Domain proposal: Sentence generation with tree-adjoining grammars (CRISP)

Alexander Koller
Cluster of Excellence
Saarland University
Saarbrücken, Germany
koller@mmci.uni-saarland.de

Ron Petrick
School of Informatics
University of Edinburgh
Edinburgh, Scotland, UK
rpetrick@inf.ed.ac.uk

Description of domain and problems

Overview The CRISP domain is motivated by the problem of *sentence generation* in *natural language generation (NLG)*.[1] NLG is a major subfield of natural language processing, concerned with computing natural language sentences or texts that convey a given piece of information to an audience. In the sentence generation task, we focus on generating a single sentence that expresses a given meaning according to a given grammar. As a planning task, a plan encodes the necessary sentence with the actions in the plan corresponding to the utterance of individual words arising from a syntactically correct (and semantically appropriate) grammatical derivation (Koller and Stone 2007).

Sentence generation and tree-adjoining grammars Our version of the sentence generation problem makes use of a particular type of lexicalized grammar called a *tree-adjoining grammar (TAG)*, an example of which is shown in Figure 1. This grammar consists of *elementary trees* (i.e., the disjoint trees in the figure), each of which contributes certain *semantic content*. For instance, say that a knowledge base contains the individuals e, r_1 and r_2, and the facts that r_1 and r_2 are rabbits, r_1 is white and r_2 is brown, and e is an event in which r_1 sleeps. We could then construct a sentence expressing the information $\{\mathsf{sleep}(e, r_1)\}$ by combining instances of the elementary trees (in which the *semantic roles*, such as self and subj, have been substituted by constants from the knowledge base) into a derivation as shown in Figure 2. In the figure, the dashed arrow indicates a *substitution* operation which "plugs" an elementary tree into the leaf of another tree; the dotted arrows stand for *adjunction*, which splices an elementary tree into an internal node. We can then read off the sentence "The white rabbit sleeps" from the derivation. We note that in addition to grammatical correctness, the derivation must also include a *referring expression* that uniquely identifies r_1 to the hearer. The sentence "The rabbit sleeps" would not be appropriate since "the rabbit" could refer to either r_1 or r_2. In this case, r_2 is referred to as a *distractor* for the referring expression, i.e., an incorrect possible interpretation of the phrase.

[1] The name CRISP refers to an NLG system that attempts to solve the sentence generation problem using planning techniques (Koller and Stone 2007).

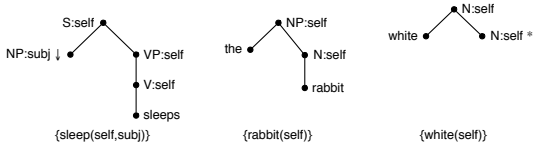

Figure 1: A grammar in the sentence generation domain.

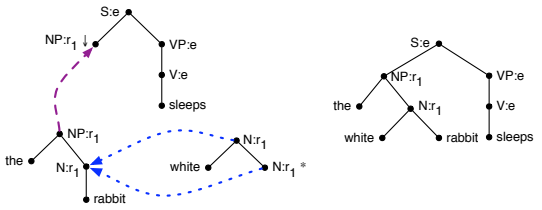

Figure 2: Derivation of "The white rabbit sleeps."

PDDL encoding To encode the sentence generation task as a PDDL planning domain, we treat each elementary tree in the tree-adjoining grammar as an individual planning operator denoting the addition of that tree to the current derivation. The planning problem in this case assumes an initial state containing an atom $\mathsf{subst}(S, \mathsf{root})$, encoding the fact that a sentence (S) must be generated starting at the node named root in the derivation tree, and a second atom $\mathsf{referent}(\mathsf{root}, e)$ which encodes the fact that the entire sentence describes the (event) individual e. The goal is characterised by a set of conditions that ensure all remaining derivations have been achieved ($\forall x \forall y. \neg \mathsf{subst}(x, y)$), no distractors remain ($\forall x \forall y. \neg \mathsf{distractor}(x, y)$), and that the resulting sentence contains the desired semantic content.

For instance, Figure 3 shows an example of the planning operators for the grammar in Figure 1. From the starting state, the action instance $\mathsf{add\text{-}sleeps}(\mathsf{root}, e, r_1)$ replaces the atom $\mathsf{subst}(S, \mathsf{root})$ with the atom $\mathsf{subst}(NP, \mathsf{n}_1)$. The action picks a fresh name for the newly introduced substitution node n_1 from a list of names $n_1, n_2, \ldots$ which is encoded in the initial state using the atoms current and next. At the same time, the operator records that the semantic information $\mathsf{sleep}(e, r_1)$ has now been expressed, and introduces all individuals except r_1 as distractors for the new referring expression at n_1. These distractors can then be removed by subsequent applications of the other two operators. Eventually we reach a goal

```
(:action add-sleeps
   :parameters (?u - node  ?u1 - node  ?un - node
                ?xself - individual  ?xsubj - individual)
   :precondition (and (subst S ?u)
                      (referent ?u ?xself)
                      (sleep ?xself ?xsubj)
                      (current ?u1) (next ?u1 ?un))
   :effect  (and (not (subst S ?u))
                 (expressed sleep ?xself ?xsubj)
                 (subst NP ?u1)
                 (referent ?u1 ?xsubj)
                 (not (current ?u1)) (current ?un)
                 (forall (?y - individual)
                    (when (not (= ?y ?xself))
                       (distractor ?u1 ?y)))))

(:action add-rabbit
   :parameters (?u - node  ?xself - individual)
   :precondition (and (subst NP ?u)
                      (referent ?u ?xself)
                      (rabbit ?xself))
   :effect  (and (not (subst NP ?u))
                 (canadjoin N ?u)
                 (forall (?y - individual)
                    (when (not (rabbit ?y))
                       (not (distractor ?u ?y))))))

(:action add-white
   :parameters (?u - node  ?xself - individual)
   :precondition (and (canadjoin N ?u)
                      (referent ?u ?xself)
                      (rabbit ?xself))
   :effect  (forall (?y - individual)
               (when (not (white ?y))
                  (not (distractor ?u ?y)))))
```

Figure 3: PDDL actions for the example grammar.

state where $\forall x \forall y. \neg \mathsf{subst}(x, y)$, $\forall x \forall y. \neg \mathsf{distractor}(x, y)$, and $\mathsf{expressed}(\mathsf{sleep}, e, r_1)$ all hold. In this case, we have a 3-step plan which performs the necessary derivation: $\mathsf{add\text{-}sleeps}(\mathsf{root}, e, r_1)$, $\mathsf{add\text{-}rabbit}(\mathsf{subj}(\mathsf{root}), r_1)$, $\mathsf{add\text{-}white}(\mathsf{subj}(\mathsf{root}), r_1)$. The sentence "The white rabbit sleeps" can be systematically reconstructed from this plan.

PDDL problem generators We have provided a generator that encodes instances of the sentence generation task as a planning problem in PDDL. The goal in each case is to generate a sentence of the form "S_1 but ... but S_n", where each S is of the form "the $\mathrm{adj}_1 \ldots \mathrm{adj}_d$ businessman admires the $\mathrm{adj}_1 \ldots \mathrm{adj}_d$ businessman". We can scale the generation problem instances along the following three dimensions:

1. the number n of sentences S that are conjoined;
2. the arity m of the verbs: $m = 1$ encodes the intransitive verb "sleeps", $m = 2$ the transitive verb "admires", and $m = 3$ the ditransitive verb "gives";
3. the number d of distractors for each referring expression, or alternatively, the number of adjectives adj_i that must be used in each referring expression to make it unique.

For each problem instance, our generator builds a generation problem instance using all lexicon entries for the relevant words from the XTAG Grammar (XTAG Research Group 2001), a large-scale TAG grammar of English. It then converts the generation problem instance into a planning problem in PDDL.

The PDDL problem generator is written in Java and invoked using the following command line:

```
java -jar CrispProblemGenerator.jar
        --pddl n m d prefix
```

Running the problem generator produces a pair of PDDL files, `prefix-domain-`n`-`m`-`d`.lisp` and `prefix-problem-`n`-`m`-`d`.lisp`, encoding the specified sentence generation problem.

Scalability and problem difficulty

In general, the problem of deciding whether a given communicative goal can be achieved with a given grammar is NP-complete (Koller and Striegnitz 2002): a naïve search algorithm that computes a derivation top-down takes exponential time. The parameters d, m, and n allow us to scale various linguistic dimensions of the problem freely. An increase in each parameter increases both the minimal plan length and the number of objects in the universe; both are $O(d \cdot m \cdot n)$. Note that the size of the search space grows exponentially both in d and in m, because the grammar allows us to express each noun phrase in many different ways.

Before the use of planning was proposed for this specific type of sentence generation problem, the state of the art in the NLG literature was to use greedy search to circumvent the combinatorial explosion (e.g., in the seminal SPUD system (Stone et al. 2003)). This is an incomplete search strategy which can make systems based on it unusable in practice. Using planning for the problem carries the promise of achieving complete search at reasonable efficiency, by employing modern search heuristics for planning, and ties in with the significant body of literature focused on using planning for other subproblems of NLG.

Interest for the deterministic track

The sentence generation domain poses interesting challenges for modern deterministic planners, both in terms of search and in terms of preprocessing.

Even small instances of the problem often give rise to state spaces with a large number of constants, high arity predicates, and complex action descriptions. A preliminary investigation using recent planners such as FF, Metric-FF, and SGPLAN6 was described in (Koller and Petrick 2008), followed by a more comprehensive study in (Koller and Petrick to appear). Overall, the results of these studies were mixed. While some planners performed well on certain problem instances (comparable to special-purpose techniques from the NLG community), other problems proved problematic. In particular, the reliance on grounding as a preprocessing technique often accounted for 90% of the overall runtime, or the inability to proceed past this stage on more challenging problem instances.

A more recent study in (Koller and Hoffmann 2010) specifically aimed at improving the preprocessing and search performance of FF on this domain. The domain exposed a number of bugs in FF's preprocessor for the first

time, which improved FF's preprocessor performance to acceptable levels. The study also showed that the search problem is hard for FF in that its Enforced Hill Climbing search heuristic effectively degenerates to breadth-first search in the CRISP domain, and FF does not always recognise dead ends. Concretely, at $m = 2$ and $d = 2$ even the optimised version of FF times out for $n > 3$, which is still not a huge problem instance in NLG. Given that FF is still a competitive planner, we conjecture that the domain will be similarly challenging for other planners.

The sentence generation task is also an example of a real research problem in NLG—a field which in the past has looked to the planning community as a source of tools, but more recently has focused on special purpose solutions to such problems. The CRISP domain therefore presents an opportunity to advance state-of-the-art techniques in both the NLG and deterministic planning communities.

References

Koller, A., and Hoffmann, J. 2010. Waking up a sleeping rabbit: On natural-language sentence generation with FF. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling*, 238–241.

Koller, A., and Petrick, R. 2008. Experiences with planning for natural language generation. In *Scheduling and Planning Applications woRKshop (SPARK 2008) at ICAPS 2008*.

Koller, A., and Petrick, R. P. A. to appear. Experiences with planning for natural language generation. *Computational Intelligence, Special Issue on Scheduling and Planning Applications*.

Koller, A., and Stone, M. 2007. Sentence generation as planning. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, 336–343.

Koller, A., and Striegnitz, K. 2002. Generation as dependency parsing. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 17–24.

Stone, M.; Doran, C.; Webber, B.; Bleam, T.; and Palmer, M. 2003. Microplanning with communicative intentions: The SPUD system. *Computational Intelligence* 19(4):311–381.

XTAG Research Group. 2001. A lexicalized tree adjoining grammar for english. Technical Report IRCS-01-03, IRCS, University of Pennsylvania. `ftp://ftp.cis.upenn.edu/pub/xtag/release-2.24.2001/tech-report.pdf`.