.g. the chart in Fig. 4 contains a rule $\mathrm{N}_{b_2}/\{b_2\} \to$ *square*$_{b_2}(\mathrm{N}_{b_2}/\{b_2\})$ (shown in Fig. 9), which can be used to construct the RE t' = "the square square button" in addition to the RE t = "the square button". Such cycles can be *increasing* with respect to a log-linear probability model, i.e. the model considers t' a better RE than t. Indeed, t has a score tuple of $\langle 0, 2, 0 \rangle$, giving $P(b_2|t) = 0.78$. By contrast, t' has a score tuple of $\langle 0, 3, 0 \rangle$, thus $P(b_2|t') = 0.91$. This can be continued indefinitely, with each addition of "square" increasing the probability of being resolved to b_2. Thus, there is no best RE for b_2; every RE can be improved by adding another copy of "square".

In such a situation, it is a challenge to even compute *any* score for every nonterminal without running into infinite loops. We can achieve this by decomposing $O(C)$ into its strongly connected components (SCCs), i.e. the maximal subgraphs in which each node is reachable from any other node. We then consider the *component graph* $O'(C)$; its nodes are the SCCs of $O(C)$, and it has an edge from c_1 to c_2 if $O(C)$ has an edge from some node in c_1 to some node in c_2. $O'(C)$ is acyclic by construction, so we can compute a topological

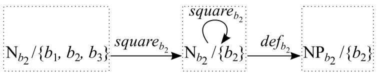


Figure 9: A fragment of the ordering graph for the chart in Fig. 4. Dotted boxes mark SCCs.

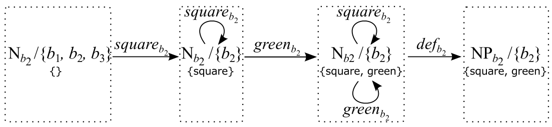


Figure 10: A fragment of a chart ordering graph for a grammar with enriched nonterminals.

sort and order all nonterminals from earlier SCCs before all nonterminals from later SCCs. Within each SCC, we order the nonterminals in the order in which they were discovered by the algorithm in Fig. 5. This yields a linear order on nonterminals, which at least ensures that by the time we evaluate a nonterminal A', there is at least one rule for A' whose right-hand nonterminals have all been evaluated; so $\mathbf{is}(A')$ gets at least *some* value.

In our example, we obtain the order $\mathrm{N}_{b_2}/\{b_1, b_2, b_3\}$, $\mathrm{N}_{b_2}/\{b_2\}$, $\mathrm{NP}_{b_2}/\{b_2\}$. The rule $\mathrm{N}_{b_2}/\{b_2\} \to \mathit{square}_{b_2}(\mathrm{N}_{b_2}/\{b_2\})$ will thus not be considered in the evaluation of $\mathrm{N}_{b_2}/\{b_2\}$, and the algorithm returns "the square button". The algorithm computes optimal REs for acyclic charts, and also for charts where all cycles are *decreasing*, i.e. using the rules in the cycle make the RE worse. This enables us, for instance, to encode the REG problem of Krahmer et al. (2003) into ours by using a feature that evaluates the rule for each attribute to its (negative) cost according to the Krahmer model. Krahmer et al. assume that every attribute has positive cost, and is only used if it is necessary to make the RE distinguishing. Thus all cycles in the chart are decreasing.

One limitation of the algorithm is that it does not overspecify. Suppose that we extend the example model in Fig. 2 with a color predicate green $= \{b_2\}$. We might then want to prefer "the green square button" over "the square button" because it is easier to understand. But since all square objects (i.e. $\{b_2\}$) are also green, using "green" does not change the denotation of the RE, i.e. it is represented by a loop from $\mathrm{N}_{b_2}/\{b_2\}$ to $\mathrm{N}_{b_2}/\{b_2\}$, which is skipped by the algorithm. One idea could be to *break* such cycles by the careful use of a richer set of nonterminals in the grammar; e.g., they might record the set of all attributes that were used in the RE. Our example rule would then become $\mathrm{N}_{b_2}/\{b_2\}/\{square, green\} \to \mathit{green}_{b_2}(\mathrm{N}_{b_2}/\{b_2\}/\{square\})$, which the algorithm can make use of (see Fig. 10).

7 Conclusion

We have shown how to generate REs using charts. Based on an algorithm for computing a chart of all valid REs, we showed how to compute the RE that maximizes the probability of being understood as the target referent. Our algorithm integrates REG with surface realization. It generates distinguishing REs if this is specified in the grammar; otherwise, it computes the best RE without regard to uniqueness, using features that prefer unambiguous REs as part of the probability model.

Our algorithm can be applied to earlier models of REG, and in these cases is guaranteed to compute optimal REs. The probability model we introduced here is more powerful, and may not admit "best" REs. We have shown how the algorithm can still do something reasonable in such cases, but this point deserves attention in future research, especially with respect to overspecification.

We evaluated the performance of our chart algorithm on a number of randomly sampled input scenes from the GIVE Challenge, which contained 24 objects on average. Our implementation is based on the IRTG tool available at `irtg.googlecode.com`. While in the worst case the chart computation is exponential in the input size, in practice runtimes did not exceed 60 ms for the grammar shown in Fig. 3.

We have focused here on *computing* best REs given a probability model. We have left *training* the model and evaluating it on real-world data for future work. Because our probability model focuses on effectiveness for the listener, rather than human-likeness, our immediate next step is to train it on an interaction corpus which records the reactions of human listeners to system-generated REs. A further avenue of research is to deliberately generate succinct but ambiguous REs when the model predicts them to be easily understood. We will explore ways of achieving this by combining the effectiveness model presented here with a language model that prefers succinct REs.

Acknowledgments. We thank Emiel Krahmer, Stephan Oepen, Konstantina Garoufi, Martín Villalba and the anonymous reviewers for their useful comments and discussions. The authors were supported by the SFB 632 "Information Structure".

References

Douglas E. Appelt. 1985. *Planning English sentences*. Cambridge University Press.

Carlos Areces, Alexander Koller, and Kristina Striegnitz. 2008. Referring expressions as formulas of description logic. In *Proceedings of the 5th International Natural Language Generation Conference (INLG)*.

Anja Belz, Eric Kow, Jette Viethen, and Albert Gatt. 2008. The GREC challenge 2008: Overview and evaluation results. In *Proceedings of the 5th International Conference on Natural Language Generation (INLG)*.

John Carroll, Ann Copestake, Dan Flickinger, and Victor Poznanski. 1999. An efficient chart generator for (semi-)lexicalist grammars. In *Proceedings of the 7th European Workshop on Natural Language Generation*.

David Chiang. 2007. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–228.

Hubert Comon, Max Dauchet, Rémi Gilleron, Christof Löding, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. 2007. Tree automata techniques and applications. Available on `http://tata.gforge.inria.fr/`.

Robert Dale and Ehud Reiter. 1995. Computational interpretations of the Gricean Maxims in the generation of referring expressions. *Cognitive Science*, 19(2):233–263.

Nikos Engonopoulos, Martin Villalba, Ivan Titov, and Alexander Koller. 2013. Predicting the resolution of referring expressions from user behavior. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Seattle.

Nicholas FitzGerald, Yoav Artzi, and Luke Zettlemoyer. 2013. Learning distributions over logical forms for referring expression generation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Konstantina Garoufi and Alexander Koller. 2013. Generation of effective referring expressions in situated context. *Language and Cognitive Processes*.

Albert Gatt and Anja Belz. 2010. Introducing shared task evaluation to NLG: The TUNA shared task evaluation challenges. In E. Krahmer and M. Theune, editors, *Empirical Methods in Natural Language Generation*, number 5790 in LNCS, pages 264–293. Springer.

Ferenc Gécseg and Magnus Steinby. 1997. Tree languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, chapter 1, pages 1–68. Springer-Verlag.

Dave Golland, Percy Liang, and Dan Klein. 2010. A game-theoretic approach to generating spatial descriptions. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Jonathan Graehl, Kevin Knight, and Jonathan May. 2008. Training tree transducers. *Computational Linguistics*, 34(3).

B. Jones, J. Andreas, D. Bauer, K.-M. Hermann, and K. Knight. 2012. Semantics-based machine translation with hyperedge replacement grammars. In *Proceedings of COLING*.

Ron Kaplan and Jürgen Wedekind. 2000. LFG generation produces context-free languages. In *Proceedings of the 18th COLING*.

Martin Kay. 1996. Chart generation. In *Proceedings of the 34th ACL*.

John Kelleher and Geert-Jan Kruijff. 2006. Incremental generation of spatial referring expressions in situated dialogue. In *In Proceedings of Coling-ACL '06, Sydney Australia*.

Alexander Koller and Marco Kuhlmann. 2011. A generalized view on parsing and translation. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 2–13. Association for Computational Linguistics.

Alexander Koller and Matthew Stone. 2007. Sentence generation as a planning problem. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics (ACL)*.

Alexander Koller, Kristina Striegnitz, Donna Byron, Justine Cassell, Robert Dale, Johanna Moore, and Jon Oberlander. 2010. The First Challenge on Generating Instructions in Virtual Environments. In E. Krahmer and M. Theune, editors, *Empirical Methods in Natural Language Generation*, number 5790 in LNAI, pages 337–361. Springer.

Yannis Konstas and Mirella Lapata. 2012. Concept-to-text generation via discriminative reranking. In *Proceedings of the 50th ACL*.

Ruud Koolen, Albert Gatt, Martijn Goudbeek, and Emiel Krahmer. 2011. Factors causing overspecification in definite descriptions. *Journal of Pragmatics*, 43:3231–3250.

Emiel Krahmer and Kees van Deemter. 2012. Computational generation of referring expressions: A survey. *Computational Linguistics*, 38(1):173–218.

Emiel Krahmer, Sebastiaan van Erk, and André Verleg. 2003. Graph-based generation of referring expressions. *Computational Linguistics*, 29(1):53–72.

Wei Lu and Hwee Tou Ng. 2011. A probabilistic forest-to-string model for language generation from typed lambda calculus expressions. In *Proceedings of EMNLP*.

Margaret Mitchell, Kees van Deemter, and Ehud Reiter. 2013. Generating expressions that refer to visible objects. In *Proceedings of NAACL-HLT*, pages 1174–1184.

Matthew Stone, Christine Doran, Bonnie Webber, Tonia Bleam, and Martha Palmer. 2003. Microplanning with communicative intentions: The SPUD system. *Computational Intelligence*, 19(4):311–381.

Liane Wardlow Lane and Victor Ferreira. 2008. Speaker-external versus speaker-internal forces on utterance form: Do cognitive demands override threats to referential success? *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 34:1466–1481.

Yuk Wah Wong and Raymond J. Mooney. 2007. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th ACL*.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI)*.