# Referring Expressions as Formulas of Description Logic


## Abstract

In this paper, we propose to reinterpret the problem of generating referring expressions (GRE) as the problem of computing a formula in a description logic that is only satisfied by the referent. This view offers a new unifying perspective under which existing GRE algorithms can be compared. We also show that by applying existing algorithms for computing simulation classes in description logic, we can obtain extremely efficient algorithms for relational referring expressions without any danger of running into infinite regress.


## 1 Introduction

The generation of referring expressions (GRE) is one of the most active and successful research areas in natural language generation. Building upon Dale and Reiter's work (Dale, 1989; Dale and Reiter, 1995), various researchers have added extensions such as reference to sets (Stone, 2000), more expressive logical connectives (van Deemter, 2001), and relational expressions (Dale and Haddock, 1991).

Referring expressions (REs) involving relations, in particular, have received increasing attention recently, particularly in the context of spatial referring expressions in situated generation (e.g. (Kelleher and Kruijff, 2006)), where it seems particularly natural to use expressions such as "the book on the table". However, the classical algorithm by Dale and Haddock (1991) was recently shown to be unable to generate satisfying REs in practice (Viethen and Dale, 2006). Furthermore, the Dale and Haddock algorithm and most of its successors (Krahmer et al., 2003; Kelleher and Kruijff, 2006) are vulnerable to the problem of "infinite regress", where the algorithm jumps back and forth between generating descriptions for two related individuals infinitely, as in "the book on the table which supports a book on the table ...".

In this paper, we propose to view GRE as the problem of computing a formula of description logic (DL) that denotes exactly the set of individuals that we want to refer to. This very natural idea has been mentioned in passing before (Krahmer et al., 2003; Gardent and Striegnitz, 2007); however, we take it one step further by proposing DL as an interlingua for comparing the REs produced by different approaches to GRE. In this way, we can organize existing GRE approaches in an expressiveness hierarchy. For instance, the classical Dale and Reiter algorithms compute purely conjunctive formulas; van Deemter (2001) extends this language by adding the other propositional connectives, whereas Dale and Haddock (1991) extends it by allowing existential quantification.

Furthermore, the view of GRE as a problem of computing DL formulas with a given extension allows us to apply existing algorithms for the latter problem to obtain efficient algorithms for GRE. We present algorithms that compute such formulas for the description logics $\mathcal{EL}$ (which allows only conjunction and existential quantification) and $\mathcal{ALC}$ (which also allows negation). These algorithms effectively compute REs for all individuals in the domain at the same time, which allows them to systematically avoid the infinite regress problem. The $\mathcal{EL}$ algorithm is capable of generating 67% of the relational REs in the Viethen and Dale (2006) dataset, in about 15 milliseconds. The $\mathcal{ALC}$ algorithm is even faster; it computes relational REs for all 100 individuals in a random model in 140 milliseconds.

The paper is structured as follows. In Section 2, we will first define description logics. We will then show how to generate REs by computing DL similarity sets for $\mathcal{ALC}$ and $\mathcal{EL}$ in Section 3. In Section 4, we evaluate our algorithms and discuss our results. Section 5 compares our approach to related research; in particular, it shows how various prominent GRE algorithms fit into the DL framework. Section 6 concludes and points to future work.

## 2 Description logics and similarity

In this paper, we will represent referring expressions as formulas of description logic (Baader et al., 2003). In order to make this point, we will now define the two description logics we will be working with: $\mathcal{ALC}$ and $\mathcal{EL}$.

*Formulas* (or *concepts*) $\varphi$ of $\mathcal{ALC}$ are generated by the following grammar:

$$\varphi, \varphi' ::= p \mid \neg\varphi \mid \varphi \sqcap \varphi' \mid \exists R.\varphi$$

where $p$ is in the set of propositional symbols $\mathsf{prop}$, and $R$ is in the set of relational symbols $\mathsf{rel}$. $\mathcal{EL}$ is the negation-free fragment of $\mathcal{ALC}$.

Formulas of both $\mathcal{ALC}$ and $\mathcal{EL}$ are interpreted in ordinary relational first-order models $\mathcal{M} = (\Delta, \|\cdot\|)$ where $\Delta$ is a non-empy set and $\|\cdot\|$ is an interpretation function such that:

$$\begin{array}{rcl} \|p\| & \subseteq & \Delta \text{ for } p \in \mathsf{prop} \\ \|R\| & \subseteq & \Delta \times \Delta \text{ for } R \in \mathsf{rel} \\ \|\neg\varphi\| & = & \Delta - \|\varphi\| \\ \|\varphi \sqcap \varphi'\| & = & \|\varphi\| \cap \|\varphi'\| \\ \|\exists R.\varphi\| & = & \{i \mid \text{for some } i', (i,i') \in \|R\| \\ & & \text{and } i' \in \|\varphi\|\}. \end{array}$$

Every formula of a description logic denotes a set of individuals in the domain; thus we can use such formulas to describe sets. For instance, in the model in Fig. 1b, the formula $\mathsf{flower}$ denotes the set $\{f_1, f_2\}$; the formula $\mathsf{flower} \sqcap \exists\mathsf{in.hat}$ denotes $\{f_2\}$; and the formula $\mathsf{flower} \sqcap \neg\exists\mathsf{in.hat}$ denotes $\{f_1\}$.

Different description logics differ in the inventory of logical connectives they allow: While $\mathcal{ALC}$ permits negation, $\mathcal{EL}$ doesn't. There are many other description logics in the literature; some that we will get back to in Section 5 are $\mathcal{CL}$ ($\mathcal{EL}$ without existential quantification, i.e., only conjunctions of atoms); $\mathcal{PL}$ ($\mathcal{ALC}$ without existential quantification, i.e., propositional logic); and $\mathcal{ELU}_{(\neg)}$ ($\mathcal{EL}$ plus disjunction and atomic negation).

Below, we will use a key notion of formula preservation that we call *similarity*. For any DL $\mathcal{L}$, we will say that an individual $i$ is *$\mathcal{L}$-similar* to $i'$ in a given model $\mathcal{M}$ if for any formula $\varphi \in \mathcal{L}$ such that $i \in \|\varphi\|$, we also have $i' \in \|\varphi\|$. Equivalently, there is no $\mathcal{L}$-formula that holds of $i$ but not of $i'$. We say that the *$\mathcal{L}$-similarity set* of some individual $i$ is the set of all individuals to which $i$ is $\mathcal{L}$-similar.

Notice that similarity is not necessarily a symmetrical relation: For instance, $f_1$ is $\mathcal{EL}$-similar to $f_2$ in Fig. 1b, but $f_2$ is not $\mathcal{EL}$-similar to $f_1$ (it satisfies the formula $\exists\mathsf{in.hat}$ and $f_1$ doesn't). However, $\mathcal{ALC}$-similarity is a symmetrical relation because the language contains negation; and indeed, $f_1$ is not $\mathcal{ALC}$-similar to $f_2$ either because it satisfied $\neg\exists\mathsf{in.hat}$. Because $\mathcal{ALC}$ is more expressive than $\mathcal{EL}$, it is generally possible for some individual $a$ to be $\mathcal{EL}$-similar but not $\mathcal{ALC}$-similar to some individual $b$, but not vice versa.

## 3 Generating referring expressions

Now we apply description logic to GRE. The core claim of this paper is that it is natural and useful to view the GRE problem as the problem of computing a formula of description logic whose extension is a given target set of individuals.

| $\mathcal{L}$-GRE PROBLEM | |
|---|---|
| Input: | A model $\mathcal{M}$ and a target set $A \subseteq \Delta$. |
| Output: | A formula $\varphi \in \mathcal{L}$ such that $\|\varphi\| = A$ (if such a formula exists). |

In the examples above, it is because $\mathsf{flower} \sqcap \exists\mathsf{in.hat}$ denotes exactly $\{f_2\}$ that we can say "the flower in the hat" to refer to $f_2$. This perspective provides a general framework into which many existing GRE approaches fit: Traditional attribute selection (Dale and Reiter, 1995) corresponds to building DL formulas that are conjunctions of atoms; relational REs as in Dale and Haddock (1991) are formulas of $\mathcal{EL}$; and so on. We will further pursue the idea of organizing GRE approaches with respect to the variant of DL they use in Section 5.

For the rest of this paper, we assume that we are generating a singular RE, i.e. the target referent $A$ will be a singleton set. In this case, we will only be able to generate a formula that denotes exactly

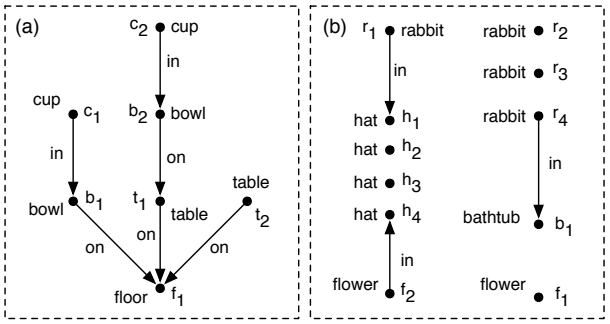


Figure 1: (a) The Dale and Haddock (1991) scenario; (b) the Stone and Webber (1998) scenario.

$A = \{a\}$ (i.e., a RE that uniquely refers to $a$) if there is no other individual $b$ to which $a$ is similar; otherwise, any formula that is satisfied by $a$ is also satisfied by $b$. Conversely, if we know that $a$ is not similar to any other individual, then there is a formula that is satisfied by $a$ and not by anything else; this formula can serve as a unique singular RE. In other words, we can reduce the $\mathcal{L}$-GRE problem for a given model to the problem of computing the $\mathcal{L}$-similarity sets of this model.

In the rest of this section, we will present algorithms that compute the similarity sets of a given model for $\mathcal{ALC}$ and $\mathcal{EL}$, together with characteristic formulas that denote them. In the $\mathcal{ALC}$ case, we adapt a standard algorithm from the literature for computing *simulation classes*; we will then further adapt this algorithm for $\mathcal{EL}$. In effect, both algorithms compute REs for all individuals in some model at the same time – very efficiently and without any danger of infinite regress.

### 3.1 Computing similarity sets

It can be shown that for $\mathcal{ALC}$, the similarity sets of a finite model coincide exactly with the *simulation classes* of this model. Simulation classes have been studied extensively in the literature (see e.g. Blackburn et al. (2001); Kurtonina and de Rijke (1998)), and there are several efficient algorithms for computing $\mathcal{ALC}$-simulation classes (Hopcroft, 1971; Paige and Tarjan, 1987; Dovier et al., 2004). However, these algorithms will only compute the simulation classes themselves. Here we extend the Hopcroft (1971) algorithm such that it computes, along with each set, also a formula that denotes exactly this set. We can then use these formulas as representations of the referring expressions.

The pseudocode for our $\mathcal{ALC}$ algorithm is shown as Algorithm 1 and Algorithm 2. Given a model $\mathcal{M} = (\Delta, \|\cdot\|)$, the algorithm computes a set $RE$ of $\mathcal{ALC}$ formulas such that $\{\|\varphi\| \mid \varphi \in RE\}$ is the set of $\mathcal{ALC}$-similarity sets of $\mathcal{M}$. The algorithm starts with $RE = \{\top\}$ (where $\|\top\| = \Delta$), and successively refines $RE$ by making its elements denote smaller and smaller sets. It maintains the invariant that at the start and end of every iteration, $\{\|\varphi\| \mid \varphi \in RE\}$ is always a partition of $\Delta$. The algorithm iterates over all propositional and relational symbols in prop and rel to construct new formulas until either all formulas in $RE$ denote singletons (i.e., there is only one individual that satisfies them), or no progress has been made in the previous iteration. In each iteration, it calls the procedure $\text{add}_{\mathcal{ALC}}(\varphi, RE)$, which intersects $\varphi$ with any formula $\psi \in RE$ which does not denote a singleton and which is not equivalent to $\varphi$ and to $\neg\varphi$. In this case, it replaces $\psi$ in $RE$ by $\psi \sqcap \varphi$ and $\psi \sqcap \neg\varphi$.

The $\mathcal{ALC}$ algorithm computes the $\mathcal{ALC}$-similarity sets of the model in time $O(n^3)$, where $n$ is the number of individuals in the domain. However, it will freely introduce negations in the case distinctions, which can make the resulting formula hard to realize (see also Section 4.3). This is why we also present an algorithm for the $\mathcal{EL}$-similarity sets; $\mathcal{EL}$ corresponds to positive relational REs, which are generally much easier to realize.

We obtain the $\mathcal{EL}$ algorithm by replacing the call to $\text{add}_{\mathcal{ALC}}$ in Algorithm 1 by a call to $\text{add}_{\mathcal{EL}}$, which is defined in Algorithm 3. As before, the algorithm maintains a set $RE = \{\varphi_1, \ldots, \varphi_n\}$ of formulas (this time of $\mathcal{EL}$) such that $\|\varphi_1\| \cup \ldots \cup \|\varphi_n\| = \Delta$, and which it refines iteratively. However, where the $\mathcal{ALC}$ algorithm maintains the invariant that $\|\varphi_1\|, \ldots, \|\varphi_n\|$ is a partition of $\Delta$, we weaken this invariant to the requirement that there are no $m \geq 2$ pairwise different indices $1 \leq i_1, \ldots, i_m \leq n$ such that $\|\varphi_{i_1}\| = \|\varphi_{i_2}\| \cup \ldots \cup \|\varphi_{i_m}\|$. We call $\varphi_{i_1}$ *subsumed* if such a decomposition exists.

Because it maintains a weaker invariant, the set $RE$ may contain more formulas at the same time in the $\mathcal{EL}$ algorithm than in the $\mathcal{ALC}$ algorithm. Prima facie, this means that it has worst-case exponential runtime, given that the whole domain has an exponential number of subsets. However, it is possible

**Algorithm 1**: Computing the $\mathcal{L}$-similarity sets

**Input**: A model $\mathcal{M} = (\Delta, \|\cdot\|)$
**Output**: A set $RE$ of formulas such that $\{\|\varphi\| \mid \varphi \in RE\}$ is the set of $\mathcal{L}$-similarity sets of $\mathcal{M}$.

```
RE ← {⊤}
for p ∈ prop do
    add_L(p, RE)
while exists some φ ∈ RE, ||φ||^M > 1 do
    for φ ∈ RE, R ∈ rel do
        add_L(∃R.φ, RE)
    if made no changes to RE then
        exit
```

**Algorithm 2**: $\text{add}_{\mathcal{ALC}}(\varphi, RE)$

```
for ψ ∈ RE with |||ψ||| > 1 do
    if ||ψ ⊓ φ|| ≠ ∅ and ||ψ ⊓ ¬φ|| ≠ ∅ then
        add ψ ⊓ φ and ψ ⊓ ¬φ to RE;
        remove ψ from RE;
```

**Algorithm 3**: $\text{add}_{\mathcal{EL}}(\varphi, RE)$

```
for ψ ∈ RE with |||ψ||| > 1 do
    if ψ ⊓ φ is not subsumed in RE and
    ||ψ ⊓ φ|| ≠ ∅ and ||ψ ⊓ φ|| ≠ ||ψ|| then
        add ψ ⊓ φ to RE
        remove subsumed formulas from RE
```

that a more careful analysis of the combinatorics of the considered subsets will reveal that the algorithm is actually polynomial.

Notice that both algorithms try to refine $RE$ first according to the propositional symbols, and then by relational expressions of increasing depth. The algorithms would remain correct if we interleaved the addition of propositional and relational symbols; however, the algorithms would become slower because the propositional refinements would have to be percolated through the relational chains in subsequent iterations.

### 3.2 Some examples

We will now see what our algorithms do for some interesting examples. Consider the model shown in Fig. 1a, which is taken from Dale and Haddock (1991), and let's run the $\mathcal{EL}$ algorithm.

The algorithm starts with $RE = \{\top\}$. In the first loop, it adds the formulas floor, bowl, cup, and table, and then removes $\top$ because it is now subsumed. Not all of these formulas denote singletons; for instance, $\|\mathsf{cup}\|$ contains two individuals. So we iterate over the relations to refine our formulas. After the first iteration over the relations, we have $RE = \{\mathsf{floor}, \mathsf{bowl} \sqcap \exists\mathsf{on.floor}, \mathsf{bowl} \sqcap \exists\mathsf{on.table}, \mathsf{cup}, \mathsf{table}\}$. Notice that bowl has become subsumed, but we haven't distinguished the cups and tables further.

Now we can use the split between the bowls to distinguish the cups in the second iteration. The result of this is $RE = \{\mathsf{floor}, \mathsf{bowl} \sqcap \exists\mathsf{on.floor}, \mathsf{bowl} \sqcap \exists\mathsf{on.table}, \mathsf{cup} \sqcap \exists\mathsf{in.}(\mathsf{bowl} \sqcap \exists\mathsf{on.floor}), \mathsf{cup} \sqcap \exists\mathsf{in.}(\mathsf{bowl} \sqcap \exists\mathsf{on.table}), \mathsf{table}\}$. At this point, all formulas except table denote singletons, and further iterations don't allow us to refine table; so the algorithm terminates. Each formula with a singleton extension $\{a\}$ is a unique description of $a$; for instance, $\mathsf{cup} \sqcap \exists\mathsf{in.}(\mathsf{bowl} \sqcap \exists\mathsf{on.table})$ is only satisfied by $c_2$, so we may refer to $c_2$ as "the cup in the bowl on the table". Notice that the algorithm didn't focus on any particular individual; it simultaneously generated REs for all individuals except for the two tables (which are similar to each other).

The $\mathcal{EL}$ algorithm has a harder time with the example in Fig. 1b (Stone and Webber, 1998). While it will correctly identify $r_1$ as "the rabbit in the hat" and $f_2$ as "the flower in the hat", it will not be able to compute a RE for $f_1$ because $f_1$ is $\mathcal{EL}$-similar to $f_2$. Indeed, the algorithm terminates with $RE$ containing both flower and $\mathsf{flower} \sqcap \exists\mathsf{in.hat}$. This is a typical pattern for asymmetrical cases of similarity in $\mathcal{EL}$: If there are two formulas $\varphi_1$ and $\varphi_2$ in the output set with $\|\varphi_1\| \subseteq \|\varphi_2\|$, then there is generally some individual $b \in \|\varphi_2\| - \|\varphi_1\|$ such that all individuals in $\|\varphi_1\|$ are similar to $b$, but not vice versa. By contrast, the $\mathcal{ALC}$ algorithm can exploit the greater expressivity of $\mathcal{ALC}$ to split flower into the two new formulas $\mathsf{flower} \sqcap \exists\mathsf{in.hat}$ and $\mathsf{flower} \sqcap \neg\exists\mathsf{in.hat}$, generating a unique RE for $f_1$ as well.

## 4 Discussion

We will now describe two experiments evaluating the quality of the $\mathcal{EL}$ algorithm's output and the effi-

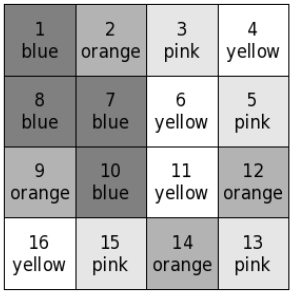


Figure 2: A schematic view of the filing cabinets.

ciency of both of our algorithms, and we discuss the interface between our algorithms and realization.

### 4.1 Evaluation: Output quality

To compare the descriptions generated by our algorithm to those humans produce, we use a corpus of human-generated referring expressions collected and made available by Jette Viethen and Robert Dale.[1] They asked human subjects to describe one of 16 filing cabinet drawers. The drawers had different colors and were arranged in a four-by-four grid (see Fig. 2). The human subjects used four non-relational properties (the drawer's color, its column and row number, and whether it is in a corner) and five relational properties (above, below, next to, left of, right of). Of the 118 referring expressions obtained in the experiment, only 15 use relations.

Viethen and Dale (2006) describe the data in more detail and present results of evaluating the Full Brevity algorithm, the Incremental Algorithm (both by Dale and Reiter (1995)), and the relational algorithm (Dale and Haddock, 1991) on this corpus. The Incremental Algorithm is dependent on a predefined ordering in which properties are added to the description. Viethen and Dale, therefore, try all possible orderings and evaluate what percentage of descriptions an algorithm can generate with any of them. The Full Brevity and the Relational Algorithms choose properties based on their discriminatory power and only use the orderings as tie breakers. Viethen and Dale found that the Incremental Algorithm is capable of generating 98 of the 103 non-relational descriptions. However, the Dale and Haddock algorithm was unable to generate even a single one of the human-generated relational descriptions.

[1] http://www.ics.mq.edu.au/~jviethen/drawers

We replicated Viethen and Dale's experiment for the $\mathcal{EL}$ algorithm presented above. In the non-relational case, our results are the same as theirs for the Incremental Algorithm: the $\mathcal{EL}$ algorithm generates 98 of the 103 non-relational descriptions, using four (of the possible) orderings. This is because the two algorithms perform essentially the same computations if there are no relations.

To evaluate the relational case, we first removed the column and row predicates from the domain, because these predicates make it possible to uniquely identify each drawer with non-relational descriptions, and our algorithm would always prefer these. Using three (of the 240 possible) different orderings, our algorithm was able to generate 10 of the 15 human-produced relational descriptions correctly. This is shown in more detail in Fig. 3; notice that four of the 15 descriptions occurred twice in the corpus. Of the five human-produced descriptions that the $\mathcal{EL}$ algorithm cannot generate, two mention column information, which we excluded from our domain, and three involve references to sets (*the two blues ones in horizontal sequence*/*the two yellow drawers*). While our algorithm cannot reproduce those descriptions, it does generate other, simpler descriptions for them.

### 4.2 Evaluation: Efficiency

Both the $\mathcal{EL}$ and the $\mathcal{ALC}$ algorithms took about 15 milliseconds to compute distinguishing formulas for all 16 individuals in the Viethen and Dale dataset.[2]

In order to get a more comprehensive picture of the algorithms' efficiency, we ran them on random models with increasing numbers of individuals. Each model had random interpretations for ten different propositional and four relational symbols; each individual had a 10% chance to be in the extension of each propositional symbol, and each pair of individuals had a 10% chance to be related by a relational symbol. The results (averaged over 10 runs for each model size) are shown in Fig. 4. As the figure shows, the $\mathcal{EL}$ algorithm takes about 350 ms on average to generate relational REs for all individuals in the model of size 100, i.e. less than 4 ms on aver-

[2] Runtimes are meassured on a MacBook Pro (Intel Core 2 Duo, 2.16 GHz) running Java 1.6 beta, and we allowed the Java VM to warm up, i.e., just-in-time compile all bytecode, before taking the measurements.

| id | *human-produced description*<br>output of the $\mathcal{EL}$ algorithm |
|---|---|
| 2 | *the orange drawer above the blue drawer*<br>**orange ⊓ ∃above.blue** / orange ⊓ ∃above.(∃below.(orange) ⊓ blue) / orange ⊓ ∃next.(blue) ⊓ ∃next.(pink) |
| 4 | *the yellow drawer on the top of the pink one*<br>**yellow ⊓ ∃above.pink** / yellow ⊓ corner ⊓ ∃above.pink / yellow ⊓ corner ⊓ ∃above.(∃next.(yellow) ⊓ pink) |
| 5 | ∗ *the pink drawer in the fourth column below the yellow one*<br>pink ⊓ ∃above.orange / pink ⊓ ∃below.yellow / pink ⊓ ∃next.(yellow) ⊓ ∃above.(∃next.(yellow) ⊓ orange) |
| 6 | *the yellow drawer on top of the yellow drawer* (2×) / ∗ *the drawer after the two blue ones in horizontal sequence*<br>**yellow ⊓ ∃above.yellow** / yellow ⊓ ∃below.pink / yellow ⊓ ∃next.(blue) ⊓ ∃next.(pink) |
| 7 | *the blue drawer below the orange one* / ∗ *the blue drawer below the orange drawer in the second column*<br>blue⊓∃above.(blue)⊓∃next.(∃above.(orange)⊓blue) / **blue⊓∃below.(orange)** / blue⊓∃next.(blue)⊓∃next.(yellow) |
| 10 | *the blue drawer above the pink drawer* (2×)<br>**blue ⊓ ∃above.(pink)** / blue ⊓ ∃above.(pink) ⊓ ∃below.(blue) / blue ⊓ ∃next.(orange) ⊓ ∃next.(yellow) |
| 11 | *the yellow drawer next to the orange drawer* (2×)<br>yellow ⊓ ∃above.orange / yellow ⊓ ∃below.yellow / **yellow ⊓ ∃next.orange** |
| 12 | *the orange drawer below the pink drawer*<br>orange ⊓ ∃above.(pink ⊓ corner) / **orange ⊓ ∃below.pink** / orange ⊓ ∃next.yellow |
| 14 | ∗ *the orange drawer below the two yellow drawers* (2×)<br>orange ⊓ ∃next.(pink ⊓ corner) ⊓ ∃next.(pink) / orange ⊓ ∃below.yellow / orange ⊓ ∃next.(pink ⊓ corner) |

Figure 3: The relational descriptions from Viethen and Dale (2006), annotated with the drawer id and the outputs of the $\mathcal{EL}$ algorithm using three different orderings. Descriptions that the $\mathcal{EL}$ algorithm cannot generate with any ordering are marked by $*$. Generated descriptions that match one produced by humans are in boldface.

age for each individual. The $\mathcal{ALC}$ algorithm is even faster, at about 140 ms for the model of size 100. As far as we know, these are by far the fastest published runtimes for any relational GRE algorithm in the literature.

Another encouraging aspect of both runtime graphs is that they seem to be approximated well by quadratic functions. So although the worst-case runtime of the $\mathcal{EL}$ algorithm might be exponential, both algorithms empirically exhibit (low) polynomial runtime on the inputs we considered.

### 4.3 Interface to realization

Our GRE algorithms do not guarantee that the formula they compute can actually be realized in language. For example, none of the formulas our algorithms computed in the Viethen and Dale domain contained an atom that would commonly be realized as a noun; the property drawer is never used because it applies to all individuals in the domain. This particular problem could easily be worked around in a post-processing step. However, another problem arises from the massive use of negation in the $\mathcal{ALC}$ algorithm; it will be hard for any realizer to find a reasonable way of expressing a formula like $\neg\exists R.(\neg P \sqcap \neg Q)$ as a smooth noun phrase. Although we agree with van Deemter (2001) and others that the careful use of negation and disjunction can improve REs, these connectives must not be overused. Thus we consider the formulas computed by the $\mathcal{EL}$ algorithm "safer" with respect to realization.

Of course, we share the problem of interfacing GRE and realization with every other approach that separates these two modules, i.e. almost the entire GRE literature (notable exceptions are e.g. Horacek (1997) and SPUD (Stone and Webber, 1998)). In principle, we believe that it is a good idea to handle sentence planning and realization in a single module; for instance, SPUD can use its awareness of the syntactic context to generate succinct REs as in "take the rabbit from the hat". We hope that the ideas we have explored here for efficient and expressive RE generation can eventually be combined with recent efficient algorithms for integrated sentence planning and realization, such as in Koller and Stone (2007).

One problem that arises in our approach specifically is that both algorithms derive some measure of efficiency from their freedom to build formulas without having to respect any linguistic constraints. It seems straightforward, for instance, to extend Krahmer et al.'s (2003) approach such that it

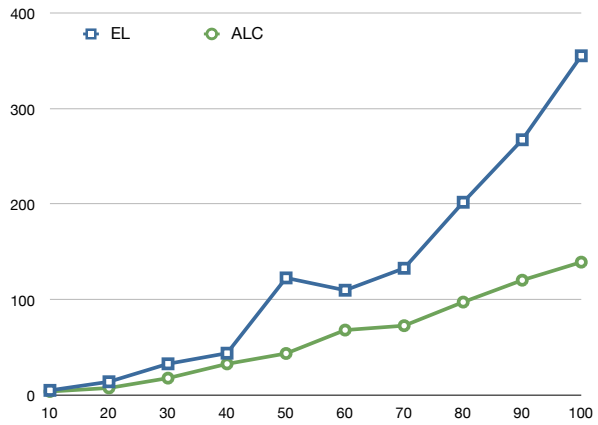


Figure 4: Average runtimes (in ms) of the two algorithms on random models with different numbers of individuals.

only considers subgraphs that can actually be realized, because their algorithm proceeds by a genuine search for uniquely identifying subgraphs, and will simply take a different branch of the search if some subgraph is useless. This would be harder in our case. Our algorithms don't search in the same way; if we disallow certain refinements of a partition, we have to allow the algorithms to backtrack and thus jeopardize the worst-case polynomial runtime. Investigating this interplay between efficiency and linguistic constraints is an interesting avenue for future research.

## 5 A unified perspective on GRE

Viewing GRE as a problem of generating DL formulas offers a unified perspective of the GRE problem: It is the problem of computing a DL formula with a given extension. Many existing approaches can be subsumed under this view; we have summarized this for some of them in Fig. 5, along with the DL fragment they use. We already discussed some of these approaches in Section 3. Furthermore, the non-relational but negative and disjunctive descriptions generated by van Deemter (2001) are simply formulas of $\mathcal{PL}$; and Gardent (2002) generalizes this into generating formulas of $\mathcal{ELU}_{(\neg)}$, i.e., $\mathcal{EL}$ plus disjunction and atomic negation. The approach presented here fits well into this landscape, and it completes the picture by showing how to generate REs in $\mathcal{ALC}$, which combines all connectives used in any of these previous approaches.

| GRE algorithm | DL variant |
|---|---|
| Dale and Reiter (1995) | $\mathcal{CL}$ |
| van Deemter (2001) | $\mathcal{PL}$ |
| Dale and Haddock (1991) | $\mathcal{EL}$ |
| Kelleher and Kruijff (2006) | $\mathcal{EL}$ |
| Gardent (2002) | $\mathcal{ELU}_{(\neg)}$ |

Figure 5: DL variants used by different GRE algorithms.

Where our approach breaks new ground is in the way these formulas are computed: It successively refines a decomposition of the domain into subsets. In this way, it is reminiscent of the Incremental Algorithm, which in fact can be seen as the special case of the $\mathcal{EL}$ algorithm for the non-relational case. However, unlike Dale and Haddock (1991) and its successors, such as Kelleher and Kruijff (2006), we do not have to take special precautions to avoid infinite regress. While Dale and Haddock's algorithm attempts to generate a RE for a single individual for successive individuals in the model, our algorithms consider all individuals in parallel and are guaranteed to always terminate.

Perhaps closest in spirit to our approach is Krahmer et al.'s graph algorithm (2003), which also computes REs by extending them successively. However, their subgraphs go beyond the expressive power of $\mathcal{ALC}$ in that they can distinguish between "the dog that bites a dog" and "the dog that bites itself". The price they pay for this increase in expressive power is an NP-complete worst-case complexity. Interestingly, Krahmer et al. themselves discuss the possibility of seeing their subgraphs as formulas of hybrid logic which are satisfied at the points where the subgraph can be embedded; and hybrid logics can be seen as very expressive description logics (Areces and ten Cate, 2006) as well.

## 6 Conclusion

In this paper, we have explored the idea of viewing the generation of singular REs as the problem of computing a DL formula with a given extension. We have shown how such formulas can be computed efficiently (for $\mathcal{ALC}$ and $\mathcal{EL}$) by adapting existing algorithms from the literature. The $\mathcal{EL}$ algorithm is able to generate 95% of the non-relational and 67% of the relational REs from Viethen and Dale (2006). Both algorithms are extremely efficient (350 ms and

140 ms respectively to generate relational REs for all individuals in a random model with 100 individuals); to our knowledge, these are by far the fastest runtimes for relational GRE reported in the literature. We will make our implementation available online.

Because they compute referring expressions for all individuals in the domain at once, our algorithms will perform especially strongly in static settings, such as the generation of descriptions for museum exhibits, in which the individuals and their properties don't change much. However, even in more dynamic settings, our algorithms have a chance to outperform search algorithms like Dale and Haddock's in the average case because they can't get stuck in unproductive branches of the search space. Nevertheless, one interesting question for future research is how to incrementally update simulation classes when the model changes. Similarly, it would be interesting to explore how linguistic constraints and different attribute orderings can be taken into account without slowing the algorithms down, and how the algorithms can be adapted to compute REs for sets. In exploring these extensions we will be able to draw on a rich body of literature that considers many variants of simulation algorithms.

In experimenting with the Viethen and Dale data, we found that there is no single ordering that covers all human-produced descriptions, which seems to be in contrast to Dale and Reiter's (1995) assumption that there is only one ordering for each given domain. In fact, it is not even the case that each speaker consistently uses just one ordering. An interesting open research question is thus what factors determine which ordering is used. Unfortunately, both in the Viethen and Dale dataset and in the TUNA corpus (van Deemter et al., 2006), only a minority of referring expressions is relational, maybe because these domains lend themselves very well to row/column style propositional REs. We are currently preparing an experiment to collect REs in a domain in which relational REs will be more frequent.

## References


C. Areces and B. ten Cate. 2006. Hybrid logics. In P. Blackburn, F. Wolter, and J. van Benthem, editors, *Handbook of Modal Logics*. Elsevier.

F. Baader, D. McGuiness, D. Nardi, and P. Patel-Schneider, editors. 2003. *The Description Logic Handbook: Theory, implementation and applications*. Cambridge University Press.

P. Blackburn, M. de Rijke, and Y. Venema. 2001. *Modal Logic*. Cambridge University Press.

R. Dale and N. Haddock. 1991. Generating referring expressions involving relations. In *Proc. of the 5th EACL*.

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

R. Dale. 1989. Cooking up referring expressions. In *Proc. of the 27th ACL*.

A. Dovier, C. Piazza, and A. Policriti. 2004. An efficient algorithm for computing bisimulation equivalence. *Theoretical Computer Science*, 311(1–3).

C. Gardent and K. Striegnitz. 2007. Generating bridging definite descriptions. In H. Bunt and R. Muskens, editors, *Computing Meaning, Vol. 3*. Springer.

C. Gardent. 2002. Generating minimal definite descriptions. In *Proc. of the 40th ACL*.

J. Hopcroft. 1971. An $n \log(n)$ algorithm for minimizing states in a finite automaton. In Z. Kohave, editor, *Theory of Machines and computations*. Academic Press.

H. Horacek. 1997. An algorithm for generating referential descriptions with flexible interfaces. In *Proc. of the 35th ACL*.

J. Kelleher and G.-J. Kruijff. 2006. Incremental generation of spatial referring expressions in situated dialog. In *Proc. of COLING/ACL*.

A. Koller and M. Stone. 2007. Sentence generation as planning. In *Proc. of the 45th ACL*.

E. Krahmer, S. van Erk, and A. Verleg. 2003. Graph-based generation of referring expressions. *Computational Linguistics*, 29(1).

N. Kurtonina and M. de Rijke. 1998. Expressiveness of concept expressions in first-order description logics. *Artificial Intelligence*, 107.

R. Paige and R. Tarjan. 1987. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6).

M. Stone and B. Webber. 1998. Textual economy through close coupling of syntax and semantics. In *Proc. of the 9th INLG workshop*.

M. Stone. 2000. On identifying sets. In *Proc. of the 1st INLG*.

K. van Deemter, I. van der Sluis, and A. Gatt. 2006. Building a semantically transparent corpus for the generation of referring expressions. In *Proc. of the 4th INLG*.

K. van Deemter. 2001. Generating referring expressions: Beyond the incremental algorithm. In *Proc. of the 4th IWCS*.

J. Viethen and R. Dale. 2006. Algorithms for generating referring expressions: Do they do what people do? In *Proc. of the 4th INLG*.