# Integrated sentence generation with charts

**Alexander Koller** and **Nikos Engonopoulos**
Saarland University, Saarbrücken, Germany
{koller|nikos}@coli.uni-saarland.de

## Abstract

Integrating surface realization and the generation of referring expressions (REs) into a single algorithm can improve the quality of the generated sentences. Existing algorithms for doing this, such as SPUD and CRISP, are search-based and can be slow or incomplete. We offer a chart-based algorithm for integrated sentence generation which supports efficient search through chart pruning.

## 1 Introduction

It has long been argued (Stone et al., 2003) that the strict distinction between surface realization and sentence planning in the classical NLG pipeline (Reiter and Dale, 2000) can cause difficulties for an NLG system. Generation decisions that look good to the sentence planner may be hard or impossible for the realizer to express in natural language. Furthermore, a standalone sentence planner must compute each RE separately, thus missing out on opportunities for succinct REs that are ambiguous in isolation but correct in context (Stone and Webber, 1998).

Algorithms such as SPUD (Stone et al., 2003) and CRISP (Koller and Stone, 2007) perform surface realization and parts of sentence planning, including RE generation, in an integrated fashion. Such integrated algorithms for sentence generation can balance the needs of the realizer and the sentence planner and take advantage of opportunities for succinct realizations. However, integrated sentence planning multiplies the complexities of two hard combinatorial problems, and thus existing, search-based algorithms can be inefficient or fail to find a good solution; SPUD's greedy search strategy may even find no solution at all, even when one exists.

By contrast, chart-based algorithms have been shown in parsing to remain efficient and accurate even for large inputs because they support structure sharing and very effective pruning techniques. Chart algorithms have been successfully applied to surface realization (White, 2004; Gardent and Kow, 2005; Carroll and Oepen, 2005; Schwenger et al., 2016), but in RE generation, most algorithms are not chart-based, see e.g. (Dale and Reiter, 1995; Krahmer et al., 2003). One exception is the chart-based RE generation of Engonopoulos and Koller (2014).

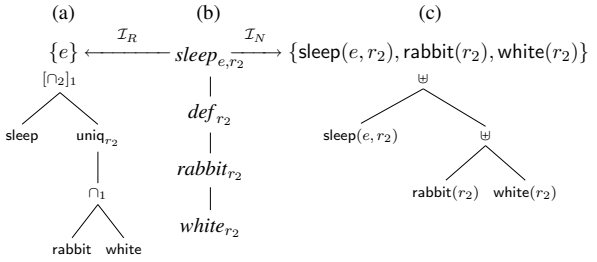
In this paper, we present a chart-based algorithm for integrated surface realization and RE generation. This makes it possible – to our knowledge, for the first time – to apply chart-based pruning techniques to integrated sentence generation. Our algorithm extends the chart-based RE generation algorithm of Engonopoulos and Koller (2014) by keeping track of the semantic content that has been expressed by each chart item. Because it is modular on the string side, the same algorithm can be used to generate with context-free grammars or TAGs, with or without feature structures, at no expense in runtime efficiency. An open-source implementation of our algorithm, based on the Alto system (Gontrum et al., 2017), can be found at `bitbucket.org/tclup/alto`.

## 2 Chart-based integrated generation

We first describe the grammar formalism we use. Then we explain the sentence generation algorithm and discuss its runtime performance.



*Proceedings of The 10th International Natural Language Generation conference*, pages 139–143,
Santiago de Compostela, Spain, September 4-7 2017. 




(d) $\mathcal{I}_S : \bullet(\bullet(\text{the}, \bullet(\text{white}, \text{rabbit})), \text{sleeps}) =$ "the white rabbit sleeps"

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

## 2.1 Semantically interpreted grammars

We describe the integrated sentence generation problem in terms of *semantically interpreted grammars (SIGs)* (Engonopoulos and Koller, 2014), a special case of Interpreted Regular Tree Grammars (Koller and Kuhlmann, 2011). We introduce SIGs by example, and refer to Engonopoulos and Koller (2014) for detailed definitions.

An example SIG grammar is shown in Fig. 2. At the core of each grammar rule is a rule of the form $\mathrm{A}_a \rightarrow f(\mathrm{B}_b, \ldots, \mathrm{Z}_z)$. The symbols $\mathrm{A}, \ldots, \mathrm{Z}$ are nonterminals such as S, NP, VP, and $a, \ldots, z$ are semantic indices, i.e. constants for individuals indicating to which object in the model a natural-language expression is intended to refer. These core rules allow us to recursively derive a *derivation tree*, such as the one shown in Fig. 1b, representing the abstract syntactic structure of a natural-language expression.

Each derivation tree $t$ is mapped to a string through a function $\mathcal{I}_S$. This function is defined recursively for each rule of the grammar. In the example grammar of Fig. 2, we have $\mathcal{I}_S(\textit{white}_{r_2}) =$ white, i.e. the word "white". Given a string $w_1$, we have $\mathcal{I}_S(\textit{rabbit}_{r_2})(w_1) = w_1 \bullet$ rabbit, where "$\bullet$" is string concatenation. This means that given a subtree $t'$ which evaluates to the string $w_1$, a string for the tree $\textit{rabbit}_{r_2}(t')$ is constructed by appending the word "rabbit" after $w_1$. For the complete derivation tree $t$ in Fig. 1, we obtain $\mathcal{I}_S(t) =$ "the white rabbit sleeps".

At the same time and in the same way, the derivation tree is also evaluated to a set of referents through a function $\mathcal{I}_R$. Constants, such as rabbit and sleep, are interpreted as subsets of and relations between the individuals in a given model. For instance, in the model of Fig. 3, rabbit denotes the set $\{r_1, r_2\}$, whereas in denotes the relation $\{(r_1, h_1), (f_1, h_2)\}$.

for all $e, a \in \mathsf{sleep}$:
$\mathrm{S}_e \rightarrow \textit{sleep}_{e,a}(\mathrm{NP}_a)$
$\mathcal{I}_S(\textit{sleep}_{e,a})(w_1) = w_1 \bullet \text{sleeps}$
$\mathcal{I}_R(\textit{sleep}_{e,a})(R_1) = [\mathsf{sleep} \cap_2 \mathsf{uniq}_a(R_1)]_1$
$\mathcal{I}_N(\textit{sleep}_{e,a})(N_1) = \{\mathsf{sleep}(e, a)\} \uplus N_1$

for all $a \in \mathsf{rabbit}$:
$\mathrm{N}_a \rightarrow \textit{rabbit}_a(\mathrm{Adj}_a)$
$\mathcal{I}_S(\textit{rabbit}_a)(w_1) = w_1 \bullet \text{rabbit}$
$\mathcal{I}_R(\textit{rabbit}_a)(R_1) = \mathsf{rabbit} \cap_1 R_1$
$\mathcal{I}_N(\textit{rabbit}_a)(N_1) = \{\mathsf{rabbit}(a)\} \uplus N_1$

for all $a \in U$:
$\mathrm{NP}_a \rightarrow \textit{def}_a(\mathrm{N}_a)$
$\mathcal{I}_S(\textit{def}_a)(w_1) = \text{the} \bullet w_1$
$\mathcal{I}_R(\textit{def}_a)(R_1) = R_1$
$\mathcal{I}_N(\textit{def}_a)(N_1) = N_1$

for all $a \in U$:
$\mathrm{N}_a \rightarrow \textit{thing}_a(\mathrm{Adj}_a)$
$\mathcal{I}_S(\textit{thing}_a)(w_1) = w_1 \bullet \text{thing}$
$\mathcal{I}_R(\textit{thing}_a)(R_1) = R_1$
$\mathcal{I}_N(\textit{thing}_a)(N_1) = N_1$

for all $a \in \mathsf{white}$:
$\mathrm{Adj}_a \rightarrow \textit{white}_a$
$\mathcal{I}_S(\textit{white}_a) = \text{white}$
$\mathcal{I}_R(\textit{white}_a) = \text{white}$
$\mathcal{I}_N(\textit{white}_a) = \{\mathsf{white}(a)\}$

for all $a \in U$:
$\mathrm{Adj}_a \rightarrow \textit{nop}_a$
$\mathcal{I}_S(\textit{nop}_a) = \epsilon$
$\mathcal{I}_R(\textit{nop}_a) = U$
$\mathcal{I}_N(\textit{nop}_a) = \emptyset$

for all $e, a, b \in \mathsf{takefrom}(e, a, b)$:
$\mathrm{S}_e \rightarrow \textit{takefrom}_{e,a,b}(\mathrm{NP}_a, \mathrm{NP}_b)$
$\mathcal{I}_S(\textit{takefrom}_{e,a,b})(w_1, w_2) = \text{take} \bullet w_1 \bullet \text{from} \bullet w_2$
$\mathcal{I}_R(\textit{takefrom}_{e,a,b})(R_1, R_2) = [(\mathsf{takefrom} \cap_2 \mathsf{uniq}_a(R_1 \cap_1 [\mathsf{in} \cap_2 R_2]_1)) \cap_3 \mathsf{uniq}_b(R_2 \cap_1 [\mathsf{in} \cap_1 R_1]_2)]_1$
$\mathcal{I}_N(\textit{takefrom}_{e,a,b})(N_1, N_2) = \{\mathsf{takefrom}(e, a, b)\} \uplus N_1 \uplus N_2$

**Figure 2:** An example grammar.


**Figure 3:** An example model.

These relations are then combined using intersection $R_1 \cap_i R_2$ (yielding the subset of elements of $R_1$ whose $i$-th component is an element of the set $R_2$), projection $[R]_i$ (yielding the set of $i$-th components of the tuples in $R$), and the uniqueness checker $\mathsf{uniq}_a(R)$ (yielding $R$ if $R = \{a\}$ and $\emptyset$ otherwise). Under this interpretation, the derivation tree in Fig. 1 maps to the set $\{e\}$, given the model in Fig. 3.

Observe, finally, that each rule is annotated with a "for all" clause, which creates instances of the rule for each tuple of individuals that satisfies the condition. We also mention that although we only use unary attributes such as "rabbit" and "white" in this example grammar, SIGs deal easily with relational attributes such as "in" or "next to"; see Engonopoulos and Koller (2014).

## 2.2 Integrated sentence generation with SIGs

Engonopoulos and Koller (2014) describe an algorithm which, given a SIG grammar $G$ and a set $R$

$$\frac{[B_1, R_1, N_1] \quad \ldots \quad [B_k, R_k, N_k] \quad A \to r(B_1, ..., B_k) \text{ in } G}{[A, \mathcal{I}_R(r)(R_1, \ldots, R_k), \mathcal{I}_N(r)(N_1, \ldots, N_k)]}$$

**Figure 4:** The chart computation algorithm.

of target referents, will compute a chart describing all derivations $t$ of $G$ with $\mathcal{I}_R(t) = R$ – that is, all semantically valid REs for $R$.

Here we extend both SIGs and this algorithm to include surface realization. We assume that the generation algorithm is given a set $N$ of *semantic atoms* in addition to the grammar $G$ and referent set $R$, and should return only derivations that refer to $R$ while expressing at least all the atoms in $N$. We achieve this by adding an interpretation $\mathcal{I}_N$ to SIGs, such that $\mathcal{I}_N(t)$ will return a set of semantic atoms expressed by the derivation tree $t$. We have added such $\mathcal{I}_N$ clauses to the grammar in Fig. 2. For example, the rule for $\textit{white}_{r_2}$ expresses the set $\{\mathsf{white}(r_2)\}$ of semantic atoms. The rule for $\textit{rabbit}_{r_2}$ evaluates to the disjoint union of $\{\mathsf{rabbit}(r_2)\}$ with whatever its "Adj" subtree expressed. Thus, the $\mathcal{I}_N$ interpretation keeps track of the semantic atoms expressed by each subtree of a derivation tree; in the example of Fig. 1, we see that the derivation tree as a whole expresses the semantic atoms $\{\mathsf{sleep}(e, r_2), \mathsf{rabbit}(r_2), \mathsf{white}(r_2)\}$.

Given a grammar $G$, target referent set $R$, and target semantic content $N$, we can now compute a chart that describes all derivation trees $t$ of $G$ such that $\mathcal{I}_R(t) = R$ and $\mathcal{I}_N(t) \supseteq N$; thus this algorithm performs surface realization and RE generation at the same time. The algorithm, shown in Fig. 4 in the form of a parsing schema (Shieber et al., 1995), computes *chart items* $[A, R, N]$ in a bottom-up fashion. Such an item states that there is a tree $t$ such that $t$ can be derived from $A$, and we have $\mathcal{I}_R(t) = R$ and $\mathcal{I}_N(t) = N$. Given $k$ items for subtrees derived from the nonterminals $B_1, \ldots, B_k$ as premises and a rule $r$ that can combine these into a nonterminal $A$, the algorithm creates a new item for $A$ in which the $\mathcal{I}_R$ and $\mathcal{I}_N$ functions for that rule were applied to the referent sets and semantic contents of the subtrees. Given the inputs $R$ and $N$, we define a *goal item* to be an item $[S, R, N']$ where $S$ is the start symbol of the grammar and $N' \supseteq N$. Each goal item the algorithm discovers thus represents a sentence that achieves the given communicative goals.

An example run of the algorithm, for the inputs $R = \{e\}$ and $N = \{\mathsf{rabbit}(r_2)\}$, is shown in Fig. 5. Each row in the chart corresponds to one application of the rule in Fig. 4. The grammar rule that was used, along with any premises, is given in brackets to the right. Observe that the only goal item, (9), corresponds to the derivation in Fig. 1, and hence the output string "the white rabbit sleeps"; the derivation can be reconstructed by following the backpointers to the premise items recursively. Observe also that (10) – for "the white thing sleeps" – is not a goal item because its semantic content is not a superset of $N$. The item (11) – for "the rabbit sleeps" – is not a goal item either: Its referent set is empty because (8) is not a unique RE for $r_2$, and thus the term $\mathsf{uniq}_{r_2}(R_1)$ in the "sleep" rule evaluates to the empty set. Thus, the algorithm performs both surface realization and RE generation.

| | | |
|---|---|---|
| 1 | $[\mathrm{Adj}_{r_2}, \{r_2\}, \{\textit{white}_{r_2}\}]$ | $(\textit{white}_{r_2})$ |
| 2 | $[\mathrm{Adj}_{r_2}, U, \emptyset]$ | $(\textit{nop}_{r_2})$ |
| 3 | $[\mathrm{N}_{r_2}, \{r_2\}, \{\textit{white}_{r_2}, \textit{rabbit}_{r_2}\}]$ | $(\textit{rabbit}_{r_2}, 1)$ |
| 4 | $[\mathrm{N}_{r_2}, \{r_2\}, \{\textit{white}_{r_2}\}]$ | $(\textit{thing}_{r_2}, 1)$ |
| 5 | $[\mathrm{N}_{r_2}, \{r_1, r_2\}, \{\textit{rabbit}_{r_2}\}]$ | $(\textit{rabbit}_{r_2}, 2)$ |
| 6 | $[\mathrm{NP}_{r_2}, \{r_2\}, \{\textit{white}_{r_2}, \textit{rabbit}_{r_2}\}]$ | $(\textit{the}_{r_2}, 3)$ |
| 7 | $[\mathrm{NP}_{r_2}, \{r_2\}, \{\textit{white}_{r_2}\}]$ | $(\textit{the}_{r_2}, 4)$ |
| 8 | $[\mathrm{NP}_{r_2}, \{r_1, r_2\}, \{\textit{rabbit}_{r_2}\}]$ | $(\textit{the}_{r_2}, 5)$ |
| 9 | $[\mathrm{S}_e, \{e\}, \{\textit{white}_{r_2}, \textit{rabbit}_{r_2}, \textit{sleep}_{e,r_2}\}]$ | $(\textit{sleep}_e, 6)$ |
| 10 | $[\mathrm{S}_e, \{e\}, \{\textit{white}_{r_2}, \textit{sleep}_{e,r_2}\}]$ | $(\textit{sleep}_e, 7)$ |
| 11 | $[\mathrm{S}_e, \emptyset, \{\textit{rabbit}_{r_2}, \textit{sleep}_{e,r_2}\}]$ | $(\textit{sleep}_e, 8)$ |

**Figure 5:** Excerpt from the chart for "The white rabbit sleeps."

### 2.3 Generating succinct REs in context

One advantage of integrating surface realization with RE generation is that REs can be more succinct in the context of a larger grammatical construction than in isolation. The shortest standalone RE for $r_1$ in Fig. 3 is "the brown rabbit", but it is perfectly felicitous to say "take *the rabbit* from the hat". Stone and Webber (1998) explain this in terms of the presuppositions of the verb "take X from Y", which say that X must be in Y, and thus the REs X and Y can mutually constrain each other. They also show how the SPUD algorithm can generate such succinct REs in the context of the verb, by global reasoning over the referent sets of all REs in the sentence.

Our algorithm can generate such REs as well, and can do it in an efficient, chart-based way. Assume $R = \{e_2\}$ and $N = \{\mathsf{takefrom}(e_2, r_1, h_1)\}$ and the grammar in Fig. 2. The chart algorithm will construct items for the sub-derivation-

trees $t_1 = \mathit{def}_{r_1}(\mathit{rabbit}_{r_1}(\mathit{nop}_{r_1}))$ ("the rabbit") and $t_2 = \mathit{def}_{h_1}(\mathit{hat}_{h_1}(\mathit{nop}_{h_1}))$ ("the hat"), with $R_1 = \mathcal{I}_R(t_1) = \{r_1, r_2\}$ and $R_2 = \mathcal{I}_R(t_2) = \{h_1, h_2\}$; thus, these REs are not by themselves unique. These trees are then combined with the rule $\mathit{takefrom}_{e_2,r_1,h_1}$. This rule intersects $R_1$ with the set of things that are "in" an element of $R_2$, encoding the presupposition of "take X from Y". Thus $R_1 \cap_1 [\mathsf{in} \cap_2 R_2]_1$ evaluates to $\{r_1\}$, satisfying the uniqueness condition. Thus, the algorithm returns $t = \mathit{takefrom}_{e_2,r_1,h_1}(t_1, t_2)$ as a valid realization.

Note that we achieved the ability to let REs mutually constrain each other by moving the requirement for semantic uniqueness to the verb that subcategorizes for the RE. This is in contrast to the standard assumption that it is the definite article that requires uniqueness, but permits us a purely grammar-based treatment of mutually constraining REs which requires no further reasoning capabilities.

### 2.4 Chart generation with heuristics

Our algorithm can enumerate all subsets $N$ of the true semantic atoms in the model, and thus has worst-case exponential runtime. This is probably unavoidable, given that surface realization and the generation of shortest REs are both NP-complete (Koller and Striegnitz, 2002; Dale and Reiter, 1995).

However, because it is a chart-based algorithm, we can use heuristics to avoid computing the whole chart, and thus speed up the search for the best solution. To get an impression of this, assume that we are looking for a *short* sentence; other optimization criteria are also possible. We first compute the full chart $C_{\mathcal{R}}$ for the $\mathcal{I}_R$ part of the input alone, using essentially the same algorithm as Engonopoulos and Koller (2014). From $C_{\mathcal{R}}$ we compute the *distance* of each chart item to a goal item, i.e. the minimal number of rules that must be applied to the item to produce a goal item. We then refine the items of $C_{\mathcal{R}}$ by adding the $\mathcal{I}_N$ parts to each chart item. Unprocessed chart items $[A, R, N]$ are organized on an agenda which is ordered lexicographically by the number of atoms in the target semantic content that are not yet realized in $N$ and then the distance of $[A, R]$ to a goal item in $C_{\mathcal{R}}$. We stop the chart computation once the first goal item has been found.

Using this pruning strategy, we measured runtimes with problems from the GIVE Challenge (Koller et al., 2010) on an 2.9 GHz Intel Core i5 CPU.[1] We compared the performance of our system against CRISP, which uses the FF planner (Hoffmann and Nebel, 2001) to perform the search. For CRISP, we only measured the time spent in running the planner. On the most complex scene from GIVE that we tried, our system took 13 ms to generate the sentence "Push the button to the left of the flower", outperforming CRISP which generated the same sentence in 50 ms. Note that it is possible to construct (not entirely realistic) inputs for the generator on which FF's much more sophisticated search strategy outperforms the heuristic described above. By incorporating such a heuristic into chart generation, e.g. as in Schwenger et al. (2016), our system could be accelerated further.

## 3 Conclusion

We have presented a chart-based algorithm for integrated surface realization and RE generation. Compared to earlier approaches to integrated sentence generation, our algorithm can exploit the capabilities of charts for structure-sharing and pruning to achieve higher runtime performance in practice. We have only presented a simple pruning strategy here, but it would be astraightforward extension to incorporate pruning strategies from surface realization (White, 2004; Schwenger et al., 2016).

One advantage of our algorithm is that it is agnostic of the grammar formalism that is used on the string side. We have used context-free rules for reading off string representations from the generated derivation trees, but because SIGs are special case of IRTGs, we could instead use a tree-adjoining grammar to construct strings instead (Koller and Kuhlmann, 2012). In fact, the runtime experiments in Section 2.4 were based on a TAG grammar to allow direct comparison with CRISP.

With the algorithm presented here, it may become feasible for the first time to perform integrated sentence generation in the context of practical applications. So far, grammars that support this lag far behind grammars for surface realization in size and complexity. It would thus be interesting to either convert existing surface realization grammars to SIGs, or to learn such grammars from data.

[1] We let the Java VM warm up before measuring runtimes.

## References


John Carroll and Stephan Oepen. 2005. High efficiency realization for a wide-coverage unification grammar. In *Proceedings of the 2nd IJCNLP*.

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 and Alexander Koller. 2014. Generating effective referring expressions using charts. In *Proceedings of the 8th International Conference on Natural Language Generation (INLG)*, Philadelphia.

Claire Gardent and Eric Kow. 2005. Generating and selecting grammatical paraphrases. In *Proceedings of ENLG*.

Johannes Gontrum, Jonas Groschwitz, Alexander Koller, and Christoph Teichmann. 2017. Alto: Rapid prototyping for parsing and translation. In *Proceedings of the EACL Demo Session*.

Jörg Hoffmann and Bernhard Nebel. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302.

Alexander Koller and Marco Kuhlmann. 2011. A generalized view on parsing and translation. In *Proceedings of the 12th International Conference on Parsing Technologies (IWPT)*.

Alexander Koller and Marco Kuhlmann. 2012. Decomposing TAG algorithms using simple algebraizations. In *Proceedings of the 11th TAG+ Workshop*.

Alexander Koller and Matthew Stone. 2007. Sentence generation as a planning problem. In *Proceedings of the 45th ACL*.

Alexander Koller and Kristina Striegnitz. 2002. Generation as dependency parsing. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 17–24, Philadelphia, PA, USA.

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 LNCS, pages 337–361. Springer.

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

Ehud Reiter and Robert Dale. 2000. *Building Natural Language Generation Systems*. Cambridge University Press, Cambridge, England.

Maximilian Schwenger, Álvaro Torralba, Jörg Hoffmann, David M. Howcroft, and Vera Demberg. 2016. From OpenCCG to AI planning: Detecting infeasible edges in sentence generation. In *Proceedings of the 26th International Conference on Computational Linguistics (COLING)*.

Stuart Shieber, Yves Schabes, and Fernando Pereira. 1995. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24(1–2):3–36.

Matthew Stone and Bonnie Webber. 1998. Textual economy through close coupling of syntax and semantics. In *Proceedings of the 9th INLG*.

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.

Michael White. 2004. Reining in CCG chart realization. In Anja Belz, Roger Evans, and Paul Piwek, editors, *Proceedings of the Third International Conference on Natural Language Generation (INLG04)*.