Talking about what is not there: Generating indefinite referring expressions in Minecraft

Arne Köhn and **Alexander Koller**
Department of Language Science and Technology
Saarland University
{koehn|koller}@coli.uni-saarland.de

Abstract

When generating technical instructions, it is often necessary to describe an object that does not exist yet. For example, an NLG system which explains how to build a house needs to generate sentences like "build *a wall of height five to your left*" and "now build *a wall on the other side*." Generating (indefinite) referring expressions to objects that do not exist yet is fundamentally different from generating the usual definite referring expressions, because the new object must be distinguished from an infinite set of possible alternatives. We formalize this problem and present an algorithm for generating such expressions, in the context of generating building instructions within the Minecraft video game.

1 Introduction

Everyone who has ever had to assemble a piece of furniture knows the importance of clearly worded technical instructions. Ideally, such instructions should be personalized to the current user and their level of expertise, and should be rephrased if the user has trouble with some instruction.

In this paper, we explore the challenge of automatically generating personalized building instructions in the context of the video game "Minecraft". Minecraft is a game in which players can explore an open world, mine materials, and build complex structures such as houses and technical devices. By situating the NLG task in Minecraft, we retain the challenge of generating complex technical instructions, within the context of a fully observable virtual environment. Minecraft is an extremely popular game, with 50 million active users, which facilitates access to experimental subjects for evaluation. There are currently about 130 million Minecraft-related videos on YouTube, many of which have humans explain how to construct complex objects; thus there is a clear interest in building instructions for Minecraft.

One challenge in generating building instructions for Minecraft is to refer to objects that do not exist yet. For instance, in order to instruct the player to build the left railing of the bridge in Fig. 2c, the NLG system might say "build *a railing on the left side of the bridge*". When this instruction is uttered, there is no left railing yet; thus one uses the indefinite referring expression (RE) in italics to refer to a non-existing object. Generating such indefinite NPs is a fundamentally different task than the more established task of generating definite NPs, in which the target referent only needs to be distinguished from a finite set of distractors. In generating indefinite REs, the target referent needs to be distinguished from an infinite set of alternatives, such as railings of different lengths and in different locations. This makes it infeasible to use existing generation algorithms for definite REs.

In this paper, we show how to generate indefinite REs for objects in Minecraft which do not exist yet. We proceed in two steps. First, we will show how to generate definite and indefinite REs to individual blocks in Minecraft, in a way which makes it possible to generate REs for locations in the environment, in addition to REs for objects. We will then build a method for generating definite and indefinite REs to complex objects (such as railings) upon this. The basic idea is to precompute possible sets of features which will uniquely identify the complex object by specifying all of its relevant attributes. We implement our RE generation method in terms of Semantically Interpreted Grammars (SIGs) (Koller and Engonopoulos, 2017) and use their chart-based sentence generator to perform the actual RE generation.

Plan of the paper. We will introduce the task and discuss some related work in Section 2 and

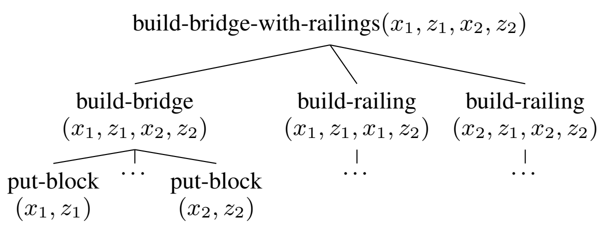
build-bridge-with-railings(x_1, z_1, x_2, z_2)
build-bridge (x_1, z_1, x_2, z_2)
build-railing (x_1, z_1, x_1, z_2)
build-railing (x_2, z_1, x_2, z_2)
put-block (x_1, z_1)
...
put-block (x_2, z_2)
...
...

Figure 1: A construction plan.

sketch chart generation with SIGs in Section 3. We will then show how to refer to individual blocks in Section 4 and to complex objects in Section 5. Section 6 concludes and discusses future work.

2 Background

There is an extensive literature on the generation of definite REs to objects which exist in the environment. These algorithms typically aim to distinguish the target referent from the other existing objects, and can thus assume that the denotations of all properties are subsets of the (finite) set of existing objects. They then intersect denotations of properties in some way, keeping track of the (finite) set of possible referents in the intersection, until the set of referents becomes singleton. The classical reference for this approach is the Incremental Algorithm of Dale and Reiter (1995); other methods for generating definite REs follow the same perspective (Krahmer et al., 2003). See (Krahmer and van Deemter, 2012; van Deemter, 2016) for an overview. To our knowledge, there is no accepted method for systematically generating indefinite REs.

There is also an extensive literature on the meaning of indefinites in theoretical semantics. Classical theories (Heim, 1982; Kamp and Reyle, 1993) focus on the ability of indefinites to update the discourse context with new referents; their ability to refer to objects in the world is only a byproduct of this. von Heusinger (2000) describes the referential potential of indefinites in terms of choice functions, i.e. the referent of “a bridge” is an arbitrary bridge. In either case, there is no requirement that an indefinite noun phrase should refer uniquely, as we require in Minecraft building instructions.

2.1 Instruction generation in Minecraft

Generating indefinite referring expressions is a sub problem of instruction generation, for which we first sketch the overall setting. In the instruction giving task, an *architect* observes the actions of a (human) builder and sends real-time instructions to the builder. The architect has to solve several tasks: First, it needs to generate a *construction plan* which describes the actions the player needs to take in the Minecraft environment in order to construct the goal object (see Wichlacz et al. (2019)). For instance, a possible construction plan for the bridge in Fig. 2 is shown in Fig. 1. Observe that the construction plan is *hierarchical*: Higher-level actions such as “build bridge with railings” are recursively decomposed into lower-level actions; this hierarchical decomposition is present an most instruction giving tasks. At the lowest level of abstraction, each action consists of the placement of individual blocks. As the actions become more fine-grained, so do the types of objects they manipulate (the complete bridge down to the individual blocks). This construction plan can be computed by a hierarchical planner.

The architect then decides at which level of abstraction each construction step should be described; for instance, it might decide to describe the construction in terms of the individual blocks, or it might decide to instruct the player to build a bridge and then the two railings. This decision depends on whether the architect can assume that the builder understands the concept “railing” in the same way the architect intends it: as a single row of blocks along the bridge. The result is an *instruction plan*. Finally, a *sentence generator* translates each step of the instruction plan to a natural-language sentence. For instance, it might map the instruction step ’build-railing(x_2, z_1, x_2, z_2)’ to the sentence “build a railing on the other side of the bridge”. This paper is concerned with this sentence generation task – more precisely, with the referring expression part.

2.2 Reference in the Minecraft environment

We distinguish two types of objects in a Minecraft environments: individual blocks and *complex objects (COs)* such as bridges, railings, or houses. Each type of object has a set of *features*: bridges for example can be described by the features $(\mathrm{type}, \mathrm{corner}_1, \mathrm{corner}_2, \mathrm{corner}_3, \mathrm{corner}_4, \mathrm{width}, \mathrm{length})$, where the corners define the coordinates of each corner in space and type defines what kind of object this is (in this case, a bridge). A concrete object has a specific value for each feature and we

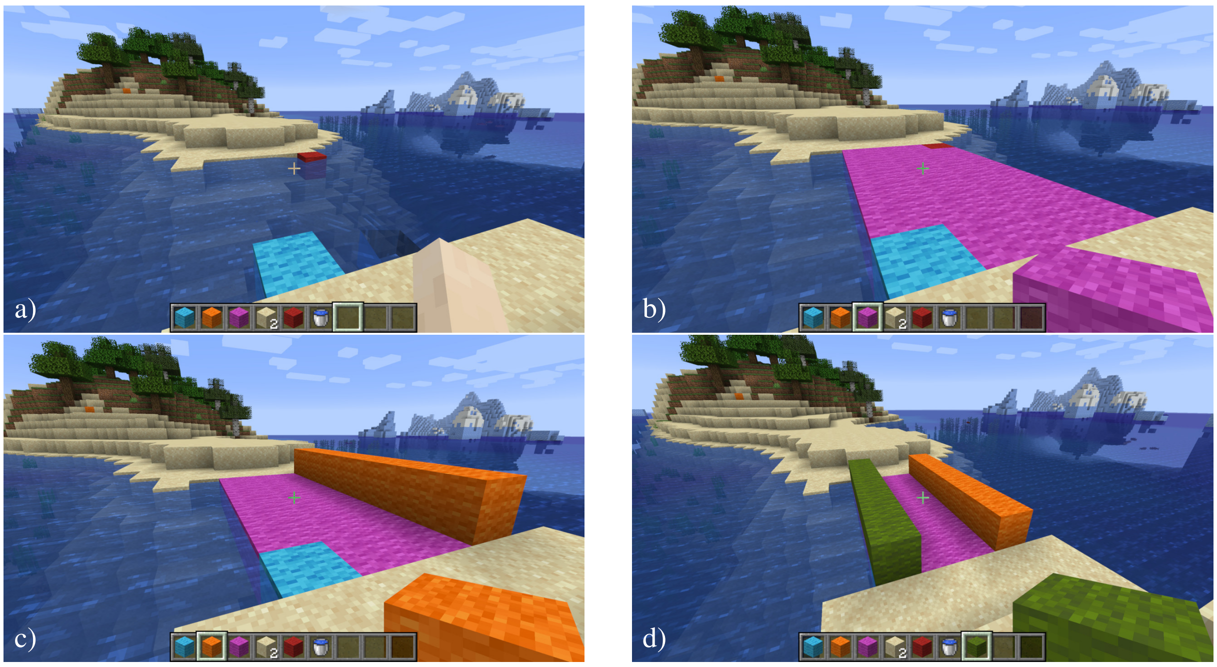

Figure 2: Instructing a user to build a bridge. Left: current state, right: target state. For the first task, the user could be instructed by saying "build a bridge from the blue block to the red block", for the second by "build a railing on the other side of the bridge".

will call a value associated with a feature a *property*. Each object – whether it already exists or not – has a property $p(f)$ for each feature f of its type.[1] For instance, a block b at the location $(1, 1, 1)$ has the two properties $[\text{position} = (1, 1, 1)]$ and $[\text{type} = \mathit{block}]$.

These properties are not necessarily independent: the width and length of a bridge are determined uniquely by the positions of the corners. Features are modeled in such a way that there is only one possible object of each type with a certain property tuple, and we identify the objects with their property tuples.

We follow the mainstream literature on RE generation by assuming that the purpose of a referring expression is to uniquely distinguish an object from all alternatives. Technically, we assume a set $\mathcal{P}$ of properties; each property $p \in \mathcal{P}$ denotes a subset $\mathcal{R}(p)$ of objects in the world. For example, the property $[\text{width} = 3]$ denotes only those objects whose width is 3; the property $[\text{type} = \mathit{bridge}]$ only those objects which are bridges; and so on. When generating a referring expression, we can keep track of which features are specified by describing the corresponding property of the target object. While it might seem unintuitive to keep track of the features and not of the corresponding properties, but this approach will come in handy when generating indefinite REs. We denote the properties of an object o corresponding to a specific set of features F as $P_o(F)$. For the block b introduced above, $P_b(\{\text{location}, \text{type}\})$ would be $\{[\text{position} = \{1, 1, 1\}], [\text{type} = \mathit{block}]\}$. When trying to refer to an object o, we need to find a set of features which together distinguish this object from all other objects. We say that a finite set F of features is *distinguishing* for a specific target object t if t is the only object with all the properties $D = P_t(F)$, i.e. $\bigcap_{p \in D} \mathcal{R}(p) = \{t\}$.

[1]Note that type itself is also a feature to distinguish objects of different types.

2.3 Referring to objects that do not exist

The starting point of this paper is the observation that indefinite reference to objects which do not exist yet differs fundamentally from definite reference to existing objects. In the extensive literature on generation of definite REs, it is customary to limit the denotation of each property to objects which already exist in the environment. This makes these denotations finite, and thus a definite RE can be computed by keeping explicit track of (finite) sets of possible referents and intersecting properties until the set becomes singleton.

By contrast, a unique indefinite RE to an object which does not exist yet must distinguish it from

type	distinguishing features
bridge	$\{\mathrm{type}, \mathrm{corner}_1, \mathrm{corner}_3\}$
	$\{\mathrm{type}, \mathrm{corner}_1, \mathrm{width}, \mathrm{length}\}$
railing	$\{\mathrm{type}, \mathrm{corner}_1, \mathrm{corner}_2\}$
	$\{\mathrm{type}, \mathrm{corner}_1, \mathrm{length}, \mathrm{orientation}\}$

Figure 3: Examples for distinguishing feature sets.

the entire infinite set of other objects which do not exist yet. For instance, in the example of Fig. 2a, it would be insufficient to say "build a bridge": There are infinitely many places at which a bridge could be built, and it could potentially point in an arbitrary direction[2]. Thus, algorithms which are based on intersecting denotations of properties will not generalize to indefinites.

We therefore choose a different approach. We predefine, for each type of object, a number of distinguishing sets of features (see Figure 3). By definition, if an RE realizes such a feature set by describing the corresponding properties of the target object, the target referent will be described uniquely, even among a potentially infinite set of alternative referents. For instance, "a bridge from the blue block to the red block" is a unique RE because it expresses the feature set $\{\mathrm{type}, \mathrm{corner}_1, \mathrm{corner}_3\}$, which is distinguishing. Similarly, "a bridge of width three and length five, starting at the blue block" is also a unique RE.

There is one way in which generating indefinite REs is easier than generating definite ones: Because there is always an infinite set of alternative referents, no matter what the actual Minecraft world looks like, the distinguishing feature sets are always the same. Thus, unlike in definite RE generation, we do not have to recalculate which feature sets are distinguishing for each world, but can simply reuse feature sets as in Fig. 3. This makes it feasible to shift some of the complexity of generating an indefinite RE from intersecting properties at sentence generation time to a preprocessing module.

[2]Even when considering saliency for an object's properties (bridges are usually built over water, walls on the ground), there is still no straight-forward way to distinguish the (possibly finite) set of plausible property tuples from the implausible ones.

3 Semantically Interpreted Grammars

In this paper, we build upon the chart generation algorithm of (Koller and Engonopoulos, 2017) for Semantically Interpreted Grammars (SIGs, (Engonopoulos and Koller, 2014)). SIGs are synchronous grammars which compositionally relate natural-language expressions with their possible referents; they are a special case of IRTGs (Koller and Kuhlmann, 2011). The K&E algorithm performs surface realization together with the generation of definite REs; we extend it to indefinites here.

At the core of a SIG is a regular tree grammar (RTG, (Comon et al., 2007)) which generates a language of *derivation trees*. These derivation trees represent the underlying structure of the referring expression; Fig. 4 shows such a derivation tree. Derivation trees are built by repeatedly replacing nonterminal symbols using the production rules of the RTG (see the first lines in the toy grammar in Fig. 5), starting from a designated start symbol. For instance, the rule $\mathrm{RefBlock} \to top(\mathrm{RefBlock})$ replaces an occurrence of the nonterminal RefBlock by a node which has the label *top* and a child labeled with the nonterminal RefBlock. This new nonterminal occurrence will again be replaced by the right-hand side of a production rule, and so on until no nonterminals are left. We write $L_S(G)$ for the set of derivation trees which can be derived from the start symbol S. If we assume that RefBlock is the start symbol, the tree in Fig. 4 is in the language $L_{\mathrm{RefBlock}}(G)$ of the toy grammar.

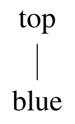

Figure 4: Derivation tree t for "The block on top of the blue block".

Interpretations SIGs describe k-ary relations by taking the derivation trees t described by the RTG and *interpreting* each t in k different ways. Technically, if the derivation trees have node labels in the ranked signature Σ, we say that an interpretation $\mathcal{I}$ consists of a domain $\Delta_\mathcal{I}$, a set of functions $\Sigma_\mathcal{I}$ over this domain ($\Delta_\mathcal{I}^n \to \Delta_\mathcal{I}$), and a homomorphism from Σ to these functions. By evaluating these functions recursively along t, we obtain a value $[\![t]\!]_\mathcal{I} \in \Delta_\mathcal{I}$ of t in the interpretation $\mathcal{I}$.

String interpretation Basic SIGs focus on two specific interpretations. The *string interpretation* $\mathcal{S}$ generates natural-language expressions from t. The

RefBlock $\rightarrow$ *top*(RefBlock)
$\mathcal{I}_S(top)(A) =$ The block on top of $\bullet$ A
$\mathcal{I}_R(top)(A) = [\mathsf{topof} \cap_2 A]_1$
$\mathcal{I}_F(top)(A) = \{\text{location}\} \cup A$

RefBlock $\rightarrow$ *blue*
$\mathcal{I}_S(blue) =$ the blue block
$\mathcal{I}_R(blue) = uniq(\mathsf{blue})$
$\mathcal{I}_F(blue) = \{\text{location}\}$

Figure 5: A grammar for referring to a block. The example is greatly simplified, see Figures 7 and 9 and Engonopoulos and Koller (2014) for more realistic grammars.

domain Δ_S consists of strings such as “the” and “the block”. The functions Σ_s only use string concatenation $\bullet$ and string literals. In our toy example, $[\![t]\!]_S$ is computed by first obtaining a value for *blue*: $\mathcal{I}_S(blue) =$ “the blue block” and based on that obtaining the value of the whole tree: $\mathcal{I}_S(top)$(“the blue block”) $=$ “the block on top of” $\bullet$ “the blue block” = “the block on top of the blue block”.

Referential interpretation Second, the *referential interpretation* $\mathcal{R}$ maps (partial) derivation trees t to sets of objects to which t might refer.

The definition of the referential interpretation relies on a *model* m of the current world state. We take a model to be a relational structure which maps names of relations to n-ary relations over some universe W. For the toy example, let's assume a model with two relations: the unary relation $\mathsf{blue} \subseteq W^1$ and the binary relation $\mathsf{topof} \subseteq W^2$; we will work with the model $m = \{\mathsf{blue}\colon \{(b1)\}; \mathsf{topof}\colon \{(b2, b1)\}\}$.

The domain of the referential interpretation is the set of tuples of objects in the universe, i.e. W^*. The functions Σ_r perform set operations utilizing the model of the world. Constants such as blue denote the corresponding set of tuples of the model. $uniq(A)$ returns A if the set A contains exactly one tuple, and the empty set otherwise. Its use in the second rule ensures that one can only say “the blue block” to refer to a block if there is exactly one blue block in the world: $[\![blue]\!]_\mathcal{R} = uniq(\mathsf{blue}) = uniq(\{(b1)\}) = \{(b1)\}$.

$A \cap_2 B$ filters A: $A \cap_i B := \{a \in A : (a[i]) \in B\}$, i. e. only those tuples of A are retained for which the i-th element is an element of B. The operator $[A]_i$ projects all tuples of A to their i-th element: $[A]_i := \{(a[i]) \mid a \in A\}$. The interpretation of t is thus

$$\begin{aligned}[\![t]\!]_\mathcal{R} &= [\mathsf{topof} \cap_2 \{(b1)\}]_1 \\ &= [\{(b2, b1)\} \cap_2 \{(b1)\}]_1 \\ &= [\{(b2, b1)\}]_1 \\ &= \{(b2)\}\end{aligned}$$

Note that because t denotes a singleton set of objects, it represents a unique RE for $b2$.

The language of a SIG A SIG G uses a string and a referential interpretation to define a relation between natural-language expressions and the objects they may denote. Given a start symbol S, we define

$$L(G, S) = \{([\![t]\!]_\mathcal{S}, [\![t]\!]_\mathcal{R}) \mid t \in L_S(G)\}.$$

In the example outlined above, $L(G, \text{RefBlock})$ contains, among others, the pair (“the block on top of the blue block”, $\{b2\}$).

In an NLG scenario, we assume that we are given a target referent b for which we should generate an RE. We can restrict the language to trees that evaluate to a specific value in certain interpretations and in this way define the language of derivation trees which refer to b:

$$L(G, S, \mathcal{R}{:}\{b\}) = \{t \in L_S(G) \mid \{[\![t]\!]_\mathcal{R} = \{b\}\}\}.$$

In the example, $L(G, \text{RefBlock}, \mathcal{R}{:}\{\text{b2}\})$ contains e.g. the derivation tree t in Fig. 4. We can read off the actual referring expression $[\![t]\!]_\mathcal{S}$ via the string interpretation. The chart generation algorithm of Koller and Engonopoulos (2017) efficiently computes a compact representation of the language $L(G, S, \mathcal{R}{:}\{b\})$.

Feature interpretation In this paper, we add a third interpretation to SIGs, which maps derivation trees t to the set of features for which t determines a property value. In the toy example, there is only one feature: location. The domain of $\mathcal{F}$ are sets of features, and Σ_F are functions performing set union. For our toy example, this yields

$$\begin{aligned}[\![t]\!]_\mathcal{F} &= \{\text{location}\} \cup [\![\text{blue}]\!]_\mathcal{F} \\ &= \{\text{location}\} \cup \{\text{location}\} \\ &= \{\text{location}\}\end{aligned}$$

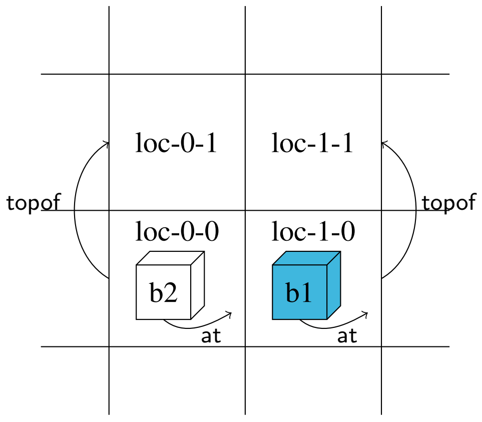

Figure 6: A small world model. The blocks are in an at relation to the locations, the locations are linked via spatial relations such as topof

Thus, we know that t expresses the location of its referent (it is on top of the blue button). We can extend the above definition to the set of derivation trees which refer to a given b and express a given set F of features:

$$L(G, S, \mathcal{R}{:}\{b\}, \mathcal{F}{:}F) = \{t \in L_S(G) \mid [\![t]\!]_{\mathcal{R}} = \{b\} \wedge [\![t]\!]_{\mathcal{F}} = F\}.$$

The chart generation algorithm generalizes easily to computing $L(G, S, \mathcal{R}{:}\{b\}, \mathcal{F}{:}F)$ for a given G, b, and F.

4 Referring expressions for locations

When we want to instruct a user to place a block at the location loc-1-1 in Fig. 6, we can say

(1) put a block on top of the blue block

which is a straightforward description of the intended action. This instruction uses the ditransitive verb “put” and the second argument to it subcategorizes a location, which is also the referring part of the instruction. In this section we will discuss how to generate these referring expressions to locations. However, the expressions still need to work with blocks to anchor the expression. We therefore model the world using a duality between blocks and locations; both blocks and locations are entities of the world. Consider these referring expressions in the world depicted in Figure 6:

(2) the blue block

(3) left of the blue block

DetRefBlock $\rightarrow drb$(RefBlock)
$\mathcal{I}_S(drb)(RefBlock) = \text{the} \bullet RefBlock$
$\mathcal{I}_R(drb)(RefBlock) = uniq(RefBlock)$
$\mathcal{I}_F(drb)(RefBlock) = \{\text{location}\} \cup RefBlock$

RefBlock $\rightarrow blue$(RefBlock)
$\mathcal{I}_S(blue)(RefBlock) = \text{blue} \bullet RefBlock$
$\mathcal{I}_R(blue)(RefBlock) = \text{blue} \cap RefBlock$
$\mathcal{I}_F(blue)(RefBlock) = RefBlock \cup \text{color}$

RefBlock $\rightarrow block$
$\mathcal{I}_S(block) = \text{block}$
$\mathcal{I}_R(block) = \text{block}$
$\mathcal{I}_F(block) = \{\text{type}\}$

RefLoc $\rightarrow top$(DetRefBlock)
$\mathcal{I}_S(top)(A) = \text{on top of} \bullet A$
$\mathcal{I}_R(top)(A) = [\text{topof} \cap_2 [\text{at} \cap_1 A]_2]_1$
$\mathcal{I}_F(top)(A) = \{\text{location}, \text{type}\}$

RefBlock $\rightarrow blockloc$(RefLoc)
$\mathcal{I}_S(blockloc)(A) = \text{block} \bullet A$
$\mathcal{I}_R(blockloc)(A) = [\text{at} \cap_2 A]_1$
$\mathcal{I}_F(blockloc)(A) = \{\text{type}\}$

Figure 7: A subset of the block laying grammar.

(4) the block left of the blue block

Sentence 2 refers to the blue block $b1$, Sentence 3 to the location loc-0-0 to the left of $b1$, and Sentence 4 to the block at the location loc-0-0.

Locations are connected with spatial relations such as topof and blocks are anchored in the world by defining at which location they are using the at relation. While Figure 6 shows only a small 2D slice of a Minecraft world, the principles extend to a 3D Minecraft world and all these relations can be captured from a Minecraft world state.

4.1 Formalizing the block laying instruction

Section 3 introduced a simple SIG to refer to blocks in the world. To generate sentences such as the one in examples 1 to 4, the grammar has to encode the duality between blocks and locations. Figure 7 shows a subset of the grammar to generate these REs. The grammar reflects the duality of blocks and locations through the non-terminals DetRefBlock (which derive referring expressions to blocks) and RefLoc (which derive referring expressions to locations). This duality between locations and blocks is encoded in the at relation, as can be seen in the last two rules of the grammar in Fig. 7: $[\mathsf{at} \cap_1 b]_2$ obtains the location of the block b and $[\mathsf{at} \cap_2 l]_1$ obtains the block at the location l.

We will illustrate how this grammar works by generating a RE to refer to loc-1-1 in the world shown in Figure 6. A model is sketched with arrows in Figure 6; the relevant subset of the model is $\{\mathsf{at}\colon \{(\text{b1}, \text{loc-1-0})\}; \mathsf{block}\colon \{(\text{b1})\}; \mathsf{topof}\colon \{(\text{loc-1-1}, \text{loc-1-0})\}\}$. With this model, a RE to loc-1-1 can be obtained with the derivation tree $t = top(drb(blue(block)))$ because it is in $L(G, \mathrm{RefLoc}, \mathcal{R}{:}\{\text{loc-1-1}\})$:

$$[\![block]\!]_{\mathcal{R}} = \{(b1), (b2)\}$$

$$\begin{aligned}[\![blue(\{(b1), (b2)\})]\!]_{\mathcal{R}} &= \mathsf{blue} \cup \{(b1), (b2)\} \\ &= \{(b1)\}\end{aligned}$$

$$\begin{aligned}[\![drb(\{(b1)\})]\!]_{\mathcal{R}} &= uniq(\{(b1)\}) \\ &= \{(b1)\}\end{aligned}$$

$$\begin{aligned}[\![top(\{(b1)\})]\!]_{\mathcal{R}} &= [\mathsf{topof} \cap_2 [\mathsf{at} \cap_1 \{(b1)\}]_2]_1 \\ &= [\mathsf{topof} \cap_2 [\{(\text{b1,loc-1-0})\}]_2]_1 \\ &= [\mathsf{topof} \cap_2 \{(\text{loc-1-0})\}]_1 \\ &= [\{(\text{loc-1-1, loc-1-0})\}]_1 \\ &= \{(\text{loc-1-1})\}\end{aligned}$$

The referring expression to loc-1-1 is obtained by computing $[\![t]\!]_{\mathcal{S}}$, which is "on top of the blue block". On the feature interpretation, the features being described are collected. Note that when using a relation such as topof to switch the referent, the features from the sub-expression are discarded.

5 Indefinite REs to complex objects

Now we extend the RE generation to complex objects. The first step is to convert the coordinate-based representation which can be easily obtained from the 3D world to relations between the objects of the world similar to the one in the previous section. This step is more complex than for simple blocks as COs have no one-to-one correspondence to locations. To generate the instruction for building the second railing in Figure 2d, we want to generate the following model, with it denoting the salient object, i. e. the one that can be referred to as "it":

$$\begin{aligned}\{\mathsf{bridge}\colon& \{(\text{b})\}; \\ \mathsf{it}\colon& \{(\text{r1})\}; \\ \mathsf{railing}\colon& \{(\text{r1}), (\text{r2})\}; \\ \mathsf{block}\colon& \{(\text{bb}), (\text{rb})\}; \\ \mathsf{from}\colon& \{(\text{b, bb})\}\}; \\ \mathsf{to}\colon& \{(\text{b, rb})\}\}; \\ \mathsf{sameshape}\colon& \{(\text{r1, r2})\}\}; \\ \mathsf{otherside}\colon& \{(\text{r1, b, r2})\}\};\end{aligned}$$

The object are the blocks rb (the red block) and bb (the blue one), the railings $r1$ and $r2$ ($r2$ is the railing that should be built) and the bridge b If the target object is a CO that does not exist yet, it is added to the objects from which the model is built so that relations between the target and the other objects are generated; this can be seen in the relations above containing $r2$.

We can assume that the input to the NLG system are descriptions of the COs in some fitting 3D description because the NLG system is embedded into an architect which reasons about construction plans containing these COs (compare Figure 1). Each object is described using properties in 3D space: the bridge b in Figure 2, for example, is described using the coordinates of its corners. In contrast to the simple block laying in Section 4, the relations can also be between several objects and not only between an object and a location. The type of relations expressed can be rather diverse: Besides basic spatial relations such as "left of" and unary relations such as the color, more complex relations were expressed when asked humans to instruct other humans in the Minecraft domain (Osmelak, 2018), such as "other side of" (with respect to an unnamed anchor object) or "same shape as" to express the shape of an object, modulo orientation.

To extract these relations from 3D space, a preprocessor iterates over all tuples of objects and checks which of these should be part of one or several relations. For example, each object is added to the unary relation with the name of its type; otherside is filled with all triples$\langle o_1, o_2, o_3\rangle$ where o_1 and o_3 are on opposite sides of o_2 and both touch o_2. In our example, this is (r1, b, r2) because the two corners of r1 are right above the left corners of b and the corners of r2 are right above the right ones of b.

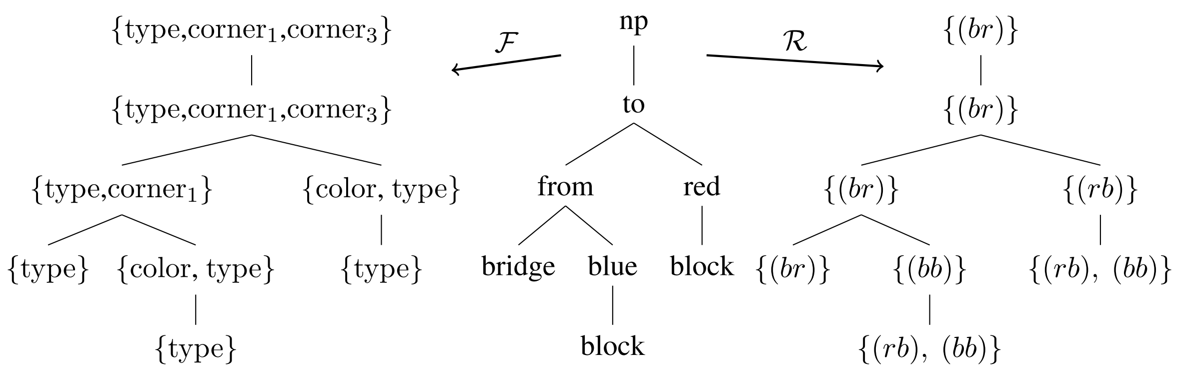

Figure 8: Derivation tree generating "a bridge from the blue block to the red block" (middle), values of the feature interpretation (left) and the referential interpretation (right). The model corresponds to the setting in Figure 2 a), *br* = bridge, *bb* = blue block, *rb* = red block.

5.1 Indefinite reference

As discussed in Section 2, when generating a referring expression for a given object, our aim is to find a set of features which distinguish this object from all other objects we currently *could* but do not want to refer to; for indefinite referents, this set is precomputed (cmp. Figure 3 and Section 2.2). With these distinguishing features sets $\mathcal{D}$, the feature sets that are admissible to be described are all supersets of each of these sets: $\hat{\mathcal{D}} = \{x | \exists d \in \mathcal{D}.x \supseteq d\}$, and the derivation trees for the indefinite REs to a target object o are:

$$L_o = \bigcup_{d \in \hat{\mathcal{D}}} L(G, \text{IndefNP}, \mathcal{R}{:}\{(\text{o})\}, \mathcal{F}{:}d)$$

Internally, the languages do not have to be computed independently; the underlying SIG formalism allows to define a disjunction of target values for interpretations. As before, the REs describing o are the string interpretations of the derivation trees of L_o. Figure 8 shows a derivation tree to generate a RE to the bridge b to generate an instruction from Figure 2 a) to b).

In the SIG (Figure 9), the feature interpretation $\mathcal{F}$ keeps track of which features have been covered: in each rule, the feature interpretation performs a union of the currently described feature with the other features described in child nodes of the derivation tree. Therefore, in the interpretations for $from$, the features from the child noun node are incorporated, but not the features expressed for the blocks used as anchor, as can be seen in Figure 8 (left). $\mathcal{F}$ keeps track about which features are described but not whether their values actually correspond to the target object. Therefore, the referential interpretation $\mathcal{R}$ keeps track of what a

IndefNP $\rightarrow np(\text{N})$
$\mathcal{I}_S(np)(N) = \text{a} \bullet N$
$\mathcal{I}_R(np)(N) = N$
$\mathcal{I}_F(np)(N) = N$

DefNP $\rightarrow dnp(\text{N})$
$\mathcal{I}_S(np)(N) = \text{the} \bullet N$
$\mathcal{I}_R(np)(N) = uniq(N)$
$\mathcal{I}_F(np)(N) = N$

N $\rightarrow from(\text{N}, \text{DetRefBlock})$
$\mathcal{I}_S(from)(N, B) = N \bullet \text{from} \bullet B$
$\mathcal{I}_R(from)(N, B) = [\text{from} \cap_2 B]_1 \cap N$
$\mathcal{I}_F(from)(N, B) = N \cup \{\text{corner}_1\}$

N $\rightarrow to(\text{N}, \text{DetRefBlock})$
$\mathcal{I}_S(to)(N, B) = N \bullet \text{to} \bullet B$
$\mathcal{I}_R(to)(N, B) = [\text{to} \cap_2 B]_1 \cap N$
$\mathcal{I}_F(to)(N, B) = N \cup \{\text{corner}_3\}$

N $\rightarrow otherside(\text{N}, \text{DefNP})$
$\mathcal{I}_S(width)(N, P) = N \bullet \text{on the other side of} \bullet P$
$\mathcal{I}_R(width)(N, P) = [(\text{otherside} \cap_2 \text{P}) \cap_3 \text{it}]_1 \cap N$
$\mathcal{I}_F(width)(N, P) = N \cup \{\text{corner}_1, \text{corner}_2\}$

N $\rightarrow bridge$
$\mathcal{I}_S(bridge) = \text{bridge}$
$\mathcal{I}_R(bridge) = \text{bridge}$
$\mathcal{I}_F(bridge) = \{\text{type}\}$

N $\rightarrow railing$
$\mathcal{I}_S(railing) = \text{railing}$
$\mathcal{I}_R(railing) = \text{railing}$
$\mathcal{I}_F(railing) = \{\text{type}\}$

Figure 9: Excerpt of a grammar to describe a bridge or a railing.

sub-expression refers to when referring to objects in the world (i.e. when generating definite referring expressions such as “the blue block”) and evaluates to the indefinite object if it describes a property of that object. On the referential interpretation, all sets of objects are intersected. If the referential interpretation of a derivation tree t is the target object ($[\![t]\!]_{\mathcal{R}} = \{(o)\}$), we can be sure that each feature described was a property of o as otherwise the interpretation would be empty. This can be observed in Figure 8 (right).

Finally, the RE to instruct the user to build the second railing (Figure 2 c) to d)) is defined by the following derivation tree:

$$t = np(otherside(railing, dnp(bridge)))$$

It refers to the correct object ($[\![t]\!]_{\mathcal{R}} = \{(r2)\}$) and describes a distinguishing set of features: $[\![t]\!]_{\mathcal{F}} = \{type, corner_1, corner_2\}$. The resulting RE is $[\![t]\!]_{\mathcal{S}} =$ “a railing on the other side of the bridge”.

6 Conclusion

We have introduced a formalization for indefinite referring expressions and exemplified it in the Minecraft instruction giving domain. This formalization is able to generate definite referring expressions for locations and for objects, as well as indefinite referring expressions, which contain definite referring sub-expressions. For this, we have extended a system for generating REs based on SIGs to produce relational definite REs for objects in the world and uses these as building blocks for indefinite REs, as they anchor the target object into the world. The method proposed allows to both keep track of the properties described for the target object to generate indefinite REs and to generate definite REs by intersecting sets of possible referents until only one object is left.

The grammar itself only defines which REs are legal, it does not define any preferences. It is possible to define functions scoring certain rules applications and we implemented a preference for the smallest derivation tree, but this is clearly not sufficient. For example, “twenty blocks left of the blue block” and “to the right of the red block” might denote the same target and their derivation trees have the same size but using the first RE is probably less likely to succeed than one using the second. These differences can be captured by a scoring model for the grammar (Garoufi and Koller, 2014) and we will work on learning one from interactions with players.

Our grammar and code is available at `https://minecraft-saar.github.io`.

Acknowledgments We would like to thank Paula Stadnikowa for working on early versions of these ideas. This work was supported by the German Research Foundation (DFG), SFB/CRC 1102 “Information density and linguistic encoding” (Project A7).

References

Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison, Marc Tommasi, and Christof Löding. 2007. *Tree Automata techniques and applications*. published online - `http://tata.gforge.inria.fr/`.

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

Kees van Deemter. 2016. *Computational Models of Referring: A study in cognitive science*. MIT Press.

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

Konstantina Garoufi and Alexander Koller. 2014. Generation of effective referring expressions in situated context. *Language, Cognition, and Neuroscience*, 29(8):986–1001.

Irene Heim. 1982. *The semantics of definite and indefinite noun phrases*. Ph.D. thesis, University of Massachusetts, Amherst.

Klaus von Heusinger. 2000. The reference of indefinites. In Klaus von Heusinger and Urs Egli, editors, *Reference and Anaphoric Relations*, pages 247–265. Springer Netherlands, Dordrecht.

Hans Kamp and Uwe Reyle. 1993. *From Discourse to Logic*. Springer.

Alexander Koller and Nikos Engonopoulos. 2017. Integrated sentence generation with charts. In *Proceedings of the 10th International Conference on Natural Language Generation (INLG)*, Santiago de Compostela.

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

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

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

Doreen Osmelak. 2018. Human experiments in minecraft. Master's thesis, Universität des Saarlandes.

Julia Wichlacz, Alvaro Torralba, and Jörg Hoffmann. 2019. Construction-planning models in minecraft. In *Proceedings of ICAPS workshop on Hierarchical Planning*.