Automated planning for situated natural language generation

Konstantina Garoufi and **Alexander Koller**
Cluster of Excellence “Multimodal Computing and Interaction”
Saarland University, Saarbrücken, Germany
{garoufi,koller}@mmci.uni-saarland.de

Abstract

We present a natural language generation approach which models, exploits, and manipulates the non-linguistic context in situated communication, using techniques from AI planning. We show how to generate instructions which deliberately guide the hearer to a location that is convenient for the generation of simple referring expressions, and how to generate referring expressions with context-dependent adjectives. We implement and evaluate our approach in the framework of the Challenge on Generating Instructions in Virtual Environments, finding that it performs well even under the constraints of real-time generation.

1 Introduction

The problem of situated natural language generation (NLG)—i.e., of generating natural language in the context of a physical (or virtual) environment—has received increasing attention in the past few years. On the one hand, this is because it is the foundation of various emerging applications, including human-robot interaction and mobile navigation systems, and is the focus of a current evaluation effort, the Challenges on Generating Instructions in Virtual Environments (GIVE; (Koller et al., 2010b)). On the other hand, situated generation comes with interesting theoretical challenges: Compared to the generation of pure text, the interpretation of expressions in situated communication is sensitive to the non-linguistic context, and this context can change as easily as the user can move around in the environment.

One interesting aspect of situated communication from an NLG perspective is that this non-linguistic context can be manipulated by the speaker. Consider the following segment of discourse between an instruction giver (IG) and an instruction follower (IF), which is adapted from the SCARE corpus (Stoia et al., 2008):

(1) IG: *Walk forward and then turn right.*
IF: (walks and turns)
IG: *OK. Now hit the button in the middle.*

In this example, the IG plans to refer to an object (here, a button); and in order to do so, gives a navigation instruction to guide the IF to a convenient location at which she can then use a simple referring expression (RE). That is, there is an interaction between navigation instructions (intended to manipulate the non-linguistic context in a certain way) and referring expressions (which exploit the non-linguistic context). Although such subdialogues are common in SCARE, we are not aware of any previous research that can generate them in a computationally feasible manner.

This paper presents an approach to generation which is able to model the effect of an utterance on the non-linguistic context, and to intentionally generate utterances such as the above as part of a process of referring to objects. Our approach builds upon the CRISP generation system (Koller and Stone, 2007), which translates generation problems into planning problems and solves these with an AI planner. We extend the CRISP planning operators with the perlocutionary effects that uttering a particular word has on the physical environment if it is understood correctly; more specifically, on the position and orientation of the hearer. This allows the planner to predict the non-linguistic context in which a later part of the utterance will be interpreted, and therefore to search for contexts that allow the use of simple REs. As a result, the work of referring to an object gets distributed over multiple utterances of low cognitive load rather than a single complex noun phrase.

A second contribution of our paper is the generation of REs involving context-dependent adjectives: A button can be described as “the left blue

Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, pages 1573–1582,
Uppsala, Sweden, 11-16 July 2010. ©2010 Association for Computational Linguistics

button" even if there is a red button to its left. We model adjectives whose interpretation depends on the nominal phrases they modify, as well as on the non-linguistic context, by keeping track of the distractors that remain after uttering a series of modifiers. Thus, unlike most other RE generation approaches, we are not restricted to building an RE by simply intersecting lexically specified sets representing the extensions of different attributes, but can correctly generate expressions whose meaning depends on the context in a number of ways. In this way we are able to refer to objects earlier and more flexibly.

We implement and evaluate our approach in the context of a GIVE NLG system, by using the GIVE-1 software infrastructure and a GIVE-1 evaluation world. This shows that our system generates an instruction-giving discourse as in (1) in about a second. It outperforms a mostly non-situated baseline significantly, and compares well against a second baseline based on one of the top-performing systems of the GIVE-1 Challenge. Next to the practical usefulness this evaluation establishes, we argue that our approach to jointly modeling the grammatical and physical effects of a communicative action can also inform new models of the pragmatics of speech acts.

Plan of the paper. We discuss related work in Section 2, and review the CRISP system, on which our work is based, in Section 3. We then show in Section 4 how we extend CRISP to generate navigation-and-reference discourses as in (1), and add context-dependent adjectives in Section 5. We evaluate our system in Section 6; Section 7 concludes and points to future work.

2 Related work

The research reported here can be seen in the wider context of approaches to generating referring expressions. Since the foundational work of Dale and Reiter (1995), there has been a considerable amount of literature on this topic. Our work departs from the mainstream in two ways. First, it exploits the situated communicative setting to deliberately modify the context in which an RE is generated. Second, unlike most other RE generation systems, we allow the contribution of a modifier to an RE to depend both on the context and on the rest of the RE.

We are aware of only one earlier study on generation of REs with focus on interleaving navigation and referring (Stoia et al., 2006). In this machine learning approach, Stoia et al. train classifiers that signal when the context conditions (e.g. visibility of target and distractors) are appropriate for the generation of an RE. This method can be then used as part of a content selection component of an NLG system. Such a component, however, can only inform a system on whether to choose navigation over RE generation at a given point of the discourse, and is not able to help it decide what kind of navigational instructions to generate so that subsequent REs become simple.

To our knowledge, the only previous research on generating REs with context-dependent modifiers is van Deemter's (2006) algorithm for generating vague adjectives. Unlike van Deemter, we integrate the RE generation process tightly with the syntactic realization, which allows us to generate REs with more than one context-dependent modifier and model the effect of their linear order on the meaning of the phrase. In modeling the context, we focus on the non-linguistic context and the influence of each of the RE's words; this is in contrast to previous research on context-sensitive generation of REs, which mainly focused on the discourse context (Krahmer and Theune, 2002). Our interpretation of context-dependent modifiers picks up ideas by Kamp and Partee (1995) and implements them in a practical system, while our method of ordering modifiers is linguistically informed by the class-based paradigm (e.g., Mitchell (2009)).

On the other hand, our work also stands in a tradition of NLG research that is based on AI planning. Early approaches (Perrault and Allen, 1980; Appelt, 1985) provided compelling intuitions for this connection, but were not computationally viable. The research we report here can be seen as combining Appelt's idea of using planning for sentence-level NLG with a computationally benign variant of Perrault et al.'s approach of modeling the intended perlocutionary effects of a speech act as the effects of a planning operator. Our work is linked to a growing body of very recent work that applies modern planning research to various problems in NLG (Steedman and Petrick, 2007; Brenner and Kruijff-Korbayová, 2008; Benotti, 2009). It is directly based on Koller and Stone's (2007) reimplementation of the SPUD generator (Stone et al., 2003) with planning. As far as we know, ours is the first system in the SPUD tradi-

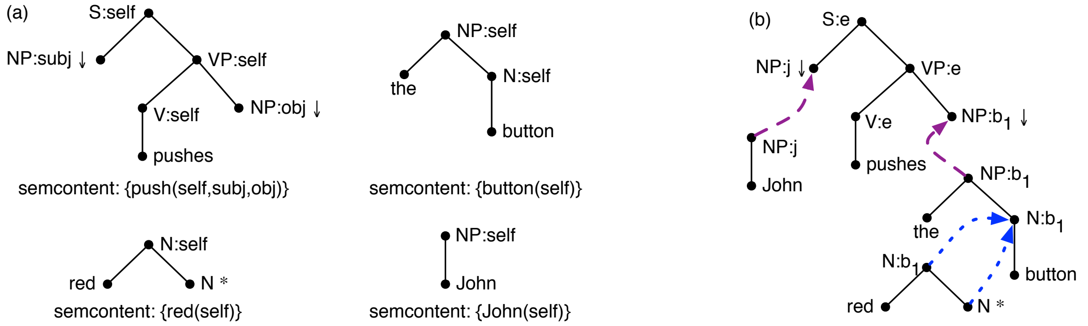


Figure 1: (a) An example grammar; (b) a derivation of “John pushes the red button” using (a).

tion that explicitly models the context change effects of an utterance.

While nothing in our work directly hinges on this, we implemented our approach in the context of an NLG system for the GIVE Challenge (Koller et al., 2010b), that is, as an instruction giving system for virtual worlds. This makes our system comparable with other approaches to instruction giving implemented in the GIVE framework.

3 Sentence generation as planning

Our work is based on the CRISP system (Koller and Stone, 2007), which encodes sentence generation with tree-adjoining grammars (TAG; (Joshi and Schabes, 1997)) as an AI planning problem and solves that using efficient planners. It then decodes the resulting plan into a TAG derivation, from which it can read off a sentence. In this section, we briefly recall how this works. For space reasons, we will present primarily examples instead of definitions.

3.1 TAG sentence generation

The CRISP generation problem (like that of SPUD (Stone et al., 2003)) assumes a lexicon of entries consisting of a TAG elementary tree annotated with semantic and pragmatic information. An example is shown in Fig. 1a. In addition to the elementary tree, each lexicon entry specifies its *semantic content* and possibly a *semantic requirement*, which can express certain presuppositions triggered by this entry. The nodes in the tree may be labeled with argument names such as *semantic roles*, which specify the participants in the relation expressed by the lexicon entry; in the example, every entry uses the semantic role self representing the event or individual itself, and the entry for “pushes” furthermore uses subj and obj for the subject and object argument, respectively. We combine here for simplicity the entries for “the” and “button” into “the button”.

For generation, we assume as input a knowledge base and a communicative goal in addition to the grammar. The goal is to compute a derivation that expresses the communicative goal in a sentence that is grammatically correct and complete; whose meaning is justified by the knowledge base; and in which all REs can be resolved to unique individuals in the world by the hearer. Let’s say, for example, that we have a knowledge base $\{\mathsf{push}(e, \mathsf{j}, b_1), \mathsf{John}(\mathsf{j}), \mathsf{button}(b_1), \mathsf{button}(b_2), \mathsf{red}(b_1)\}$. Then we can combine instances of the trees for “John”, “pushes”, and “the button” into a grammatically complete derivation. However, because both b_1 and b_2 satisfy the semantic content of “the button”, we must adjoin “red” into the derivation to make the RE refer uniquely to b_1. The complete derivation is shown in Fig. 1b; we can read off the output sentence “John pushes the red button” from the leaves of the derived tree we build in this way.

3.2 TAG generation as planning

In the CRISP system, Koller and Stone (2007) show how this generation problem can be solved by converting it into a planning problem (Nau et al., 2004). The basic idea is to encode the partial derivation in the planning state, and to encode the action of adding each elementary tree in the planning operators. The encoding of our example as a planning problem is shown in Fig. 2.

In the example, we start with an initial state which contains the entire knowledge base, plus atoms $\mathsf{subst}(\mathrm{S}, \mathsf{root})$ and $\mathsf{ref}(\mathsf{root}, e)$ expressing that we want to generate a sentence about the event e. We can then apply the (instantiated) action **pushes**$(\mathsf{root}, \mathsf{n}_1, \mathsf{n}_2, \mathsf{n}_3, e, \mathsf{j}, b_1)$, which models the act of substituting the elementary tree for “pushes”

pushes$(u, u_1, u_2, u_n, x, x_1, x_2)$:
Precond: $\mathsf{subst}(\mathrm{S}, u), \mathsf{ref}(u, x), \mathsf{push}(x, x_1, x_2),$
$\mathsf{current}(u_1), \mathsf{next}(u_1, u_2), \mathsf{next}(u_2, u_n)$
Effect: $\neg\mathsf{subst}(\mathrm{S}, u), \mathsf{subst}(\mathrm{NP}, u_1), \mathsf{subst}(\mathrm{NP}, u_2),$
$\mathsf{ref}(u_1, x_1), \mathsf{ref}(u_2, x_2), \forall y.\mathsf{distractor}(u_1, y),$
$\forall y.\mathsf{distractor}(u_2, y)$

John(u, x):
Precond: $\mathsf{subst}(\mathrm{NP}, u), \mathsf{ref}(u, x), \mathsf{John}(x)$
Effect: $\neg\mathsf{subst}(\mathrm{NP}, u), \forall y.\neg\mathsf{John}(y) \rightarrow \neg\mathsf{distractor}(u, y)$

the-button(u, x):
Precond: $\mathsf{subst}(\mathrm{NP}, u), \mathsf{ref}(u, x), \mathsf{button}(x)$
Effect: $\neg\mathsf{subst}(\mathrm{NP}, u), \mathsf{canadjoin}(\mathrm{N}, u),$
$\forall y.\neg\mathsf{button}(y) \rightarrow \neg\mathsf{distractor}(u, y)$

red(u, x):
Precond: $\mathsf{canadjoin}(\mathrm{N}, u), \mathsf{ref}(u, x), \mathsf{red}(x)$
Effect: $\forall y.\neg\mathsf{red}(y) \rightarrow \neg\mathsf{distractor}(u, y)$

Figure 2: CRISP planning operators for the elementary trees in Fig. 1.

into the substitution node root: It can only be applied because root is an unfilled substitution node (precondition $\mathsf{subst}(\mathrm{S}, \mathsf{root})$), and its effect is to remove $\mathsf{subst}(\mathrm{S}, \mathsf{root})$ from the planning state while adding two new atoms $\mathsf{subst}(\mathrm{NP}, \mathsf{n}_1)$ and $\mathsf{subst}(\mathrm{NP}, \mathsf{n}_2)$ for the substitution nodes of the "pushes" tree. The planning state maintains information about which individual each node refers to in the ref atoms. The current and next atoms are needed to select unused names for newly introduced syntax nodes.[1] Finally, the action introduces a number of distractor atoms including $\mathsf{distractor}(\mathsf{n}_2, e)$ and $\mathsf{distractor}(\mathsf{n}_2, b_2)$, expressing that the RE at n_2 can still be misunderstood by the hearer as e or b_2.

In this new state, all subst and distractor atoms for n_1 can be eliminated with the action **John**$(\mathsf{n}_1, \mathsf{j})$. We can also apply the action **the-button**(n_2, b_1) to eliminate $\mathsf{subst}(\mathrm{NP}, \mathsf{n}_2)$ and $\mathsf{distractor}(\mathsf{n}_2, e)$, since e is not a button. However $\mathsf{distractor}(\mathsf{n}_2, b_2)$ remains. Now because the action **the-button** also introduced the atom $\mathsf{canadjoin}(\mathrm{N}, \mathsf{n}_2)$, we can remove the final distractor atom by applying **red**(n_2, b_1). This brings us into a goal state, and we are done. Goal states in CRISP planning problems are characterized by axioms such as $\forall A \forall u.\neg\mathsf{subst}(A, u)$ (encoding grammatical completeness) and $\forall u \forall x.\neg\mathsf{distractor}(u, x)$ (requiring unique reference).

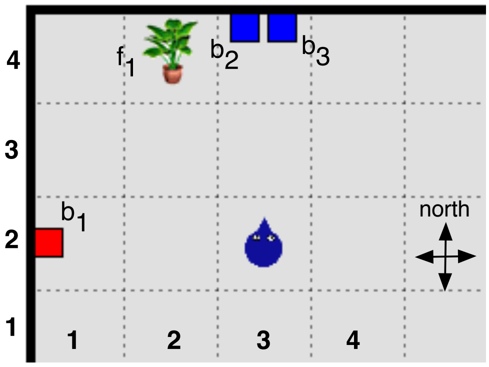


Figure 3: An example map for instruction giving.

3.3 Decoding the plan

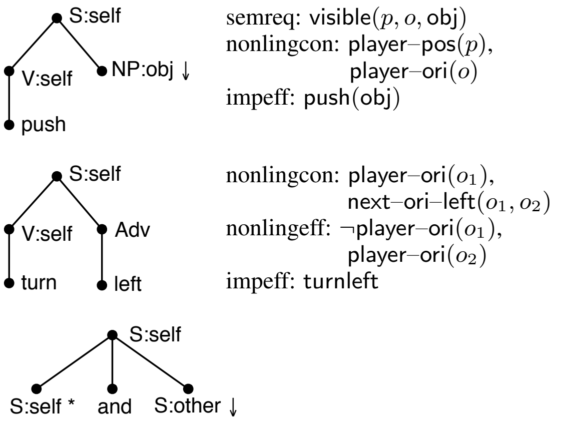
An AI planner such as FF (Hoffmann and Nebel, 2001) can compute a plan for a planning problem that consists of the planning operators in Fig. 2 and a specification of the initial state and the goal. We can then decode this plan into the TAG derivation shown in Fig. 1b. The basic idea of this decoding step is that an action with a precondition $\mathsf{subst}(A, u)$ fills the substitution node u, while an action with a precondition $\mathsf{canadjoin}(A, u)$ adjoins into a node of category A in the elementary tree that was substituted into u. CRISP allows multiple trees to adjoin into the same node. In this case, the decoder executes the adjunctions in the order in which they occur in the plan.

4 Context manipulation

We are now ready to describe our NLG approach, SCRISP ("Situated CRISP"), which extends CRISP to take the non-linguistic context of the generated utterance into account, and deliberately manipulate it to simplify RE generation.

As a simplified version of our introductory instruction giving example (1), consider the map in Fig. 3. The instruction follower (IF), who is located on the map at position $\mathsf{pos}_{3,2}$ facing north, sees the scene from the first-person perspective as in Fig. 7. Now an instruction giver (IG) could instruct the IF to press the button b_1 in this scene by saying "push the button on the wall to your left". Interpreting this instruction is difficult for the IF because it requires her to either memorize the RE until she has turned to see the button, or to perform a mental rotation task to visualize b_1 internally. Alternatively, the IG can first instruct the IF to "turn left"; once the IF has done this, the IG can then simply say "now push the button in front

[1] This is a different solution to the name-selection problem than in Koller and Stone (2007). It is simpler and improves computational efficiency.



semreq: visible(p, o, obj)
nonlingcon: player–pos(p),
player–ori(o)
impeff: push(obj)

nonlingcon: player–ori(o_1),
next–ori–left(o_1, o_2)
nonlingeff: ¬player–ori(o_1),
player–ori(o_2)
impeff: turnleft

Figure 4: An example SCRISP lexicon.

of you". This lowers the cognitive load on the IF, and presumably improves the rate of correctly interpreted REs.

SCRISP is capable of deliberately generating such context-changing navigation instructions. The key idea of our approach is to extend the CRISP planning operators with preconditions and effects that describe the (simulated) physical environment: A "turn left" action, for example, modifies the IF's orientation in space and changes the set of visible objects; a "push" operator can then pick up this changed set and restrict the distractors of the forthcoming RE it introduces (i.e. "the button") to only objects that are visible in the changed context. We also extend CRISP to generate imperative rather than declarative sentences.

4.1 Situated CRISP

We define a lexicon for SCRISP to be a CRISP lexicon in which every lexicon entry may also describe *non-linguistic conditions*, *non-linguistic effects* and *imperative effects*. Each of these is a set of atoms over constants, semantic roles, and possibly some free variables. Non-linguistic conditions specify what must be true in the world so a particular instance of a lexicon entry can be uttered felicitously; non-linguistic effects specify what changes uttering the word brings about in the world; and imperative effects contribute to the IF's "to-do list" (Portner, 2007) by adding the properties they denote.

A small lexicon for our example is shown in Fig. 4. This lexicon specifies that saying "push X" puts pushing X on the IF's to-do list, and carries the presupposition that X must be visible from the location where "push X" is uttered; this reflects our simplifying assumption that the IG can only refer to objects that are currently visible. Similarly, "turn left" puts turning left on the IF's agenda. In addition, the lexicon entry for "turn left" specifies that, under the assumption that the IF understands and follows the instruction, they will turn 90 degrees to the left after hearing it. The planning operators are written in a way that assumes that the intended (perlocutionary) effects of an utterance actually come true. This assumption is crucial in connecting the non-linguistic effects of one SCRISP action to the non-linguistic preconditions of another, and generalizes to a scalable model of planning perlocutionary acts. We discuss this in more detail in Koller et al. (2010a).

turnleft(u, x, o_1, o_2):
  Precond: subst(S, u), ref(u, x), player–ori(o_1),
    next–ori–left(o_1, o_2), . . .
  Effect: ¬subst(S, u), ¬player–ori(o_1), player–ori(o_2),
    to–do(turnleft), . . .

push($u, u_1, u_n, x, x_1, p, o$):
  Precond: subst(S, u), ref(u, x), player–pos(p),
    player–ori(o), visible(p, o, x_1), . . .
  Effect: ¬subst(S, u), subst(NP, u_1), ref(u_1, x_1),
    $\forall y.(y \neq x_1 \wedge$ visible$(p, o, y) \rightarrow$ distractor$(u_1, y))$,
    to–do(push(x_1)), canadjoin(S, u), . . .

and(u, u_1, u_n, e_1, e_2):
  Precond: canadjoin(S, u), ref(u, e_1), . . .
  Effect: subst(S, u_1), ref(u_1, e_2), . . .

Figure 5: SCRISP planning operators for the lexicon in Fig. 4.

We then translate a SCRISP generation problem into a planning problem. In addition to what CRISP does, we translate all non-linguistic conditions into preconditions and all non-linguistic effects into effects of the planning operator, adding any free variables to the operator's parameters. An imperative effect P is translated into an effect to–do(P). The operators for the example lexicon of Fig. 4 are shown in Fig. 5. Finally, we add information about the situated environment to the initial state, and specify the planning goal by adding to–do(P) atoms for each atom P that is to be placed on the IF's agenda.

4.2 An example

Now let's look at how this generates the appropriate instructions for our example scene of Fig. 3. We encode the state of the world as depicted in the map in an initial state which contains, among others, the atoms player–pos(pos$_{3,2}$), player–ori(north), next–ori–left(north, west),

$\mathsf{visible}(\mathsf{pos}_{3,2}, \mathsf{west}, b_1)$, etc.[2] We want the IF to press b_1, so we add $\mathsf{to\text{–}do}(\mathsf{push}(b_1))$ to the goal.

We can start by applying the action **turnleft**$(\mathsf{root}, e, \mathsf{north}, \mathsf{west})$ to the initial state. Next to the ordinary grammatical effects from CRISP, this action makes $\mathsf{player\text{–}ori}(\mathsf{west})$ true. The new state does not contain any subst atoms, but we can continue the sentence by adjoining "and", i.e. by applying the action **and**$(\mathsf{root}, \mathsf{n}_1, \mathsf{n}_2, e, e_1)$. This produces a new atom $\mathsf{subst}(\mathrm{S}, e_1)$, which satisfies one precondition of **push**$(\mathsf{n}_1, \mathsf{n}_2, \mathsf{n}_3, e_1, b_1, \mathsf{pos}_{3,2}, \mathsf{west})$. Because **turnleft** changed the player orientation, the $\mathsf{visible}$ precondition of **push** is now satisfied too (unlike in the initial state, in which b_1 was not visible). Applying the action **push** now introduces the need to substitute a noun phrase for the object, which we can eliminate with an application of **the-button**(n_2, b_1) as in Subsection 3.2.

Since there are no other visible buttons from $\mathsf{pos}_{3,2}$ facing west, there are no remaining $\mathsf{distractor}$ atoms at this point, and a goal state has been reached. Together, this four-step plan decodes into the sentence "turn left and push the button". The final state contains the atoms $\mathsf{to\text{–}do}(\mathsf{push}(b_1))$ and $\mathsf{to\text{–}do}(\mathsf{turnleft})$, indicating that an IF that understands and accepts this instruction also accepts these two commitments into their to-do list.

5 Generating context-dependent adjectives

Now consider if we wanted to instruct the IF to press b_2 in Fig. 3 instead of b_1, say with the instruction "push the left button". This is still challenging, because (like most other approaches to RE generation) CRISP interprets adjectives by simply intersecting all their extensions. In the case of "left", the most reasonable way to do this would be to interpret it as "leftmost among all visible objects"; but this is f_1 in the example, and so there is no distinguishing RE for b_2.

In truth, spatial adjectives like "left" and "upper" depend on the context in two different ways. On the one hand, they are interpreted with respect to the current spatio-visual context, in that what is on the left depends on the current position and orientation of the hearer. On the other hand, they also depend on the meaning of the phrase they modify: "the left button" is not necessarily both a button and further to the left than all other objects, it is only the leftmost object among the buttons.

left(u, x):
Precond: $\forall y.\neg(\mathsf{distractor}(u, y) \wedge \mathsf{left\text{–}of}(y, x))$,
$\mathsf{canadjoin}(\mathsf{N}, u), \mathsf{ref}(u, x)$
Effect: $\forall y.(\mathsf{left\text{–}of}(x, y) \rightarrow \neg\mathsf{distractor}(u, y))$,
$\mathsf{premod\text{–}index}(u, 2), \ldots$

red(u, x):
Precond: $\mathsf{red}(x), \mathsf{canadjoin}(\mathsf{N}, u), \mathsf{ref}(u, x)$,
$\neg\mathsf{premod\text{–}index}(u, 2)$
Effect: $\forall y.(\neg\mathsf{red}(y) \rightarrow \neg\mathsf{distractor}(u, y))$,
$\mathsf{premod\text{–}index}(u, 1), \ldots$

Figure 6: SCRISP operators for context-dependent and context-independent adjectives.

We will now show how to extend SCRISP so it can generate REs that use such context-dependent adjectives.

5.1 Context-dependence of adjectives in SCRISP

As a planning-based approach to NLG, SCRISP is not limited to simply intersecting sets of potential referents that only depend on the attributes that contribute to an RE: Distractors are removed by applying operators which may have context-sensitive conditions depending on the referent and the distractors that are still left.

Our encoding of context-dependent adjectives as planning operators is shown in Fig. 6. We only show the operators here for lack of space; they can of course be computed automatically from lexicon entries. In addition to the ordinary CRISP preconditions, the **left** operator has a precondition requiring that no current distractor for the RE u is to the left of x, capturing a presupposition of the adjective. Its effect is that everything that is to the right of x is no longer a distractor for u. Notice that we allow that there may still be distractors after **left** has been applied (above or below x); we only require unique reference in the goal state. (Ignore the $\mathsf{premod\text{–}index}$ part of the effect for now; we will get to that in a moment.)

Let's say that we are computing a plan for referring to b_2 in the example map of Fig. 3, starting with **push**$(\mathsf{root}, \mathsf{n}_1, \mathsf{n}_2, e, b_2, \mathsf{pos}_{3,1}, \mathsf{north})$ and **the-button**(n_1, b_2). The state after these two actions is not a goal state, because it still contains the atom $\mathsf{distractor}(\mathsf{n}_1, b_3)$ (the plant f_1 was removed as a distractor by the action **the-button**).

[2] In a more complex situation, it may be infeasible to exhaustively model visibility in this way. This could be fixed by connecting the planner to an external spatial reasoner (Dornhege et al., 2009).

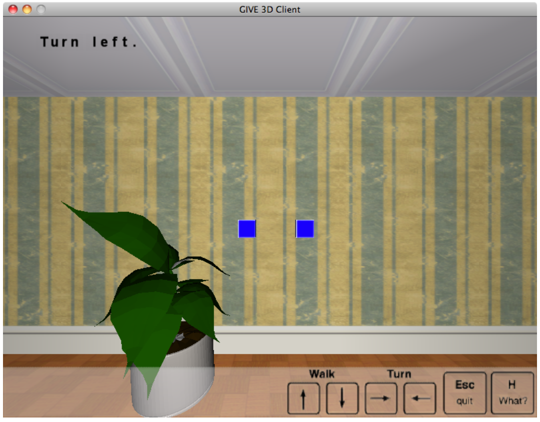
Now assume that we have modeled the spatial relations between all objects in the initial state in left–of and above atoms; in particular, we have left–of(b_2, b_3). Then the action instance **left**(n_1, b_2) is applicable in this state, as there is no other object that is still a distractor in this state and that is to the left of b_2. Applying **left** removes distractor(n_1, b_3) from the state. Thus we have reached a goal state; the complete plan decodes to the sentence "push the left button".

This system is sensitive to the order in which operators for context-dependent adjectives are applied. To generate the RE "the upper left button", for instance, we first apply the **left** action and then the **upper** action, and therefore **upper** only needs to remove distractors in the leftmost position. On the other hand, the RE "the left upper button" corresponds to first applying **upper** and then **left**. These action sequences succeed in removing all distractors for different context states, which is consistent with the difference in meaning between the two REs.

Furthermore, notice that the adjective operators themselves do not interact directly with the encoding of the context in atoms like visible and player–pos, just like the noun operators in Section 4 didn't. The REs to which the adjectives and nouns contribute are introduced by verb operators; it is these verb operators that inspect the current context and initialize the distractor set for the new RE appropriately. This makes the correctness of the generated sentence independent of the order in which noun and adjective operators occur in the plan. We only need to ensure that the verbs are ordered correctly, and the workload of modeling interactions with the non-linguistic context is limited to a single place in the encoding.

5.2 Adjective word order

One final challenge that arises in our system is to generate the adjectives in the correct order, which on top of semantically valid must be linguistically acceptable. In particular, it is known that some types of adjectives are limited with respect to the word order in which they can occur in a noun phrase. For instance, "large foreign financial firms" sounds perfectly acceptable, but "? foreign large financial firms" sounds odd (Shaw and Hatzivassiloglou, 1999). In our setting, some adjective orders are forbidden because only one order produces a correct and distinguishing description of the target referent (cf. "upper left" vs. "left upper" example above). However, there are also other constraints at work: "? the red left button" is rather odd even when it is a semantically correct description, whereas "the left red button" is fine.



Turn left.

Figure 7: The IF's view of the scene in Fig. 3, as rendered by the GIVE client.

To ensure that SCRISP chooses to generate these adjectives correctly, we follow a class-based approach to the premodifier ordering problem (Mitchell, 2009). In our lexicon we assign adjectives denoting spatial relations ("left") to one class and adjectives denoting color ("red") to another; then we require that spatial adjectives must always precede color adjectives. We enforce this by keeping track of the current *premodifier index* of the RE in atoms of the form premod–index. Any newly generated RE node starts off with a premodifier index of zero; adjoining an adjective of a certain class then raises this number to the index for that class. As the operators in Fig. 6 illustrate, color adjectives such as "red" have index one and can only be used while the index is not higher; once an adjective from a higher class (such as "left", of a class with index two) is used, the premod–index precondition of the "red" operator will fail. For this reason, we can generate a plan for "the left red button", but not for "? the red left button", as desired.

6 Evaluation

To establish the quality of the generated instructions, we implemented SCRISP as part of a generation system in the GIVE-1 framework, and evaluated it against two baselines. GIVE-1 was the First Challenge on Generating Instructions in Virtual Environments, which was completed in 2009

SCRISP	*1. Turn right and move one step.* *2. Push **the right red button**.*
Baseline A	*1. Press **the right red button on the wall to your right**.*
Baseline B	*1. Turn right.* *2. Walk forward 3 steps.* *3. Turn right.* *4. Walk forward 1 step.* *5. Turn left.* *6. Good! Now press **the left button**.*

Table 1: Example system instructions generated in the same scene. REs for the target are typeset in boldface.

	success		**RE**	
	rate	**time**	**success**	**distance**
SCRISP	69%	306	71%	2.49
Baseline A	16%**	230	49%**	1.97*
Baseline B	84%	288	81%*	2.00*

Table 2: Evaluation results. Differences to SCRISP are significant at *$p < .05$, **$p < .005$ (Pearson's chi-square test for system success rates; unpaired two-sample t-test for the rest).

(Koller et al., 2010b). In this challenge, systems must generate real-time instructions that help users perform a task in a treasure-hunt virtual environment such as the one shown in Fig. 7.

We conducted our evaluation in World 2 from GIVE-1, which was deliberately designed to be challenging for RE generation. The world consists of one room filled with several objects and buttons, most of which cannot be distinguished by simple descriptions. Moreover, some of those may activate an alarm and cause the player to lose the game. The player's moves and turns are discrete and the NLG system has complete and accurate real-time information about the state of the world. Instructions that each of the three systems under comparison generated in an example scene of the evaluation world are presented in Table 1.

The evaluation took place online via the Amazon Mechanical Turk, where we collected 25 games for each system. We focus on four measures of evaluation: success rates for solving the task and resolving the generated REs, average task completion time (in seconds) for successful games, and average distance (in steps) between the IF and the referent at the time when the RE was generated. As in the challenge, the task is considered as solved if the player has correctly been led through manipulating all target objects required to discover and collect the treasure; in World 2, the minimum number of such targets is eight. An RE is successfully resolved if it results in the manipulation of the referent, whereas manipulation of an alarm-triggering distractor ends the game unsuccessfully.

6.1 The SCRISP system

Our system receives as input a plan for what the IF should do to solve the task, and successively takes object-manipulating actions as the communicative goals for SCRISP. Then, for each of the communicative goals, it generates instructions using SCRISP, segments them into navigation and action parts, and presents these to the user as separate instructions sequentially (see Table 1).

For each instruction, SCRISP thus draws from a knowledge base of about 1500 facts and a grammar of about 30 lexicon entries. We use the FF planner (Hoffmann and Nebel, 2001; Koller and Hoffmann, 2010) to solve the planning problems. The maximum planning time for any instruction is 1.03 seconds on a 3.06 GHz Intel Core 2 Duo CPU. So although our planning-based system tackles a very difficult search problem, FF is very good at solving it—fast enough to generate instructions in real time.

6.2 Comparison with Baseline A

Baseline A is a very basic system designed to simulate the performance of a classical RE generation module which does not attempt to manipulate the visual context. We hand-coded a correct distinguishing RE for each target button in the world; the only way in which Baseline A reacts to changes of the context is to describe on which wall the button is with respect to the user's current orientation (e.g. "Press the right red button *on the wall to your right*").

As Table 2 shows, our system guided 69% of users to complete the task successfully, compared to only 16% for Baseline A (difference is statistically significant at $p < .005$; Pearson's chi-square test). This is primarily because only 49% of the REs generated by Baseline A were successful. This comparison illustrates the importance of REs that minimize the cognitive load on the IF to avoid misunderstandings.

6.3 Comparison with Baseline B

Baseline B is a corrected and improved version of the "Austin" system (Chen and Karpov, 2009), one of the best-performing systems of the GIVE-1 Challenge. Baseline B, like the original "Austin" system, issues navigation instructions by precomputing the shortest path from the IF's current location to the target, and generates REs using the description logic based algorithm of Areces et al. (2008). Unlike the original system, which inflexibly navigates the user all the way to the target, Baseline B starts off with navigation, and opportunistically instructs the IF to push a button once it has become visible and can be described by a distinguishing RE. We fixed bugs in the original implementation of the RE generation module, so that Baseline B generates only unambiguous REs. The module nonetheless naively treats all adjectives as intersective and is not sensitive to the context of their comparison set. Specifically, a button cannot be referred to as "the *right* red button" if it is not the rightmost of all visible objects—which explains the long chain of navigational instructions the system produced in Table 1.

We did not find any significant differences in the success rates or task completion times between this system and SCRISP, but the former achieved a higher RE success rate (see Table 2). However, a closer analysis shows that SCRISP was able to generate REs from significantly further away. This means that SCRISP's RE generator solves a harder problem, as it typically has to deal with more visible distractors. Furthermore, because of the increased distance, the system's execution monitoring strategies (e.g. for detection and repair of misunderstandings) become increasingly important, and this was not a focus of this work. In summary, then, we take the results to mean that SCRISP performs quite capably in comparison to a top-ranked GIVE-1 system.

7 Conclusion

In this paper, we have shown how situated instructions can be generated using AI planning. We exploited the planner's ability to model the perlocutionary effects of communicative actions for efficient generation. We showed how this made it possible to generate instructions that manipulate the non-linguistic context in convenient ways, and to generate correct REs with context-dependent adjectives.

We believe that this illustrates the power of a planning-based approach to NLG to flexibly model very different phenomena. An interesting topic for future work, for instance, is to expand our notion of context by taking visual and discourse salience into account when generating REs. In addition, we plan to experiment with assigning costs to planning operators in a metric planning problem (Hoffmann, 2002) in order to model the cognitive cost of an RE (Krahmer et al., 2003) and compute minimal-cost instruction sequences.

On a more theoretical level, the SCRISP actions model the physical effects of a correctly understood and grounded instruction directly as effects of the planning operator. This is computationally much less complex than classical speech act planning (Perrault and Allen, 1980), in which the intended physical effect comes at the end of a long chain of inferences. But our approach is also very optimistic in estimating the perlocutionary effects of an instruction, and must be complemented by an appropriate model of execution monitoring. What this means for a novel scalable approach to the pragmatics of speech acts (Koller et al., 2010a) is, we believe, an interesting avenue for future research.

Acknowledgments. We are grateful to Jörg Hoffmann for improving the efficiency of FF in the SCRISP domain at a crucial time, and to Margaret Mitchell, Matthew Stone and Kees van Deemter for helping us expand our view of the context-dependent adjective generation problem. We also thank Ines Rehbein and Josef Ruppenhofer for testing early implementations of our system, and Andrew Gargett as well as the reviewers for their helpful comments.

References

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

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*, pages 42–49, Salt Fork, Ohio, USA.

Luciana Benotti. 2009. Clarification potential of instructions. In *Proceedings of the SIGDIAL 2009 Conference*, pages 196–205, London, UK.

Michael Brenner and Ivana Kruijff-Korbayová. 2008. A continual multiagent planning approach to situated dialogue. In *Proceedings of the 12th Workshop on the Semantics and Pragmatics of Dialogue*, London, UK.

David Chen and Igor Karpov. 2009. The GIVE-1 Austin system. In *The First GIVE Challenge: System descriptions*. http://www.give-challenge.org/research/files/GIVE-09-Austin.pdf.

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

Christian Dornhege, Patrick Eyerich, Thomas Keller, Sebastian Trüg, Michael Brenner, and Bernhard Nebel. 2009. Semantic attachments for domain-independent planning systems. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling*, pages 114–121.

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

Jörg Hoffmann. 2002. Extending FF to numerical state variables. In *Proceedings of the 15th European Conference on Artificial Intelligence*, Lyon, France.

Aravind K. Joshi and Yves Schabes. 1997. Tree-Adjoining Grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 69–123. Springer-Verlag, Berlin, Germany.

Hans Kamp and Barbara Partee. 1995. Prototype theory and compositionality. *Cognition*, 57(2):129 – 191.

Alexander Koller and Jörg Hoffmann. 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*, Toronto, Canada.

Alexander Koller and Matthew Stone. 2007. Sentence generation as planning. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, Prague, Czech Republic.

Alexander Koller, Andrew Gargett, and Konstantina Garoufi. 2010a. A scalable model of planning perlocutionary acts. In *Proceedings of the 14th Workshop on the Semantics and Pragmatics of Dialogue*, Poznan, Poland.

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

Emiel Krahmer and Mariet Theune. 2002. Efficient context-sensitive generation of referring expressions. In Kees van Deemter and Rodger Kibble, editors, *Information Sharing: Reference and Presupposition in Language Generation and Interpretation*, pages 223–264. CSLI Publications.

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

Margaret Mitchell. 2009. Class-based ordering of prenominal modifiers. In *Proceedings of the 12th European Workshop on Natural Language Generation*, pages 50–57, Athens, Greece.

Dana Nau, Malik Ghallab, and Paolo Traverso. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.

C. Raymond Perrault and James F. Allen. 1980. A plan-based analysis of indirect speech acts. *American Journal of Computational Linguistics*, 6(3–4):167–182.

Paul Portner. 2007. Imperatives and modals. *Natural Language Semantics*, 15(4):351–383.

James Shaw and Vasileios Hatzivassiloglou. 1999. Ordering among premodifiers. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 135–143, College Park, Maryland, USA.

Mark Steedman and Ronald P. A. Petrick. 2007. Planning dialog actions. In *Proceedings of the 8th SIGdial Workshop on Discourse and Dialogue*, pages 265–272, Antwerp, Belgium.

Laura Stoia, Donna K. Byron, Darla Magdalene Shockley, and Eric Fosler-Lussier. 2006. Sentence planning for realtime navigational instructions. In *NAACL '06: Proceedings of the Human Language Technology Conference of the NAACL*, pages 157–160, Morristown, NJ, USA.

Laura Stoia, Darla M. Shockley, Donna K. Byron, and Eric Fosler-Lussier. 2008. SCARE: A situated corpus with annotated referring expressions. In *Proceedings of the 6th International Conference on Language Resources and Evaluation*, Marrakech, Morocco.

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.

Kees van Deemter. 2006. Generating referring expressions that involve gradable properties. *Computational Linguistics*, 32(2).