Sentence generation as a planning problem

Alexander Koller
Center for Computational Learning Systems
Columbia University
koller@cs.columbia.edu

Matthew Stone
Computer Science
Rutgers University
Matthew.Stone@rutgers.edu

Abstract

We translate sentence generation from TAG grammars with semantic and pragmatic information into a planning problem by encoding the contribution of each word declaratively and explicitly. This allows us to exploit the performance of off-the-shelf planners. It also opens up new perspectives on referring expression generation and the relationship between language and action.

1 Introduction

Systems that produce natural language must synthesize the primitives of linguistic structure into well-formed utterances that make desired contributions to discourse. This is fundamentally a planning problem: Each linguistic primitive makes certain contributions while potentially introducing new goals. In this paper, we make this perspective explicit by translating the sentence generation problem of TAG grammars with semantic and pragmatic information into a planning problem stated in the widely used Planning Domain Definition Language (PDDL, McDermott (2000)). The encoding provides a clean separation between computation and linguistic modelling and is open to future extensions. It also allows us to benefit from the past and ongoing advances in the performance of off-the-shelf planners (Blum and Furst, 1997; Kautz and Selman, 1998; Hoffmann and Nebel, 2001).

While there have been previous systems that encode generation as planning (Cohen and Perrault, 1979; Appelt, 1985; Heeman and Hirst, 1995), our approach is distinguished from these systems by its focus on the grammatically specified contributions of each individual word (and the TAG tree it anchors) to syntax, semantics, and local pragmatics (Hobbs et al., 1993). For example, words directly achieve content goals by adding a corresponding semantic primitive to the conversational record. We deliberately avoid reasoning about utterances as coordinated rational behavior, as earlier systems did; this allows us to get by with a much simpler logic.

The problem we solve encompasses the generation of referring expressions (REs) as a special case. Unlike some approaches (Dale and Reiter, 1995; Heeman and Hirst, 1995), we do not have to distinguish between generating NPs and expressions of other syntactic categories. We develop a new perspective on the lifecycle of a distractor, which allows us to generate more succinct REs by taking the rest of the utterance into account. More generally, we do not split the process of sentence generation into two separate steps of sentence planning and realization, as most other systems do, but solve the joint problem in a single integrated step. This can potentially allow us to generate higher-quality sentences. We share these advantages with systems such as SPUD (Stone et al., 2003).

Crucially, however, our approach describes the dynamics of interpretation explicitly and declaratively. We do not need to assume extra machinery beyond the encoding of words as PDDL planning operators; for example, our planning operators give a self-contained description of how each individual word contributes to resolving references. This makes our encoding more direct and transparent than those in work like Thomason and Hobbs (1997) and Stone et al. (2003).

We present our encoding in a sequence of steps, each of which adds more linguistic information to

Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics, pages 336–343,
Prague, Czech Republic, June 2007.

the planning operators. After a brief review of LTAG and PDDL, we first focus on syntax alone and show how to cast the problem of generating grammatically well-formed LTAG trees as a planning problem in Section 2. In Section 3, we add semantics to the elementary trees and add goals to communicate specific content (this corresponds to surface realization). We complete the account by modeling referring expressions and go through an example. Finally, we assess the practical efficiency of our approach and discuss future work in Section 4.

2 Grammaticality as planning

We start by reviewing the LTAG grammar formalism and giving an intuition of how LTAG generation is planning. We then add semantic roles to the LTAG elementary trees in order to distinguish different substitution nodes. Finally, we review the PDDL planning specification language and show how LTAG grammaticality can be encoded as a PDDL problem and how we can reconstruct an LTAG derivation from the plan.

2.1 Tree-adjoining grammars

The grammar formalism we use here is that of *lexicalized tree-adjoining grammars* (LTAG; Joshi and Schabes (1997)). An LTAG grammar consists of a finite set of lexicalized *elementary trees* as shown in Fig. 1a. Each elementary tree contains exactly one *anchor* node, which is labelled by a word. Elementary trees can contain *substitution nodes*, which are marked by down arrows (↓). Those elementary trees that are *auxiliary trees* also contain exactly one *foot node*, which is marked with an asterisk (∗). Trees that are not auxiliary trees are called *initial trees*.

Elementary trees can be combined by *substitution* and *adjunction* to form larger trees. Substitution is the operation of replacing a substitution node of some tree by another initial tree with the same root label. Adjunction is the operation of splicing an auxiliary tree into some node v of a tree, in such a way that the root of the auxiliary tree becomes the child of v's parent, and the foot node becomes the parent of v's children. If a node carries a *null adjunction constraint* (indicated by no-adjoin), no adjunction is allowed at this node; if it carries an *obligatory adjunction constraint* (indicated by adjoin!), an auxiliary tree *must* be adjoined there.

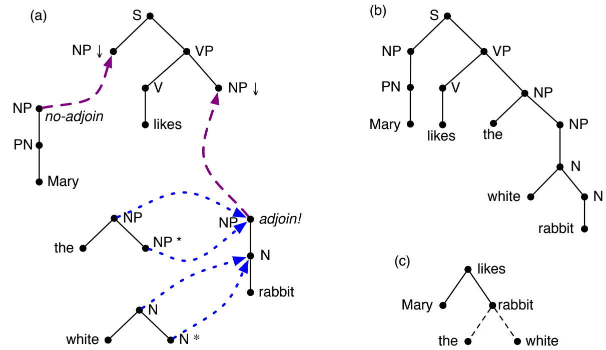

Figure 1: Building a derived (b) and a derivation tree (c) by combining elementary trees (a).

In Fig. 1a, we have combined some elementary trees by substitution (indicated by the dashed/magenta arrows) and adjunction (dotted/blue arrows). The result of these operations is the *derived tree* in Fig. 1b. The *derivation tree* in Fig. 1c represents the tree combination operations we used by having one node per elementary tree and drawing a solid edge if we combined the two trees by substitution, and a dashed edge for adjunctions.

2.2 The basic idea

Consider the process of constructing a derivation tree top-down. To build the tree in Fig. 1c, say, we start with the empty derivation tree and an obligation to generate an expression of category S. We satisfy this obligation by adding the tree for "likes" as the root of the derivation; but in doing so, we have introduced new unfilled substitution nodes of category NP, i.e. the derivation tree is not complete. We use the NP tree for "Mary" to fill one substitution node and the NP tree for "rabbit" to fill the other. This fills both substitution nodes, but the "rabbit" tree introduces an obligatory adjunction constraint, which we must satisfy by adjoining the auxiliary tree for "the". We now have a grammatical derivation tree, but we are free to continue by adding more auxiliary trees, such as the one for "white".

As we have just presented it, the generation of derivation trees is essentially a planning problem. A planning problem involves *states* and *actions* that can move from one state to another. The task is to find a sequence of actions that moves us from the *initial state* to a state that satisfies all the *goals*. In our case, the states are defined by the unfilled substitution nodes, the unsatisfied obligatory adjunction constraints, and the nodes that are available for adjunction in some (possibly incomplete) derivation tree. Each action adds a single elementary tree to the derivation, removing some of these "open nodes" while introducing new ones. The initial state is associated with the empty derivation tree and a requirement to generate an expression for the given root category. The goal is for the current derivation tree to be grammatically complete.

2.3 Semantic roles

Formalizing this intuition requires unique names for each node in the derived tree. Such names are necessary to distinguish the different open substitution nodes that still need to be filled, or the different available adjunction sites; in the example, the planner needed to be aware that "likes" introduces *two* separate NP substitution nodes to fill.

There are many ways to assign these names. One that works particularly well in the context of PDDL (as we will see below) is to assume that each node in an elementary tree, except for ones with null adjunction constraints, is marked with a *semantic role*, and that all substitution nodes are marked with different roles. Nothing hinges on the particular role inventory; here we assume an inventory including the roles ag for "agent" and pat for "patient". We also assume one special role self, which must be used for the root of each elementary tree and must never be used for substitution nodes.

We can now assign a unique name to every substitution node in a derived tree by assigning arbitrary but distinct *indices* to each use of an elementary tree, and giving the substitution node with role r in the elementary tree with index i the *identity* $i.r$. In the example, let's say the "likes" tree has index 1 and the semantic roles for the substitution nodes were ag and pat, respectively. The planner action that adds this tree would then require substitution of one NP with identity 1.ag and another NP with identity 1.pat; the "Mary" tree would satisfy the first requirement and the "rabbit" tree the second. If we assume that no elementary tree contains two internal nodes with the same category and role, we can refer to adjunction opportunities in a similar way.

Action `S-likes-1`(u). Precond: $\mathsf{subst}(\mathrm{S},u), \mathsf{step}(1)$
Effect: $\neg\mathsf{subst}(\mathrm{S},u), \mathsf{subst}(\mathrm{NP},1.\mathsf{ag}),$
$\mathsf{subst}(\mathrm{NP},1.\mathsf{pat}), \neg\mathsf{step}(1), \mathsf{step}(2)$

Action `NP-Mary-2`(u). Precond: $\mathsf{subst}(\mathrm{NP},u), \mathsf{step}(2)$
Effect: $\neg\mathsf{subst}(\mathrm{NP},u), \neg\mathsf{step}(2), \mathsf{step}(3)$

Action `NP-rabbit-3`(u). Precond: $\mathsf{subst}(\mathrm{NP},u), \mathsf{step}(3)$
Effect: $\neg\mathsf{subst}(\mathrm{NP},u), \mathsf{canadjoin}(\mathrm{NP},u),$
$\mathsf{mustadjoin}(\mathrm{NP},u), \neg\mathsf{step}(3), \mathsf{step}(4)$

Action `NP-the-4`(u). Precond: $\mathsf{canadjoin}(\mathrm{NP},u), \mathsf{step}(4)$
Effect: $\neg\mathsf{mustadjoin}(\mathrm{NP},u), \neg\mathsf{step}(4), \mathsf{step}(5)$

Figure 2: Some actions for the grammar in Fig. 1.

2.4 Encoding in PDDL

Now we are ready to encode the problem of generating grammatical LTAG derivation trees into PDDL. PDDL (McDermott, 2000) is the standard input language for modern planning systems. It is based on the well-known STRIPS language (Fikes and Nilsson, 1971). In this paradigm, a planning state is defined as a finite set of ground atoms of predicate logic that are true in this state; all other atoms are assumed to be false. Actions have a number of *parameters*, as well as a *precondition* and *effect*, both of which are logical formulas. When a planner tries to apply an action, it will first create an action *instance* by binding all parameters to constants from the domain. It must then verify that the precondition of the action instance is satisfied in the current state. If so, the action can be applied, in which case the effect is processed in order to change the state. In STRIPS, the precondition and effect both had to be conjunctions of atoms or negated atoms; positive effects are interpreted as making the atom true in the new state, and negative ones as making it false. PDDL permits numerous extensions to the formulas that can be used as preconditions and effects.

Each action in our planning problem encodes the effect of adding some elementary tree to the derivation tree. An initial tree with root category A translates to an action with a parameter u for the identity of the node that the current tree is substituted into. The action carries the precondition $\mathsf{subst}(A,u)$, and so can only be applied if u is an open substitution node in the current derivation with the correct category A. Auxiliary trees are analogous, but carry the precondition $\mathsf{canadjoin}(A,u)$. The effect of an initial tree is to remove the subst condition from the planning state (to record that the substitu-

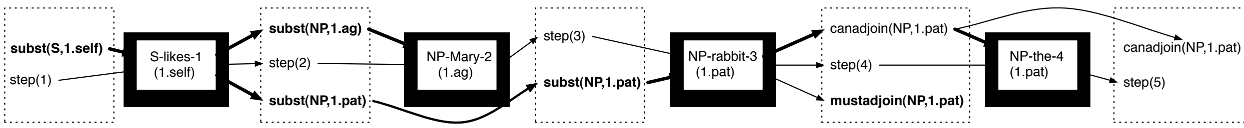

Figure 3: A plan for the actions in Fig. 2.

tion node u is now filled); an auxiliary tree has an effect $\neg\mathsf{mustadjoin}(A,u)$ to indicate that any obligatory adjunction constraint is satisfied but leaves the canadjoin condition in place to allow multiple adjunctions into the same node. In both cases, effects add subst, canadjoin and mustadjoin atoms representing the substitution nodes and adjunction sites that are introduced by the new elementary tree.

One remaining complication is that an action must assign new identities to the nodes it introduces; thus it must have access to a tree index that was not used in the derivation tree so far. We use the number of the current plan step as the index. We add an atom $\mathsf{step}(1)$ to the initial state of the planning problem, and we introduce k different copies of the actions for each elementary tree, where k is some upper limit on the plan size. These actions are identical, except that the i-th copy has an extra precondition $\mathsf{step}(i)$ and effects $\neg\mathsf{step}(i)$ and $\mathsf{step}(i+1)$. It is no restriction to assume an upper limit on the plan size, as most modern planners search for plans smaller than a given maximum length anyway.

Fig. 2 shows some of the actions into which the grammar in Fig. 1 translates. We display only one copy of each action and have left out most of the canadjoin effects. In addition, we use an initial state containing the atoms $\mathsf{subst}(S,1.\mathsf{self})$ and $\mathsf{step}(1)$ and a final state consisting of the following goal:

$$\forall A,u.\neg\mathsf{subst}(A,u) \wedge \forall A,u.\neg\mathsf{mustadjoin}(A,u).$$

We can then send the actions and the initial state and goal specifications to any off-the-shelf planner and obtain the plan in Fig. 3. The straight arrows in the picture link the actions to their preconditions and (positive) effects; the curved arrows indicate atoms that carry over from one state to the next without being changed by the action. Atoms are printed in boldface iff they contradict the goal.

This plan can be read as a derivation tree that has one node for each action instance in the plan, and an edge from node u to node v if u establishes a subst or canadjoin fact that is a precondition of v. These causal links are drawn as bold edges in Fig. 3. The mapping is unique for substitution edges because subst atoms are removed by every action that has them as their precondition. There may be multiple action instances in the plan that introduce the same atom $\mathsf{canadjoin}(A,u)$. In this case, we can freely choose one of these instances as the parent.

3 Sentence generation as planning

Now we extend this encoding to deal with semantics and referring expressions.

3.1 Communicative goals

In order to use the planner as a *surface realization* algorithm for TAG along the lines of Koller and Striegnitz (2002), we attach *semantic content* to each elementary tree and require that the sentence achieves a certain *communicative goal*. We also use a *knowledge base* that specifies the speaker's knowledge, and require that we can only use trees that express information in this knowledge base.

We follow Stone et al. (2003) in formalizing the semantic content of a lexicalized elementary tree t as a finite set of atoms; but unlike in earlier approaches, we use the semantic roles in t as the arguments of these atoms. For instance, the semantic content of the "likes" tree in Fig. 1 is $\{\mathsf{like}(\mathsf{self},\mathsf{ag},\mathsf{pat})\}$ (see also the semcon entries in Fig. 4). The knowledge base is some finite set of ground atoms; in the example, it could contain such entries as $\mathsf{like}(e,m,r)$ and $\mathsf{rabbit}(r)$. Finally, the communicative goal is some subset of the knowledge base, such as $\{\mathsf{like}(e,m,r)\}$.

We implement unsatisfied communicative goals as flaws that the plan must remedy. To this end, we add an atom $\mathsf{cg}(P,a_1,\dots,a_n)$ for each element $P(a_1,\dots,a_n)$ of the communicative goal to the initial state, and we add a corresponding conjunct $\forall P,x_1,\dots,x_n.\neg\mathsf{cg}(P,x_1,\dots,x_n)$ to the goal. In addition, we add an atom $\mathsf{skb}(P,a_1,\dots,a_n)$ to the initial state for each element $P(a_1,\dots,a_n)$ of the (speaker's) knowledge base.

We then add parameters $x_1, \ldots, x_n$ to each action with n semantic roles (including self). These new parameters are intended to be bound to individual constants in the knowledge base by the planner. For each elementary tree t and possible step index i, we establish the relationship between these parameters and the roles in two steps. First we fix a function id that maps the semantic roles of t to node identities. It maps self to u and each other role r to $i.r$. Second, we fix a function ref that maps the outputs of id bijectively to the parameters $x_1, \ldots, x_n$, in such a way that $\mathsf{ref}(u) = x_1$.

We can then capture the contribution of the i-th action for t to the communicative goal by giving it an effect $\neg\mathsf{cg}(P, \mathsf{ref}(\mathsf{id}(r_1)), \ldots, \mathsf{ref}(\mathsf{id}(r_n)))$ for each element $P(r_1, \ldots, r_n)$ of the elementary tree's semantic content. We restrict ourselves to only expressing true statements by giving the action a precondition $\mathsf{skb}(P, \mathsf{ref}(\mathsf{id}(r_1)), \ldots, \mathsf{ref}(\mathsf{id}(r_n)))$ for each element of the semantic content.

In order to keep track of the connection between node identities and individuals for future reference, each action gets an effect $\mathsf{referent}(\mathsf{id}(r), \mathsf{ref}(\mathsf{id}(r)))$ for each semantic role r except self. We enforce the connection between u and x_1 by adding a precondition $\mathsf{referent}(u, x_1)$.

In the example, the most interesting action in this respect is the one for the elementary tree for "likes". This action looks as follows:

Action `S-likes-1`(u, x_1, x_2, x_3).

Precond: $\mathsf{subst}(\mathrm{S}, u), \mathsf{step}(1), \mathsf{referent}(u, x_1),$
$\mathsf{skb}(\mathsf{like}, x_1, x_2, x_3)$

Effect: $\neg\mathsf{subst}(\mathrm{S}, u), \mathsf{subst}(\mathrm{NP}, 1.\mathsf{ag}), \mathsf{subst}(\mathrm{NP}, 1.\mathsf{pat}),$
$\neg\mathsf{step}(1), \mathsf{step}(2),$
$\mathsf{referent}(1.\mathsf{ag}, x_2), \mathsf{referent}(1.\mathsf{pat}, x_3),$
$\neg\mathsf{cg}(\mathsf{like}, x_1, x_2, x_3)$

We can run a planner and interpret the plan as above; the main difference is that complete plans not only correspond to grammatical derivation trees, but also express all communicative goals. Notice that this encoding models some aspects of lexical choice: The semantic content sets of the elementary trees need not be singletons, and so there may be multiple ways of partitioning the communicative goal into the content sets of various elementary trees.

3.2 Referring expressions

Finally, we extend the system to deal with the generation of referring expressions. While this problem is typically taken to require the generation of a noun phrase that refers uniquely to some individual, we don't need to make any assumptions about the syntactic category here. Moreover, we consider the problem in the wider context of generating referring expressions within a sentence, which can allow us to generate more succinct expressions.

Because a referring expression must allow the *hearer* to identify the intended referent uniquely, we keep track of the hearer's knowledge base separately. We use atoms $\mathsf{hkb}(P, a_1, \ldots, a_n)$, as with skb above. In addition, we assume *pragmatic* information of the form $\mathsf{pkb}(P, a_1, \ldots, a_n)$. The three pragmatic predicates that we will use here are hearer-new, indicating that the hearer does not know about the existence of an individual and can't infer it (Stone et al., 2003), hearer-old for the opposite, and contextset. The *context set* of an intended referent is the set of all individuals that the hearer might possibly confuse it with (DeVault et al., 2004). It is empty for hearer-new individuals. To say that b is in a's context set, we put the atom $\mathsf{pkb}(\mathsf{contextset}, a, b)$ into the initial state.

In addition to the semantic content, we equip every elementary tree in the grammar with a *semantic requirement* and a *pragmatic condition* (Stone et al., 2003). The semantic requirement is a set of atoms spelling out presuppositions of an elementary tree that can help the hearer identify what its arguments refer to. For instance, "likes" has the selectional restriction that its agent must be animate; thus the hearer will not consider inanimate individuals as distractors for the referring expression in agent position. The pragmatic condition is a set of atoms over the predicates in the pragmatic knowledge base.

In our setting, every substitution node that is introduced during the derivation introduces a new referring expression. This means that we can distinguish the referring expressions by the identity of the substitution node that introduced them. For each referring expression u (where u is a node identity), we keep track of the *distractors* in atoms of the form $\mathsf{distractor}(u, x)$. The presence of an atom $\mathsf{distractor}(u, a)$ in some planning state represents the fact that the current derivation tree is not yet informative enough to allow the hearer to identify the intended referent for u uniquely; a is another individual that is not the intended referent, but consistent with the partial referring expression we have constructed so far. We enforce uniqueness of all referring expressions by adding the conjunct $\forall u, x \neg \mathsf{distractor}(u,x)$ to the planning goal.

Now whenever an action introduces a new substitution node u, it will also introduce some distractor atoms to record the initial distractors for the referring expression at u. An individual a is in the initial distractor set for the substitution node with role r if (a) it is not the intended referent, (b) it is in the context set of the intended referent, and (c) there is a choice of individuals for the other parameters of the action that satisfies the semantic requirement together with a. This is expressed by adding the following effect for each substitution node; the conjunction is over the elements $P(r_1, \ldots, r_n)$ of the semantic requirement, and there is one universal quantifier for y and for each parameter x_j of the action except for $\mathsf{ref}(\mathsf{id}(r))$.

$$\begin{array}{l}\forall y, x_1, \ldots, x_n \\ \quad (y \neq \mathsf{ref}(\mathsf{id}(r)) \wedge \mathsf{pkb}(\mathsf{contextset}, \mathsf{ref}(\mathsf{id}(r)), y) \wedge \\ \quad \bigwedge \mathsf{hkb}(P, \mathsf{ref}(\mathsf{id}(r_1)), \ldots, \mathsf{ref}(\mathsf{id}(r_n)))[y/\mathsf{ref}(\mathsf{id}(r))]) \\ \quad \rightarrow \mathsf{distractor}(\mathsf{id}(r), y)\end{array}$$

On the other hand, a distractor a for a referring expression introduced at u is removed when we substitute or adjoin an elementary tree into u which rules a out. For instance, the elementary tree for "rabbit" will remove all non-rabbits from the distractor set of the substitution node into which it is substituted. We achieve this by adding the following effect to each action; here the conjunction is over all elements of the semantic content.

$$\begin{array}{l}\forall y. (\neg \bigwedge \mathsf{hkb}(P, \mathsf{ref}(\mathsf{id}(r_1)), \ldots, \mathsf{ref}(\mathsf{id}(r_n))))[y/x_1] \\ \quad \rightarrow \neg \mathsf{distractor}(u, y),\end{array}$$

Finally, each action gets its pragmatic condition as a precondition.

3.3 The example

By way of example, Fig. 5 shows the full versions of the actions from Fig. 2, for the extended grammar in Fig. 4. Let's say that the hearer knows about two rabbits r (which is white) and r' (which is not), about a person m with the name Mary, and about an event e, and that the context set of r is $\{r, r', m, e\}$. Let's also say that our communicative goal is $\{\mathsf{like}(e, m, r)\}$. In this case, the first action instance in Fig. 3, `S-likes-1`$(1.\mathsf{self}, e, m, r)$, introduces a substitution node with identity `1.pat`. The

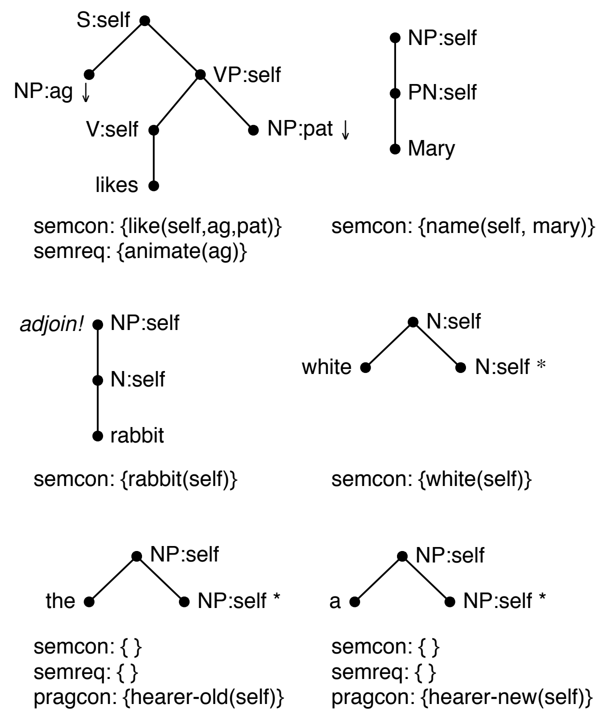

Figure 4: The extended example grammar.

initial distractor set of this node is $\{r', m\}$ – the set of all individuals in r's context set except for inanimate objects (which violate the semantic requirement) and r itself. The `NP-rabbit-3` action removes m from the distractor set, but at the end of the plan in Fig. 3, r' is still a distractor, i.e. we have not reached a goal state. We can complete the plan by performing a final action `NP-white-5`$(1.\mathsf{pat}, r)$, which will remove this distractor and achieve the planning goal. We can still reconstruct a derivation tree from the complete plan literally as described in Section 2.

Now let's say that the hearer did not know about the existence of the individual r before the utterance we are generating. We model this by marking r as hearer-new in the pragmatic knowledge base and assigning it an empty context set. In this case, the referring expression `1.pat` would be initialized with an empty distractor set. This entitles us to use the action `NP-a-4` and generate the four-step plan corresponding to the sentence "Mary likes a rabbit."

4 Discussion and future work

In conclusion, let's look in more detail at computational issues and the role of mutually constraining referring expressions.

Action $\texttt{S-likes-1}(u,x_1,x_2,x_3)$.
Precond: $\mathsf{referent}(u,x_1), \mathsf{skb}(\mathsf{like},x_1,x_2,x_3), \mathsf{subst}(\mathrm{S},u), \mathsf{step}(1)$
Effect: $\neg\mathsf{cg}(\mathsf{like},x_1,x_2,x_3), \neg\mathsf{subst}(\mathrm{S},u), \neg\mathsf{step}(1), \mathsf{step}(2), \mathsf{subst}(\mathrm{NP},1.\mathrm{ag}), \mathsf{subst}(\mathrm{NP},1.\mathrm{pat})$,
$\forall y.\neg\mathsf{hkb}(\mathsf{like},y,x_2,x_3) \to \neg\mathsf{distractor}(u,y)$,
$\forall y,x_1,x_3.x_2 \neq y \wedge \mathsf{pkb}(\mathsf{contextset},x_2,y) \wedge \mathsf{animate}(y) \to \mathsf{distractor}(1.\mathrm{ag},y)$,
$\forall y,x_1,x_2.x_3 \neq y \wedge \mathsf{pkb}(\mathsf{contextset},x_3,y) \to \mathsf{distractor}(1.\mathrm{pat},y)$

Action $\texttt{NP-Mary-2}(u,x_1)$.
Precond: $\mathsf{referent}(u,x_1), \mathsf{skb}(\mathsf{name},x_1,\mathsf{mary})$,
$\mathsf{subst}(\mathrm{NP},u), \mathsf{step}(2)$
Effect: $\neg\mathsf{cg}(\mathsf{name},x_1,\mathsf{mary}), \neg\mathsf{subst}(\mathrm{NP},u)$,
$\neg\mathsf{step}(2), \mathsf{step}(3)$,
$\forall y.\neg\mathsf{hkb}(\mathsf{name},y,\mathsf{mary}) \to \neg\mathsf{distractor}(u,y)$

Action $\texttt{NP-rabbit-3}(u,x_1)$.
Precond: $\mathsf{referent}(u,x_1), \mathsf{skb}(\mathsf{rabbit},x_1)$,
$\mathsf{subst}(\mathrm{N},u), \mathsf{step}(3)$
Effect: $\neg\mathsf{cg}(\mathsf{rabbit},x_1), \neg\mathsf{subst}(\mathrm{N},u), \neg\mathsf{step}(3), \mathsf{step}(4)$,
$\mathsf{canadjoin}(\mathrm{NP},u), \mathsf{mustadjoin}(\mathrm{NP},u)$,
$\forall y.\neg\mathsf{hkb}(\mathsf{rabbit},y) \to \neg\mathsf{distractor}(u,y)$

Action $\texttt{NP-the-4}(u,x_1)$.
Precond: $\mathsf{referent}(u,x_1), \mathsf{canadjoin}(\mathrm{NP},u), \mathsf{step}(4)$,
$\mathsf{pkb}(\mathsf{hearer\text{-}old},x_1)$
Effect: $\neg\mathsf{mustadjoin}(\mathrm{NP},u), \neg\mathsf{step}(4), \mathsf{step}(5)$

Action $\texttt{NP-a-4}(u,x_1)$.
Precond: $\mathsf{referent}(u,x_1), \mathsf{canadjoin}(\mathrm{NP},u), \mathsf{step}(4)$,
$\mathsf{pkb}(\mathsf{hearer\text{-}new},x_1)$
Effect: $\neg\mathsf{mustadjoin}(\mathrm{NP},u), \neg\mathsf{step}(4), \mathsf{step}(5)$

Action $\texttt{NP-white-5}(u,x_1)$.
Precond: $\mathsf{referent}(u,x_1), \mathsf{skb}(\mathsf{white},x_1), \mathsf{canadjoin}(\mathrm{NP},u), \mathsf{step}(5)$
Effect: $\neg\mathsf{cg}(\mathsf{white},x_1), \neg\mathsf{mustadjoin}(\mathrm{NP},u), \neg\mathsf{step}(5), \mathsf{step}(6)$,
$\forall y.\neg\mathsf{hkb}(\mathsf{white},y) \to \neg\mathsf{distractor}(u,y)$

Figure 5: Some of the actions corresponding to the grammar in Fig. 4.

4.1 Computational issues

We lack the space to present the formal definition of the sentence generation problem we encode into PDDL. However, this problem is NP-complete, by reduction of Hamiltonian Cycle – unsurprisingly, given that it encompasses realization, and the very similar realization problem in Koller and Striegnitz (2002) is NP-hard. So any algorithm for our problem must be prepared for exponential runtimes.

We have implemented the translation described in this paper and experimented with a number of different grammars, knowledge bases, and planners. The FF planner (Hoffmann and Nebel, 2001) can compute the plans in Section 3.3 in under 100 ms using the grammar in Fig. 4. If we add 10 more lexicon entries to the grammar, the runtime grows to 190 ms; and for 20 more entries, to 360 ms. The runtime also grows with the plan length: It takes 410 ms to generate a sentence "Mary likes the Adj ... Adj rabbit" with four adjectives and 890 ms for six adjectives, corresponding to a plan length of 10. We compared these results against a planning-based reimplementation of SPUD's greedy search heuristic (Stone et al., 2003). This system is faster than FF for small inputs (360 ms for four adjectives), but becomes slower as inputs grow larger (1000 ms for six adjectives); but notice that while FF is also a heuristic planner, it is guaranteed to find a solution if one exists, unlike SPUD.

Planners have made tremendous progress in efficiency in the past decade, and by encoding sentence generation as a planning problem, we are set to profit from any future improvements; it is an advantage of the planning approach that we can compare very different search strategies like FF's and SPUD's in the same framework. However, our PDDL problems are challenging for modern planners because most planners start by computing all instances of atoms and actions. In our experiments, FF generally spent only about 10% of the runtime on search and the rest on computing the instances; that is, there is a lot of room for optimization. For larger grammars and knowledge bases, the number of instances can easily grow into the billions. In future work, we will therefore collaborate with experts on planning systems to compute action instances only by need.

4.2 Referring expressions

In our analysis of referring expressions, the tree t that introduces the new substitution nodes typically initializes the distractor sets with proper subsets of the entire domain. This allows us to generate succinct descriptions by encoding t's presuppositions as semantic requirements, and localizes the interactions between the referring expressions generated for different substitution nodes within t's action.

However, an important detail in the encoding of referring expressions above is that an individual a counts as a distractor for the role r if there is any tuple of values that satisfies the semantic requirement and has a in the r-component. This is correct, but can sometimes lead to overly complicated referring expressions. An example is the construction "X takes Y from Z", which presupposes that Y is in Z. In a scenario that involves multiple rabbits, multiple hats, and multiple individuals that are inside other individuals, but only one pair of a rabbit r inside a hat h, the expression "X takes the rabbit from the hat" is sufficient to refer uniquely to r and h (Stone and Webber, 1998). Our system would try to generate an expression for Y that suffices by itself to distinguish r from all distractors, and similarly for Z. We will explore this issue further in future work.

5 Conclusion

In this paper, we have shown how sentence generation with TAG grammars and semantic and pragmatic information can be encoded into PDDL. Our encoding is declarative in that it can be used with any correct planning algorithm, and explicit in that the actions capture the complete effect of a word on the syntactic, semantic, and local pragmatic goals. In terms of expressive power, it captures the core of SPUD, except for its inference capabilities.

This work is practically relevant because it opens up the possibility of using efficient planners to make generators faster and more flexible. Conversely, our PDDL problems are a challenge for current planners and open up NLG as an application domain that planning research itself can target.

Theoretically, our encoding provides a new framework for understanding and exploring the general relationships between language and action. It suggests new ways of going beyond SPUD's expressive power, to formulate utterances that describe and disambiguate concurrent real-world actions or exploit the dynamics of linguistic context within and across sentences.

Acknowledgments. This work was funded by a DFG research fellowship and the NSF grants HLC 0308121, IGERT 0549115, and HSD 0624191. We are indebted to Henry Kautz for his advice on planning systems, and to Owen Rambow, Bonnie Webber, and the anonymous reviewers for feedback.

References

D. Appelt. 1985. *Planning English Sentences*. Cambridge University Press, Cambridge England.

A. Blum and M. Furst. 1997. Fast planning through graph analysis. *Artificial Intelligence*, 90:281–300.

P. R. Cohen and C. R. Perrault. 1979. Elements of a plan-based theory of speech acts. *Cognitive Science*, 3(3):177–212.

R. Dale and E. Reiter. 1995. Computational interpretations of the Gricean maxims in the generation of referring expressions. *Cognitive Science*, 19.

D. DeVault, C. Rich, and C. Sidner. 2004. Natural language generation and discourse context: Computing distractor sets from the focus stack. In *Proc. FLAIRS*.

R. Fikes and N. Nilsson. 1971. STRIPS: A new approach in the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208.

P. Heeman and G. Hirst. 1995. Collaborating on referring expressions. *Computational Linguistics*, 21(3):351–382.

J. Hobbs, M. Stickel, D. Appelt, and P. Martin. 1993. Interpretation as abduction. *Artificial Intelligence*, 63:69–142.

J. Hoffmann and B. Nebel. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14.

A. Joshi and Y. Schabes. 1997. Tree-Adjoining Grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, chapter 2, pages 69–123. Springer-Verlag, Berlin.

H. Kautz and B. Selman. 1998. Blackbox: A new approach to the application of theorem proving to problem solving. In *Workshop Planning as Combinatorial Search, AIPS-98*.

A. Koller and K. Striegnitz. 2002. Generation as dependency parsing. In *Proc. 40th ACL*, Philadelphia.

D. V. McDermott. 2000. The 1998 AI Planning Systems Competition. *AI Magazine*, 21(2):35–55.

M. Stone and B. Webber. 1998. Textual economy through close coupling of syntax and semantics. In *Proc. INLG*.

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

R. Thomason and J. Hobbs. 1997. Interrelating interpretation and generation in an abductive framework. In *AAAI Fall Symposium on Communicative Action*.