# Using Logic in the Generation of Referring Expressions

Carlos Areces[1], Santiago Figueira[⋆2], and Daniel Gorín[3]

[1] INRIA Nancy, Grand Est, France
areces@loria.fr
[2] Departamento de Computación, FCEyN, UBA and CONICET, Argentina
[3] Departamento de Computación, FCEyN, UBA, Argentina
{santiago,dgorin}@dc.uba.fr

**Abstract.** The problem of generating referring expressions (GRE) is an important task in natural language generation. In this paper, we advocate for the use of logical languages in the output of the content determination phase (i.e., when the relevant features of the object to be referred are selected). Many different logics can be used for this and we argue that, for a particular application, the actual choice shall constitute a compromise between expressive power (how many objects can be distinguished), computational complexity (how difficult it is to determine the content) and realizability (how often will the selected content be realized to an idiomatic expression). We show that well-known results from the area of computational logic can then be transferred to GRE. Moreover, our approach is orthogonal to previous proposals and we illustrate this by generalizing well-known content-determination algorithms to make them parametric on the logic employed.

## 1 Generating Referring Expressions

The generation of referring expressions (GRE) –given a context and an element in that context generate a grammatically correct expression in a given natural language that uniquely represents the element– is a basic task in natural language generation, and one of active research (see [4,5,6,20,8] among others). Most of the work in this area is focused on the *content determination* problem (i.e., finding a collection of properties that singles out the target object from the remaining objects in the context) and leaves the actual *realization* (i.e., expressing a given content as a grammatically correct expression) to standard techniques[4].

However, there is yet no general agreement on the basic representation of both the input and the output to the problem; this is handled in a rather ad-hoc way by each new proposal instead.

Krahmer et al. [17] make the case for the use of *labeled directed graphs* in the context of this problem: graphs are abstract enough to express a large number of

[⋆] S. Figueira was partially supported by CONICET (grant PIP 370) and UBA (grant UBACyT 20020090200116).

[4] For exceptions to this practice see, e.g., [16,21]

domains and there are many attractive, and well-known algorithms for dealing with this type of structures. Indeed, these are nothing other than an alternative representation of relational models, typically used to provide semantics for formal languages like first and higher-order logics, modal logics, etc. Even valuations, the basic models of propositional logic, can be seen as a single-pointed labeled graph. It is not surprising then that they are well suited to the task.

In this article, we side with [17] and use labeled graphs as input, but we argue that an important notion has been left out when making this decision. Exactly because of their generality, graphs do not define, by themselves, a unique notion of *sameness*. When do we say that two nodes in the graphs can or cannot be referred uniquely in terms of their properties? This question only makes sense once we fix a certain level of expressiveness which determines when two graphs, or two elements in the same graph, are equivalent.

Expressiveness can be formalized using structural relations on graphs (isomorphisms, etc.) or, alternatively, logical languages. Both ways are presented in §2, where we also discuss how fixing a notion of expressiveness impacts on the number of instances of the GRE problem that have a solution; the computational complexity of the GRE algorithms involved; and the complexity of the surface realization problem. We then investigate the GRE problem in terms of different notions of expressiveness. We first explore in §3 how well-known algorithms from computational logic can be applied to GRE. This is a systematization of the approach of [1], and we are able to answer a complexity question that was left open there. In §4 we take the opposite route: we take the well-known GRE-algorithm of [17], identify its underlying expressivity and rewrite in term of other logics. We then show in §5 that both approaches can be combined and finally discuss in §6 the size of an RE relative to the expressiveness employed. We conclude in §7 with a short discussion and prospects for future work.

## 2 Measuring Expressive Power

Relational structures are very suitable for representing *situations* or *scenes*. A relational structure (also called "relational model") is a non-empty set of objects –the *domain*– together with a collection of relations, each with a fixed arity.

Formally, assume a fixed and finite (but otherwise arbitrary) vocabulary of $n$-ary relation symbols.[5] A relational model $\mathcal{M}$ is a tuple $\langle \Delta, \| \cdot \| \rangle$ where $\Delta$ is a nonempty set, and $\| \cdot \|$ is a suitable interpretation function, that is, $\|r\| \subseteq \Delta^n$ for every $n$-ary relation symbol $r$ in the vocabulary. We say that $\mathcal{M}$ is *finite* whenever $\Delta$ is finite. The *size* of a model $\mathcal{M}$ is the sum $\#\Delta + \#\| \cdot \|$, where $\#\Delta$ is the cardinality of $\Delta$ and $\#\| \cdot \|$ is the sum of the sizes of all relations in $\| \cdot \|$.

Figure 1 below shows how we can represent a scene as a relational model. Intuitively, $a$, $b$ and $d$ are dogs, while $c$ and $e$ are cats; $d$ is a small beagle; $b$ and $c$ are also small. We read *sniffs*$(d, e)$ as "*$d$ is sniffing $e$*".

Logical languages are fit for the task of (formally) *describing* elements of a relational structure. Consider, e.g., the classical language of first-order logic

[5] Constants and function symbols can be represented as relations of adequate arity.

$$\begin{aligned} \Delta &= \{a, b, c, d, e\} \\ \|\mathit{dog}\| &= \{a, b, d\} \\ \|\mathit{cat}\| &= \{c, e\} \\ \|\mathit{beagle}\| &= \{d\} \\ \|\mathit{small}\| &= \{b, c, d\} \\ \|\mathit{sniffs}\| &= \{(a, a), (b, a), (c, b), (d, e), (e, d)\} \end{aligned}$$

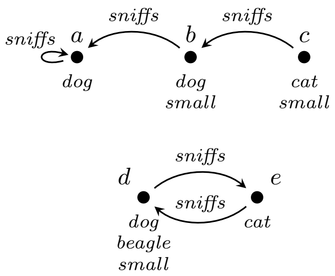

**Fig. 1.** Graph representation of scene $\mathcal{S}$.

(with equality), $\mathcal{FO}$, given by:

$$\top \mid x_i \not\approx x_j \mid r(\bar{x}) \mid \neg\gamma \mid \gamma \wedge \gamma' \mid \exists x_i.\gamma$$

where $\gamma, \gamma' \in \mathcal{FO}$, $r$ is an $n$-ary relation symbol and $\bar{x}$ is an $n$-tuple of variables. As usual, $\gamma \vee \gamma'$ and $\forall x.\gamma$ are short for $\neg(\neg\gamma \wedge \neg\gamma')$ and $\neg\exists x.\neg\gamma$, respectively. Formulas of the form $\top$, $x_i \not\approx x_j$ and $r(\bar{x})$ are called *atoms*.[6] Given a relational model $\mathcal{M} = \langle \Delta, \|\cdot\| \rangle$ and a formula $\gamma$ with free variables[7] among $x_1 \dots x_n$, we inductively define the *extension* or *interpretation* of $\gamma$ as the set of $n$-tuples $\|\gamma\|^n \subseteq \Delta^n$ that satisfy:

$$\begin{aligned} \|\top\|^n &= \Delta^n & \|x_i \not\approx x_j\|^n &= \{\bar{a} \mid \bar{a} \in \Delta^n, a_i \neq a_j\} \\ \|\neg\delta\|^n &= \Delta^n \setminus \|\delta\|^n & \|r(x_{i_1} \dots x_{i_k})\|^n &= \{\bar{a} \mid \bar{a} \in \Delta^n, (a_{i_1} \dots a_{i_k}) \in \|r\|\} \\ \|\delta \wedge \theta\|^n &= \|\delta\|^n \cap \|\theta\|^n & \|\exists x_l.\delta\|^n &= \{\bar{a} \mid \bar{a}e \in \|\delta'\|^{n+1} \text{ for some } e\} \end{aligned}$$

where $1 \leq i, j, i_1, \dots, i_k \leq n$, $\bar{a} = (a_1 \dots a_n)$, $\bar{a}e = (a_1 \dots a_n, e)$ and $\delta'$ is obtained by replacing all occurrences of $x_l$ in $\delta$ by $x_{n+1}$. When the cardinality of the tuples involved is known from the context we will just write $\|\gamma\|$ instead of $\|\gamma\|^n$.

With a language syntax and semantics in place, we can now formally define the problem of $\mathcal{L}$-GRE for a target set of elements $T$ (we slightly adapt the definition in [1]):

$\mathcal{L}$-GRE PROBLEM

Input: a model $\mathcal{M} = \langle \Delta, \|\cdot\| \rangle$ and a nonempty target set $T \subseteq \Delta$.
Output: a formula $\varphi \in \mathcal{L}$ such that $\|\varphi\| = T$ if it exists, and $\perp$ otherwise.

When the output is not $\perp$, we say that $\varphi$ is an *$\mathcal{L}$-referring expression ($\mathcal{L}$-RE) for $T$ in $\mathcal{M}$*. Simply put then, the output of the $\mathcal{L}$-GRE problem is a formula of $\mathcal{L}$ whose interpretation in the input model is the target set, if such a formula exists. This definition applies also to the GRE for objects of the domain by taking a singleton set as target.

[6] For technical reasons, we include the inequality symbol $\not\approx$ as primitive. Equality can be defined using negation.

[7] W.l.o.g. we assume that no variable appears both free and bound, that no variable is bound twice, and that the index of bound variables in a formula increases from left to right.

By using formulas with $n$ free variables one could extend this definition to describe $n$-ary relations; but here we are only interested in describing subsets of the domain. Actually, we shall restrict our attention a little further:

**Convention 1.** We will only consider relational models with unary and binary relation symbols (i.e., labeled graphs). We will consistently use $p$ for a unary relation symbol (and called it a *proposition*) and $r$ for a binary relation symbol.

This convention captures the usual models of interest when describing scenes as the one presented in Figure 1. Accommodating relations of higher arity in our theoretical framework is easy, but it might affect computational complexity.

### 2.1 Choosing the Appropriate Language

Given a model $\mathcal{M}$, there might be an infinite number of formulas that uniquely describe a target (even formulas which are not logically equivalent might have the same interpretation once a model is fixed). Despite having the same interpretation in $\mathcal{M}$, they may be quite different with respect to other parameters.

As it is well known in the automated text generation community, different realizations of the same content might result in expressions which are more or less appropriate in a given context. Although, as we mentioned in the introduction, we will only address the content determination part (and not the surface realization part) of the GRE problem, we will argue that generating content using languages with different expressive power can have an impact in the posterior surface generation step.

Let us consider again the scene in Figure 1. Formulas $\gamma_1$–$\gamma_4$ shown in Table 1 are all such that $\gamma_i$ uniquely describes $b$ (i.e., $\|\gamma_i\| = \{b\}$) in model $\mathcal{S}$. Arguably, $\gamma_1$ can be easily realized as *"the small dog that sniffs a dog"*. Syntactically, $\gamma_1$ is characterized as a positive, conjunctive, existential formula (i.e., it contains no negation and uses only conjunction and existential quantification). Expressions with these characteristics are, by large, the most commonly found in corpora as those compiled in [22,9,7]. Formula $\gamma_2$, on the other hand, contains negation, disjunction and universal quantification. It could be realized as *"the small dog that only sniffs things that are not cats"* which sounds unnatural. Even a small change in the form of $\gamma_2$ makes it more palatable: rewrite it using $\exists$, $\neg$, and $\wedge$ to obtain *"the small dog that is not sniffing a cat"*. Similarly, $\gamma_3$ and $\gamma_4$ seem computationally harder to realize than $\gamma_1$: $\gamma_3$ contains an inequality (*"the dog sniffing another dog"*), while the quantified object appears in the first argument position in the binary relation in $\gamma_4$ (*"the dog that is sniffed by a small cat"*).

Summing up, we can ensure, already during the content determination phase, certain properties of the generated referring expression by paying attention to the formal language used in the representation. And we can do this, even before taking into account other fundamental linguistics aspects that will make certain realization preferable like saliency, the cognitive capacity of the hearer (can she recognize a *beagle* from another kind of dog?), etc.

As a concrete example, let $\mathcal{FO}^-$ be the fragment of $\mathcal{FO}$-formulas where the operator $\neg$ does not occur (but notice that atoms $x_i \not\approx x_j$ are permitted). By

$$\begin{aligned}
\gamma_1 &: \mathit{dog}(x) \wedge \mathit{small}(x) \wedge \exists y.(\mathit{sniffs}(x,y) \wedge \mathit{dog}(y))\\
\gamma_2 &: \mathit{dog}(x) \wedge \mathit{small}(x) \wedge \forall y.(\neg \mathit{cat}(y) \vee \neg \mathit{sniffs}(x,y))\\
\gamma_3 &: \mathit{dog}(x) \wedge \exists y.(x \not\approx y \wedge \mathit{dog}(y) \wedge \mathit{sniffs}(x,y))\\
\gamma_4 &: \mathit{dog}(x) \wedge \exists y.(\mathit{cat}(y) \wedge \mathit{small}(y) \wedge \mathit{sniffs}(y,x))
\end{aligned}$$

**Table 1.** Alternative descriptions for object $b$ in the model shown in Figure 1.

restricting content determination to $\mathcal{FO}^-$, we ensure that formulas like $\gamma_2$ will not be generated. If we ban $\not\approx$ from the language, $\gamma_3$ is precluded.

The fact that the representation language used has an impact on content determination is obvious, but it has not received the attention it deserves. Areces et al. [1] use different description logics (a family of formal languages used in knowledge representation, see [2]) to classify, and give a formal framework to previous work on GRE. Let us quickly introduce some of these languages as we will be mentioning them in future sections. Using description logics instead of $\mathcal{FO}$ fragments is just a notational issue, as most description logics can be seen as implicit fragments of $\mathcal{FO}$. For example, the language of the description logic $\mathcal{ALC}$, syntactically defined as the set of formulas,

$$\top \mid p \mid \neg\gamma \mid \gamma \wedge \gamma' \mid \exists r.\gamma$$

(where $p$ is a propositional symbol, $r$ a binary relational symbol, and $\gamma, \gamma' \in \mathcal{ALC}$) corresponds to a syntactic fragment of $\mathcal{FO}$ without $\not\approx$, as shown by the standard translation $\tau_x$:

$$\begin{aligned}
\tau_{x_i}(\top) &= \top & \tau_{x_i}(\gamma_1 \wedge \gamma_2) &= \tau_{x_i}(\gamma_1) \wedge \tau_{x_i}(\gamma_2)\\
\tau_{x_i}(p) &= p(x_i) & \tau_{x_i}(\exists r.\gamma) &= \exists x_{i+1}.(r(x_i, x_{i+1}) \wedge \tau_{x_{i+1}}(\gamma))\\
\tau_{x_i}(\neg\gamma) &= \neg\tau_{x_i}(\gamma)
\end{aligned}$$

Indeed, given a relational model $\mathcal{M}$, the extension of an $\mathcal{ALC}$ formula $\varphi$ in $\mathcal{M}$ exactly coincides with the extension of $\tau_{x_1}(\varphi)$ (see, e.g., [2]). Thanks to this result, for any formula $\varphi$ of $\mathcal{ALC}$ and its sublanguages we can define $\|\varphi\| = \|\tau_{x_1}(\varphi)\|$. Coming back to our previous example, by restricting content generation to $\mathcal{ALC}$ formulas (or equivalently, the corresponding fragment of $\mathcal{FO}$) we would avoid formulas like $\gamma_3$ (no equality) and $\gamma_4$ (quantified element appears always in second argument position).

Generation is discussed in [1] in terms of different description logics like $\mathcal{ALC}$ and $\mathcal{EL}$ ($\mathcal{ALC}$ without negation). We will extend the results in that paper, considering for instance $\mathcal{EL}^+$ ($\mathcal{ALC}$ with negation allowed only in front of unary relations) but, more generally, we take a model theoretic approach and argue that the primary question is not whether one should use one or other (description) logic for content generation, but rather which are the *semantic differences* one cares about. This determines the required logical formalism but also impacts on both the content determination and the surface realization problems. Each logical language can be seen as a compromise between expressiveness, realizability and computational complexity. The appropriate selection for a particular GRE task should depend on the actual context.

| $\mathcal{L}$ | ATOM$_L$ | ATOM$_R$ | REL$_L$ | REL$_R$ | INJ$_L$ | INJ$_R$ |
|---|---|---|---|---|---|---|
| $\mathcal{FO}$ | × | × | × | × | × | × |
| $\mathcal{FO}^-$ | × | | × | | × | |
| $\mathcal{ALC}$ | × | × | × | × | | |
| $\mathcal{EL}$ | × | | × | | | |
| $\mathcal{EL}^+$ | × | × | × | | | |

**Table 2.** $\mathcal{L}$-simulations for several logical languages $\mathcal{L}$.

### 2.2 Defining *Sameness*

Intuitively, given a logical language $\mathcal{L}$ we say that an object $u$ in a model $\mathcal{M}_1$ is similar in $\mathcal{L}$ to an object $v$ in a model $\mathcal{M}_2$ whenever all $\mathcal{L}$-formulas satisfied by $u$ are also satisfied by $v$. Formally, let $\mathcal{M}_1 = \langle \Delta_1, \|\cdot\|_1 \rangle$ and $\mathcal{M}_2 = \langle \Delta_2, \|\cdot\|_2 \rangle$ be two relational models with $u \in \Delta_1$ and $v \in \Delta_2$; we follow the terminology of [1] and say that *$u$ is $\mathcal{L}$-similar to $v$* (notation $u \overset{\mathcal{L}}{\rightsquigarrow} v$) whenever $u \in \|\gamma\|_1$ implies $v \in \|\gamma\|_2$, for every $\gamma \in \mathcal{L}$. It is easy to show that $\mathcal{L}$-similarity is reflexive for all $\mathcal{L}$, and symmetric for languages that contain negation.

Observe that $\mathcal{L}$-similarity captures the notion of 'identifiability in $\mathcal{L}$'. If we take $\mathcal{M}_1$ and $\mathcal{M}_2$ to be the same model, then an object $u$ in the model can be uniquely identified using $\mathcal{L}$ if there is no object $v$ different from $u$ such that $u \overset{\mathcal{L}}{\rightsquigarrow} v$. In other words, if there are two objects $u$ and $v$ in a model $\mathcal{M}$ such that $u \overset{\mathcal{L}}{\rightsquigarrow} v$, then the $\mathcal{L}$-GRE problem with input $\mathcal{M}$ and target $T = \{u\}$ will not succeed since for all formulas $\gamma \in L$ we have $\{u, v\} \subseteq \|\gamma\| \neq \{u\}$.

The notion of $\mathcal{L}$-similarity then, gives us a handle on the $\mathcal{L}$-GRE problem. Moreover, we can recast this definition in a structural way, so that we do not need to consider infinitely many $\mathcal{L}$-formulas to decide whether $u$ is $\mathcal{L}$-similar to $v$. We can reinterpret $\mathcal{L}$-similarity in terms of standard model-theoretic notions like isomorphisms or bisimulations which describe structural properties of the model, instead. Given two models $\langle \Delta_1, \|\cdot\|_1 \rangle$ and $\langle \Delta_2, \|\cdot\|_2 \rangle$, consider the following properties of a binary relation $\sim\, \subseteq \Delta_1 \times \Delta_2$ (cf. Convention 1):

ATOM$_L$: If $u_1{\sim}u_2$, then $u_1 \in \|p\|_1 \Rightarrow u_2 \in \|p\|_2$
ATOM$_R$: If $u_1{\sim}u_2$, then $u_2 \in \|p\|_2 \Rightarrow u_1 \in \|p\|_1$
REL$_L$ : If $u_1{\sim}u_2$ and $(u_1, v_1) \in \|r\|_1$, then $\exists v_2$ s.t. $v_1{\sim}v_2$ and $(u_2, v_2) \in \|r\|_2$
REL$_R$ : If $u_1{\sim}u_2$ and $(u_2, v_2) \in \|r\|_2$, then $\exists v_1$ s.t. $u_1{\sim}v_1$ and $(u_1, v_1) \in \|r\|_1$
INJ$_L$ : $\sim$ is an injective function (when restricted to its domain)
INJ$_R$ : $\sim^{-1}$ is an injective function (when restricted to its domain)

We will say that a non-empty binary relation $\sim$ is an *$\mathcal{L}$-simulation* when it satisfies the properties indicated in Table 2. For example, a non-empty binary relation that satisfies ATOM$_L$, and REL$_L$ is an $\mathcal{EL}$-simulation, as indicated in row 4 of Table 2. Moreover, we will say that an object *$v$ $\mathcal{L}$-simulates $u$* (notation $u \overset{\mathcal{L}}{\underline{\rightarrow}} v$) if there is a relation $\sim$ satisfying the corresponding properties such that $u \sim v$. The following is a fundamental model-theoretic result:

**Theorem 1.** *If $\mathcal{M}_1 = \langle \Delta_1, \|\cdot\|_1 \rangle$ and $\mathcal{M}_2 = \langle \Delta_2, \|\cdot\|_2 \rangle$ are finite models, $u \in \Delta_1$ and $v \in \Delta_2$, then $u \overset{\mathcal{L}}{\rightsquigarrow} v$ iff $u \overset{\mathcal{L}}{\underline{\rightarrow}} v$ (for $\mathcal{L} \in \{\mathcal{FO}, \mathcal{FO}^-, \mathcal{ALC}, \mathcal{EL}, \mathcal{EL}^+\}$).*

*Proof.* Some results are well-known: $\overset{\mathcal{FO}}{\rightarrow}$ is isomorphism on labeled graphs [11]; $\overset{\mathcal{ALC}}{\rightarrow}$ corresponds to the notion of bisimulation [3, Def. 2.16]; $\overset{\mathcal{EL}}{\rightarrow}$ is a simulation as defined in [3, Def. 2.77]. The remaining cases are simple variations of these.

Therefore, on finite models[8] simulations capture exactly the notion of similarity. The right to left implication does not hold in general on infinite models.

$\mathcal{L}$-simulations allow us to determine, in an effective way, when an object is indistinguishable from another in a given model with respect to $\mathcal{L}$.

For example, we can verify that $a \overset{\mathcal{EL}}{\rightarrow} b$ in the model of Figure 1 (the relation $\sim\, = \{(a,a),(a,b)\}$ satisfies $\text{ATOM}_L$ and $\text{REL}_L$). Using Theorem 1 we conclude that there is no $\mathcal{EL}$-description for $a$, since for any $\mathcal{EL}$-formula $\gamma$, if $a \in \|\gamma\|$, then $b \in \|\gamma\|$. Observe that $b \overset{\mathcal{EL}}{\not\rightarrow} a$, since (again applying Theorem 1), $b \in \|small(x)\|$ but $a \notin \|small(x)\|$. If one chooses a language richer than $\mathcal{EL}$, such as $\mathcal{EL}^+$, one may be able to describe $a$: take, for instance the $\mathcal{EL}^+$-formula $dog(x) \wedge \neg small(x)$.

As we will discuss in the next section, simulation gives us an efficient, computationally feasible approach to the $\mathcal{L}$-GRE problem. Algorithms to compute many kinds of $\mathcal{L}$-simulations are well known (see, [15,18,14,10]), and for many languages (e.g., $\mathcal{ALC}$, $\mathcal{ALC}$ with inverse relations, $\mathcal{EL}^+$ and $\mathcal{EL}$) they run in polynomial time (on the other hand, no polynomial algorithm for $\mathcal{FO}$- or $\mathcal{FO}^-$-simulation is known and even the exact complexity of the problem in these cases is open [13]).

## 3 GRE via Simulator Sets

In this section we will discuss how to solve the $\mathcal{L}$-GRE problem using simulation. Given a model $\mathcal{M} = \langle \Delta, \|\cdot\| \rangle$, Theorem 1 tells us that if two distinct elements $u$ and $v$ in $\Delta$ are such that $u \overset{\mathcal{L}}{\rightarrow} v$ then every $\mathcal{L}$-formula that is true at $u$ is also true at $v$. Hence there is no formula in $\mathcal{L}$ that can uniquely refer to $u$. From this perspective, knowing whether the model contains an element that is $\mathcal{L}$-similar but distinct from $u$ is equivalent to decide whether there exists an $\mathcal{L}$-RE for $u$.

Assume a fixed language $\mathcal{L}$ and a model $\mathcal{M}$. Suppose we want to refer to an element $u$ in the domain of $\mathcal{M}$. We would like to compute the *simulator set* of $u$ defined as $sim^{\mathcal{M}}_{\mathcal{L}}(u) = \{v \in \Delta \mid u \overset{\mathcal{L}}{\rightarrow} v\}$. When the model $\mathcal{M}$ is clear from the context, we just write $sim_{\mathcal{L}}$. If $sim^{\mathcal{M}}_{\mathcal{L}}(u)$ is not the singleton $\{u\}$, the $\mathcal{L}$-GRE problem with target $\{u\}$ in $\mathcal{M}$ will fail.

An algorithm is given in [14] to compute $sim_{\mathcal{EL}^+}(v)$ for each element $v$ of a given finite model $\mathcal{M} = \langle \Delta, \|\cdot\| \rangle$ in time $O(\#\Delta \times \#\|\cdot\|)$. Intuitively, this algorithm defines $S(v)$ as a set of candidates for simulating $v$ and successively refines it by removing those which fail to simulate $v$. In the end, $S(v) = sim_{\mathcal{EL}^+}(v)$. The algorithm can be adapted to compute $sim_{\mathcal{L}}$ for many other languages $\mathcal{L}$. In particular, we can use it to compute $sim_{\mathcal{EL}}$ in polynomial time which will give us the basic algorithm for establishing an upper bound to the complexity of the $\mathcal{EL}$-GRE problem –this will answer an open question of [1]. The pseudo-code is shown in Algorithm 1, which uses the following notation: $\mathcal{P}$ is a fixed set of

[8] Finiteness is not the weakest hypothesis, but it is enough for our development.

unary relation symbols, for $v \in \Delta$, let $P(v) = \{p \in \mathcal{P} \mid v \in \|p\|\}$ and let also $suc_r(v) = \{u \in \Delta \mid (v, u) \in \|r\|\}$ for $r$ a binary relation symbol.

```
Algorithm 1: Computing $\mathcal{EL}$-similarity
input  : a finite model $\mathcal{M} = \langle \Delta, \|\cdot\| \rangle$
output: $\forall v \in \Delta$, the simulator set $sim^{\mathcal{M}}_{\mathcal{EL}}(v) = S(v)$
foreach $v \in \Delta$ do
    $S(v) := \{u \in \Delta \mid P(v) \subseteq P(u)\}$
while $\exists r, u, v, w : v \in suc_r(u), w \in S(u), suc_r(w) \cap S(v) = \emptyset$ do
    $S(u) := S(u) \setminus \{w\}$
```

The algorithm is fairly straightforward. We initialize $S(v)$ with the set of all elements $u \in \Delta$ such that $P(v) \subseteq P(u)$, i.e., the set of all elements satisfying at least the same unary relations as $v$ (this guarantees that property $\text{ATOM}_L$ holds). At each step, if there are three elements $u$, $v$ and $w$ such that for some relation $r$, $(u, v) \in \|r\|$, $w \in S(u)$ (i.e., $w$ is a candidate to simulate $u$) but $suc_r(w) \cap S(v) = \emptyset$ (there is no element $w'$ such that $(w, w') \in \|r\|$ and $w' \in S(v)$) then clearly condition $\text{REL}_L$ is not satisfied under the supposition that $sim_{\mathcal{EL}} = S$. $S$ is 'too big' because $w$ cannot simulate $u$. Hence $w$ is removed from $S(u)$.

Algorithm 1 will only tell us whether an $\mathcal{EL}$-RE for an element $u$ exists (that is, whether $sim_{\mathcal{EL}}(u) = \{u\}$ or not). It does not compute an $\mathcal{EL}$-formula $\varphi$ that uniquely refers to $v$. But we can adapt it to obtain such a formula. Algorithm 1's main strategy to compute simulations is to successively refine an over-approximation of the simulator sets. The "reason" behind each refinement can be encoded using an $\mathcal{EL}$-formula. Using this insight, one can transform an algorithm that computes $\mathcal{L}$-simulator sets with a similar strategy, into one that additionally computes an $\mathcal{L}$-RE for each set.

Algorithm 2 shows a transformed version of Algorithm 1 following this principle. The idea is that each node $v \in \Delta$ is now tagged with a formula $F(v)$ of $\mathcal{EL}$. The formulas $F(v)$ are updated along the execution of the loop, whose invariant ensures that $v \in \|F(v)\|$ and $\|F(u)\| \subseteq S(u)$ hold for all $u, v \in \Delta$.

Initially $F(v)$ is the conjunction of all the unary relations that satisfy $v$ (if there is none, then $F(v) = \top$). Each time the algorithm finds elements $r, u, v, w$ such that $(u, v) \in \|r\|$, $w \in S(u)$ and $suc_r(w) \cap S(v) = \emptyset$, it updates $F(u)$ to $F(u) \wedge \exists r.F(v)$. Again this new formula $\varphi$ is in $\mathcal{EL}$ and it can be shown that $v \in \|\varphi\|$ and $w \notin \|\varphi\|$, hence witnessing that $v \overset{\mathcal{EL}}{\rightsquigarrow} w$ is false.

Algorithm 2 can be easily modified to calculate the $\mathcal{EL}^+$-RE of each simulator set $sim_{\mathcal{EL}^+}$ by adjusting the initialization: replace $\subseteq$ by $=$ in the initialization of $S(v)$ and initialize $F(v)$ as $\bigwedge \big(P(v) \cup \overline{P}(v)\big)$, where $\overline{P}(v) = \{\neg p \mid v \notin \|p\|\}$.

With a naive implementation Algorithm 2 executes in time $O(\#\Delta^3 \times \#\|\cdot\|^2)$ providing a polynomial solution to the $\mathcal{EL}$ and $\mathcal{EL}^+$-GRE problems. Algorithm 1 can be transformed to run with a lower complexity as in shown in [14]; moreover this version of the algorithm can be adapted to compute $\mathcal{EL}$- and $\mathcal{EL}^+$-RE for

**Algorithm 2**: Computing $\mathcal{EL}$-similarity and $\mathcal{EL}$-RE

```
input  : a finite model $\mathcal{M} = \langle \Delta, \| \cdot \| \rangle$
output: $\forall v \in \Delta$, a formula $F(v) \in \mathcal{EL}$, and the simulator set $S(v)$ such that
        $\|F(v)\| = S(v) = sim_{\mathcal{EL}}(v)$

foreach $v \in \Delta$ do
    $S(v) := \{u \in \Delta \mid P(v) \subseteq P(u)\}$;
    $F(v) := \bigwedge P(v)$;

while $\exists r, u, v, w : v \in suc_r(u), w \in S(u), suc_r(w) \cap S(v) = \emptyset$ do
    invariant $\forall u, v : \|F(u)\| \subseteq S(u) \wedge v \in \|F(v)\|$
    $S(u) := S(u) \setminus \{w\}$;
    if $\exists r.F(v)$ is not a conjunct of $F(u)$ then
        $F(u) := F(u) \wedge \exists r.F(v)$;
```

an arbitrary subset of the domain of $\langle \Delta, \| \cdot \| \rangle$ in $O(\#\Delta \times \#\| \cdot \|)$ steps. We shall skip the details.

**Theorem 2.** *The $\mathcal{EL}$ and $\mathcal{EL}^+$-GRE problems over $\mathcal{M} = \langle \Delta, \| \cdot \| \rangle$ have complexity $O(\#\Delta \times \#\| \cdot \|)$.*

Theorem 2 answers a question left open in [1]: the $\mathcal{EL}$-GRE problem can be solved in polynomial time. Note, however, that this result assumes a convenient representation of formulas like, for example, directed acyclic graphs, to ensure that each step of the formula construction can be done in $O(1)$. In §6 we will take a closer look at the issue and its relation to the size of the smallest $\mathcal{L}$-RE.

Algorithm 2 was obtained by adding formula annotations to a standard '$\mathcal{EL}$-simulation-minimization' algorithm. Given an $\mathcal{L}$-simulation-minimization, we can typically adapt it in an analogous way to obtain an $\mathcal{L}$-GRE algorithm. The obtained algorithm computes $\mathcal{L}$-REs for *every* element of the domain simultaneously. This will make it particularly suitable for applications with static domains requiring references to many objects. Moreover, the algorithm can be adapted to dynamic domains by using techniques used to recompute simulations (see [19]), so that only those RE that were affected by a change in the domain need to be recomputed.

We have not addressed so far other relevant issues of the GRE problem besides computational complexity. In particular, Algorithm 2 pays no attention to the use of preferences among relations when generating an RE (i.e., preferring the selection of certain attributes over others, when possible). While there is room for improvement (e.g., making a weighted choice instead of the non-deterministic choice when choosing elements in the main loop of the algorithm), support for preferences is not one of the strong points of this family of algorithms. We consider algorithms with strong support for preferences in the following section.

## 4 GRE via Building Simulated Models

Krahmer et al. [17] introduce an algorithm for content determination based on the computation of *subgraph isomorphisms*. It is heavily regulated by cost functions and is therefore apt to implement different preferences. In fact, they show that using suitable cost functions it can simulate most of the previous proposals. Their algorithm takes as input a labeled directed graph $G$ and a node $e$ and returns, if possible, a connected subgraph $H$ of $G$, containing $e$ and enough edges to distinguish $e$ from the other nodes.

In this section we will identify its underlying notion of expressiveness and will extend it to accommodate other notions. To keep the terminology of [17], in what follows we may alternatively talk of *labeled graphs* instead of relational models. The reader should observe that they are essentially the same mathematical object, but notice that in [17], propositions are encoded using looping binary relations (e.g., they write $\mathit{dog}(e,e)$ instead of $\mathit{dog}(e)$).

The main ideas of their algorithm can be intuitively summarized as follows. Given two labeled graphs $H$ and $G$, and vertices $v$ of $H$ and $w$ of $G$, we say that the pair $(v,H)$ *refers* to the pair $(w,G)$ iff $H$ is connected and $H$ can be "placed over" $G$ in such a way that: 1) $v$ is placed over $w$; 2) each node of $H$ is placed over a node of $G$ with at least the same unary predicates (but perhaps more); and 3) each edge from $H$ is placed over an edge with the same label. Furthermore, $(v,H)$ *uniquely refers* to $(w,G)$ if $(v,H)$ refers to $(w,G)$ and there is no vertex $w' \neq w$ in $G$ such that $(v,H)$ refers to $(w',G)$. The formal notion of a labeled graph being "placed over" another one is that of *subgraph isomorphism*: $H = \langle \Delta_H, \|\cdot\|_H \rangle$ can be placed over $G$ iff there is a labeled subgraph (i.e., a graph obtained from $G$ by possibly deleting certain nodes, edges, and propositions from some nodes) $G' = \langle \Delta_{G'}, \|\cdot\|_{G'} \rangle$ of $G$ such that $H$ is *isomorphic* to $G'$, which means that there is a bijection $f : \Delta_H \to \Delta_{G'}$ such that for all vertices $u, v \in \Delta_H$, $u \in \|p\|_H$ iff $f(u) \in \|p\|_{G'}$ and $(u,v) \in \|r\|_H$ iff $(f(u), f(v)) \in \|r\|_{G'}$.

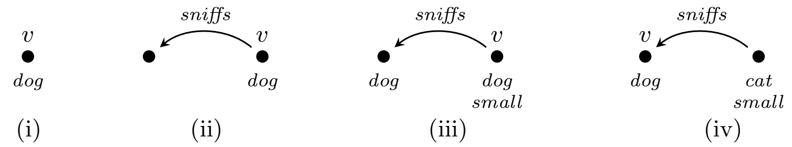

**Fig. 2.** Some connected subgraphs $(v,H)$ of scene $\mathcal{S}$ in Figure 1.

As an example, consider the relational model depicted in Figure 1 as a labeled graph $G$, and let us discuss the pairs of nodes and connected subgraphs $(v,H)$ shown in Figure 2. Clearly, (i) refers to the pair $(w,G)$ for any node $w \in \{a,b,d\}$; (ii) refers to $(w,G)$ for $w \in \{b,d\}$; and both (iii) and (iv) uniquely refer to $(b,G)$. Notice that (i)–(iv) can be respectively realized as "*a dog*", "*a dog that sniffs something*", "*a small dog that sniffs a dog*" (cf. $\gamma_1$ in Table 1) and "*the dog that is sniffed by a small cat*" (cf. $\gamma_4$ in Table 1).

It is important to emphasize that there is a substantial difference between the algorithm presented in [17] and the one we discussed in the previous sections: while the input is a labeled graph $G$ and a target node $v$, the output is, in this case (and unlike the definition of $\mathcal{L}$-GRE problem presented in §2 where the output is a *formula*), the cheapest (with respect to some, previously specified cost function) connected *subgraph* $H$ of $G$ which uniquely refers to $(v, G)$ if there is such $H$, and $\perp$ otherwise.

We will not deal with cost functions here; it is enough to know that a cost function is a monotonic function that assigns to each subgraph of a scene graph a non-negative number which expresses the goodness of a subgraph –e.g. in Figure 2, one may tune the cost function so that (iii) is cheaper than (iv), and hence (iii) will be preferred over (iv).

For reasons of space we will not introduce here the detailed algorithm proposed in [17]. Roughly, it is a straightforward branch and bound algorithm that systematically tries all relevant subgraphs $H$ of $G$ by starting with the subgraph containing only vertex $v$ and expanding it recursively by trying to add edges from $G$ that are adjacent to the subgraph $H$ constructed up to that point. In the terminology of [17] a *distractor* is a node of $G$ different from $v$ that is also referred by $H$. The algorithm ensures that a subgraph uniquely refers to the target $v$ when it has no distractors. Reached this point we have a new candidate for the solution, but there can be other cheaper solution so the search process continues until the cheapest solution is detected. Cost functions are used to guide the search process and to give preference to some solutions over others.

Here is the key link between the graph-based method of [17] and our logical-oriented perspective: on finite relational models, subgraph isomorphism corresponds to $\mathcal{FO}^-$-simulations, in the sense that given two nodes $u, v$ of $G$, there is a subgraph isomorphic to $G$ via $f$, containing $u$ and $v$, and such that $f(u) = v$ iff $u \overset{\mathcal{FO}^-}{\underline{\rightarrow}} v$. Having made explicit this notion of sameness and, with it, the logical language associated to it, we can proceed to generalize the algorithm to make it work for other languages, and to adapt it in order to output a formula instead of a graph. This is shown in Algorithms 3 and 4.

**Algorithm 3**: $\texttt{makeRE}_{\mathcal{L}}(v)$

```
input  : an implicit finite
         $G = \langle \Delta_G, \| \cdot \| \rangle$ and
         $v \in \Delta_G$
output: an $\mathcal{L}$-RE for $v$ in $G$ if
         there is one, or else $\perp$

$H := \langle \{v\}, \emptyset \rangle$;
$f := \{v \mapsto v\}$;
$H' := \texttt{find}_{\mathcal{L}}\ (v, \perp, H, f)$;
return $\texttt{buildF}_{\mathcal{L}}\ (H', v)$;
```

**Algorithm 4**: $\texttt{find}_{\mathcal{L}}(v, best, H, f)$

```
if $best \neq \perp \wedge \texttt{cost}(best) \leq \texttt{cost}(H)$ then
    return $best$
$distractors := \{n \mid n \in \Delta_G, n \neq v, v \overset{\mathcal{L}}{\underline{\rightarrow}} n\}$;
if $distractors = \emptyset$ then
    return $H$
foreach $\langle H', f' \rangle \in \texttt{extend}_{\mathcal{L}}(H, f)$ do
    $I := \texttt{find}_{\mathcal{L}}\ (v, best, H', f')$;
    if $best = \perp \vee \texttt{cost}(I) \leq \texttt{cost}(best)$ then
        $best := I$
return $best$;
```

These algorithms are parametric on $\mathcal{L}$; to make them concrete, one needs to provide appropriate versions of $\texttt{buildF}_{\mathcal{L}}$ and $\texttt{extend}_{\mathcal{L}}$. The former transforms the computed *graph* which uniquely refers to the target $v$ into an $\mathcal{L}$-RE *formula* for $v$; the latter tells us how to extend $H$ at each step of the main loop of Algorithm 4. Note that, unlike the presentation of [17], $\texttt{makeRE}_{\mathcal{L}}$ computes not only a graph $H$ but also an $\mathcal{L}$-simulation $f$. In order to make the discussion of the differences with the original algorithm simpler, we analyze next the case $\mathcal{L} = \mathcal{FO}^-$ and $\mathcal{L} = \mathcal{EL}$.

*The case of* $\mathcal{FO}^-$. From the computed cheapest isomorphic subgraph $H'$ one can easily build an $\mathcal{FO}^-$-formula that uniquely describes the target $v$, as is shown in Algorithm 5. Observe that if $\mathcal{FO}$-simulations were used instead, we would have to include also which unary and binary relations *do not hold* in $H'$.

```
Algorithm 5: buildF_{FO^-}$(H', v)$
let $H' = \langle\{a_1 \dots a_n\}, \|\cdot\|\rangle$,$v = a_1$;
$\gamma := \bigwedge_{a_i \neq a_j}(x_i \not\approx x_j) \wedge \bigwedge_{(a_i,a_j)\in\|r\|} r(x_i, x_j) \wedge \bigwedge_{a_i\in\|p\|} p(x_i)$
return $\exists x_2 \dots \exists x_n.\gamma$;
```

```
Algorithm 6: extend_{FO^-}$(H, f)$
$A := \{H{+}p(u) \mid u \in \Delta_H,$
        $u \in \|p\|_G \setminus \|p\|_H\}$;
$B := \{H{+}r(u,v) \mid u \in \Delta_H,$
$\{(u,v),(v,u)\} \cap \|r\|_G \setminus \|r\|_H \neq \emptyset\}$;
return $(A \cup B) \times \{id\}$;
```

Regarding the function which extends the given graph in all possible ways (Algorithm 6), since $H$ is a subgraph of $G$, $f$ is the trivial identity function $id(x) = x$. We will see the need for $f$ when discussing the case of less expressive logics like $\mathcal{EL}$. In $\texttt{extend}_{\mathcal{FO}^-}$ we follow the notation of [17] and write, for a relational model $G = \langle \Delta, \|\cdot\|\rangle$, $G + p(u)$ to denote the model $\langle \Delta \cup \{u\}, \|\cdot\|'\rangle$ such that $\|p\|' = \|p\| \cup \{u\}$ and $\|q\|' = \|q\|$ when $q \neq p$. Similarly, $G + r(u, v)$ denotes the model $\langle \Delta \cup \{u, v\}, \|\cdot\|'\rangle$ such that $\|r\|' = \|r\| \cup \{(u, v)\}$ and $\|q\|' = \|q\|$ when $q \neq r$. It is clear, then, that this function is returning all the *extensions* of $H$ by adding a missing attribute or relation to $H$, just like is done in the original algorithm.

*The case of* $\mathcal{EL}$. Observe that $\texttt{find}_{\mathcal{EL}}$ uses an $\mathcal{EL}$-simulation, and any $\mathcal{FO}^-$-simulation is an $\mathcal{EL}$-simulation. One could, in principle, just use $\texttt{extend}_{\mathcal{FO}^-}$ also for $\mathcal{EL}$. If we do this, the result of $\texttt{find}_{\mathcal{EL}}$ will be a subgraph $H$ of $G$ such that for every $\mathcal{EL}$-simulation $\sim$, $u \sim v$ iff $u = v$. The problem is that this subgraph $H$ may contain cycles and, as it is well known, $\mathcal{EL}$ (even $\mathcal{ALC}$) are incapable to distinguish a cycle from its *unraveling*[9]. Hence, although subgraph isomorphism get along with $\mathcal{FO}^-$, it is too strong to deal with $\mathcal{EL}$.

A well-known result establishes that every relational model $\mathcal{M}$ is equivalent, with respect to $\mathcal{EL}$-formulas,[10] to the unraveling of $\mathcal{M}$. That is, any model and its unraveling satisfy exactly the same $\mathcal{EL}$-formulas. Moreover, the unraveling of $\mathcal{M}$ is always a tree, and as we show in Algorithm 7, it is straightforward to extract a suitable $\mathcal{EL}$-formula from a tree.

Therefore, we need $\texttt{extend}_{\mathcal{EL}}$ to return all the possible "extensions" of $H$. Now "extension" does not mean to be a subgraph of the original graph $G$ anymore. We do this by either adding a new proposition or a new edge that is present in the unraveling of $G$ but not in $H$. This is shown in Algorithm 8.

[9] Informally, the unraveling of $G$, is a new graph, whose points are paths of $G$ from a given starting node. That is, transition sequences in $G$ are explicitly represented as nodes in the unraveled model. See [3] for a formal definition.

[10] Actually, the result holds even for $\mathcal{ALC}$-formulas.

**Algorithm 7**: $\texttt{buildF}_{\mathcal{EL}}(H', v)$

```
requires $H'$ to be a tree
$\gamma := \{\exists r.\texttt{buildF}_{\mathcal{EL}}(H', u) \mid (v, u) \in \|r\|\}$;
return $(\bigwedge \gamma) \wedge (\bigwedge_{v \in \|p\|} p)$;
```

**Algorithm 8**: $\texttt{extend}_{\mathcal{EL}}(H, f)$

```
$A :=$
    $\{\langle H{+}p(u), f\rangle \mid u \in \Delta_H, u \in \|p\|_G \setminus \|p\|_H\}$;
$B := \emptyset$;
foreach $u \in \Delta_G$ do
    foreach $u_H \in \Delta_H/(f(u_h), u) \in \|r\|_G$ do
        if $\forall v : (u_H, v) \in \|r\|_H \Rightarrow f(v) \neq u$
        then
            $n :=$ new node;
            $B := B \cup$
                $\{\langle H + r(u_H, n), f \cup \{n \mapsto u\}\rangle\}$;
return $A \cup B$;
```

Observe that the behavior of $\texttt{find}_{\mathcal{EL}}$ is quite sensible to the `cost` function employed. For instance, on cyclic models, a `cost` function that does not guarantee the unraveling is explored in a breadth-first way may lead to non-termination (since $\texttt{find}_{\mathcal{EL}}$ may loop exploring an infinite branch).

As a final note on complexity, although the set of $\mathcal{EL}$-distractors may be computed more efficiently than $\mathcal{FO}^-$-distractors (since $\mathcal{EL}$-distractors can be computed in polynomial time, and computing $\mathcal{FO}^-$-distractors seems to require a solution to the subgraph isomorphism problem which NP-complete), we cannot conclude that $\texttt{find}_{\mathcal{EL}}$ is more efficient than $\texttt{find}_{\mathcal{FO}^-}$ in general: the model built in the first case may be exponentially larger –it is an unraveling, after all. We will come back to this in §6.

## 5 Combining GRE Methods

An appealing feature of formulating the GRE problem modulo expressivity is that one can devise general strategies that combine $\mathcal{L}$-GRE algorithms. We illustrate this with an example.

The algorithms based on $\mathcal{L}$-simulator sets like the ones in §3 simultaneously compute referring expressions for every object in the domain, and do this for many logics in polynomial time. This is an interesting property when one anticipates the need of referring to a large number of elements. However, this family of algorithms is not as flexible in terms of implementing preferences as those we introduced in §4 –though some flexibility can be obtained by using cost functions for selecting $u$, $v$ and $w$ in the main loop of Algorithm 2 instead of the non-deterministic choices.

There is a simple way to obtain an algorithm that is a compromise between these two techniques. Let $A_1$ and $A_2$ be two procedures that solve the $\mathcal{L}$-GRE problem based on the techniques of §3 and §4, respectively. One can first compute an $\mathcal{L}$-RE for every possible object using $A_1$ and then (lazily) replace the calculated RE for $u$ with $A_2(u)$ whenever the former does not conform to some predefined criterion. This is correct but we do better, taking advantage of the equivalence classes obtained using $A_1$.

Since $A_1$ computes, for a given $\mathcal{M} = \langle \Delta, \|\cdot\| \rangle$, the set $sim(u)$ for every $u \in \Delta$, one can build in polynomial time, using the output of $A_1$, the model $\mathcal{M}_{\mathcal{L}} = \langle \{[u] \mid u \in \Delta\}, \|\cdot\|_{\mathcal{L}} \rangle$, such that: $[u] = \{v \mid u \xrightarrow{\mathcal{L}} v \text{ and } v \xrightarrow{\mathcal{L}} u\}$ and $\|r\|_{\mathcal{L}} = \{([u_1] \ldots [u_n]) \mid (u_1 \ldots u_n) \in \|r\|\}$. $\mathcal{M}_{\mathcal{L}}$ is known as *the $\mathcal{L}$-minimization of $\mathcal{M}$*. By a straightforward induction on $\gamma$ one can verify that $(u_1 \ldots u_n) \in \|\gamma\|$ iff $([u_1] \ldots [u_n]) \in \|\gamma\|_{\mathcal{L}}$ and this implies that $\gamma$ is an $\mathcal{L}$-RE for $u$ in $\mathcal{M}$ iff it is an $\mathcal{L}$-RE for $[u]$ in $\mathcal{M}_{\mathcal{L}}$.

If $\mathcal{M}$ has a large number of indistinguishable elements (using $\mathcal{L}$), then $\mathcal{M}_{\mathcal{L}}$ will be much smaller than $\mathcal{M}$. Since the computational complexity of $A_2$ depends on the size of $\mathcal{M}$, for very large scenes, one should compute $A_2([u])$ instead.

## 6 On the Size of Referring Expressions

The expressive power of a language $\mathcal{L}$ determines if there is an $\mathcal{L}$-RE for an element $u$. It also influences the *size* of the *shortest* $\mathcal{L}$-RE (when they exist). Intuitively, with more expressive power we are able to 'see' more differences and therefore have more resources at hand to build a shorter formula.

A natural question is, then, whether we can characterize the relative size of the $\mathcal{L}$-REs for a given $\mathcal{L}$. That is, if we can give (tight) upper bounds for the size of the shortest $\mathcal{L}$-REs for the elements of an arbitrary model $\mathcal{M}$, as a function of the size of $\mathcal{M}$.

For the case of one of the most expressive logics considered in this article, $\mathcal{FO}^-$, the answer follows from algorithm $\texttt{makeRE}_{\mathcal{FO}^-}$ in §4. Indeed, if an $\mathcal{FO}^-$-RE exists, it is computed by $\texttt{buildF}_{\mathcal{FO}^-}$ from a model $H$ that is not bigger than the input model. It is easy to see that this formula is linear in the size of $H$ and, therefore the size of any $\mathcal{FO}^-$-RE is $O(\#\Delta + \#\|\cdot\|)$. It is not hard to see that this upper bound holds for $\mathcal{FO}$-REs too.

One is tempted to conclude from Theorem 2 that the size of the shortest $\mathcal{EL}$-RE is $O(\#\Delta \times \#\|\cdot\|)$, but there is a pitfall. Theorem 2 assumes that formulas are represented as a DAG and it guarantees that this DAG is polynomial in the size of the input model. One can easily reconstruct (the syntax tree of) the formula from the DAG, but this, in principle, may lead to a exponential blow-up –the result will be an exponentially larger formula, but composed of only a polynomial number of different subformulas. As the following example shows, it is indeed possible to obtain an $\mathcal{EL}$-formula that is exponentially larger when expanding the DAG representation generated by Algorithm 2.

*Example 1.* Consider a language with only one binary relation $r$, and let $\mathcal{M} = \langle \Delta, \|\cdot\| \rangle$ where $\Delta = \{1, 2, \ldots, n\}$ and $(i, j) \in \|r\|$ iff $i < j$. Algorithm 2 initializes $F(j) = \top$ for all $j \in \Delta$. Suppose the following choices in the execution: For $i = 1, \ldots, n-1$, iterate $n - i$ times picking $v = w = n - i + 1$ and successively $u = n - i, \ldots, 1$. It can be shown that each time a formula $F(j)$ is updated, it changes from $\varphi$ to $\varphi \wedge \exists r.\varphi$ and hence it doubles its size. Since $F(1)$ is updated $n - 1$ many times, the size of $F(1)$ is greater than $2^n$.

The large $\mathcal{EL}$-RE of Example 1 is due to an unfortunate (non-deterministic) choice of elements. Example 2 shows that another execution leads to a quadratic RE (but notice the shortest one is linear: $(\exists r)^{(n-1)}.\top$).

*Example 2.* Suppose now that in the first $n-1$ iterations we successively choose $v = w = n - i$ and $u = v - 1$ for $i = 0 \ldots n-2$. It can be seen that for further convenient choices, $F(1)$ is of size $O(n^2)$.

But is it always possible to obtain an $\mathcal{EL}$-RE of size polynomial in the size of the input model, when we represent a formula as a string, and not as a DAG? In [12] it is shown that the answer is 'no': for $\mathcal{L} \in \{\mathcal{ALC}, \mathcal{EL}, \mathcal{EL}^{+}\}$, the lower bound for the length of the $\mathcal{L}$-RE is exponential in the size of the input model[11], and this lower bound is tight.

## 7 Conclusions

The content determination phase during the generation of referring expressions identifies which 'properties' will be used to refer to a given target object or set of objects. What is considered as a 'property' is specified in different ways by each of the many algorithms for content determination existing in the literature. In this article, we put forward that this issue can be addressed by deciding when two elements should be considered to be equal, that is, by deciding which discriminatory power we want to use. Formally, the discriminatory power we want to use in a particular case can be specified syntactically by choosing a particular formal language, or semantically, by choosing a suitable notion of simulation. It is irrelevant whether we choose first the language (and obtain the associated notion of simulation afterwards) or vice versa.

We maintain that having both at hand is extremely useful. Obviously, the formal language will come handy as representation language for the output to the content determination problem. But perhaps more importantly, once we have fixed the expressivity we want to use, we can rely on model theoretical results defining the adequate notion of sameness underlying each language, which indicates what can and cannot be said (as we discussed in §2). Moreover, we can transfer general results from the well-developed fields of computational logics and graph theory as we discuss in §3 and §4, where we generalized known algorithms into *families* of GRE algorithms for different logical languages.

An explicit notion of expressiveness also provides a cleaner interface, either between the content determination and surface realization modules or between two collaborating content determination modules. An instance of the latter was exhibited in §5.

As a future line of research, one may want to avoid sticking to a fixed $\mathcal{L}$ but instead favor an incremental approach in which features of a more expressive language $\mathcal{L}_1$ are used only when $\mathcal{L}_0$ is not enough to distinguish certain element.

[11] More precisely, there are infinite models $G_1, G_2, \ldots$ such that for every $i$, the size of $G_i$ is linear in $i$ but the size of the minimum RE for some element in $G_i$ is bounded from below by a function which is exponential on $i$.

## References

1. Areces, C., Koller, A., Striegnitz, K.: Referring expressions as formulas of description logic. In: Proc. of the 5th INLG. Salt Fork, OH, USA (2008)
2. Baader, F., McGuiness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook: Theory, implementation and applications. Cambridge University Press (2003)
3. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge University Press (2001)
4. Dale, R.: Cooking up referring expressions. In: Proc. of the 27th ACL (1989)
5. Dale, R., Haddock, N.: Generating referring expressions involving relations. In: Proc. of the 5th EACL (1991)
6. Dale, R., Reiter, E.: Computational interpretations of the Gricean maxims in the generation of referring expressions. Cognitive Science 19 (1995)
7. Dale, R., Viethen, J.: Referring expression generation through attribute-based heuristics. In: Proc. of the 12th ENLG workshop. pp. 58–65 (2009)
8. van Deemter, K.: Generating referring expressions: Boolean extensions of the incremental algorithm. Computational Linguistics 28(1), 37–52 (2002)
9. van Deemter, K., van der Sluis, I., Gatt, A.: Building a semantically transparent corpus for the generation of referring expressions. In: Proc. of the 4th INLG (2006)
10. Dovier, A., Piazza, C., Policriti, A.: An efficient algorithm for computing bisimulation equivalence. Theor. Comput. Sci 311, 221–256 (2004)
11. Ebbinghaus, H., Flum, J., Thomas, W.: Mathematical Logic. Springer (1996)
12. Figueira, S., Gorín., D.: On the size of shortest modal descriptions. In: Advances in Modal Logic. vol. 8, pp. 114–132 (2010)
13. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. Freeman (1979)
14. Henzinger, M.R., Henzinger, T.A., Kopke, P.W.: Computing simulations on finite and infinite graphs. In: Proc. of 36th Annual Symposium on Foundations of Computer Science. pp. 453–462. IEEE Computer Society Press (1995)
15. Hopcroft, J.: An $nlog(n)$ algorithm for minimizing states in a finite automaton. In Z. Kohave, editor, *Theory of Machines and Computations*, Academic Press (1971)
16. Horacek, H.: An algorithm for generating referential descriptions with flexible interfaces. In: Proc. of the 35th ACL. pp. 206–213 (1997)
17. Krahmer, E., van Erk, S., Verleg, A.: Graph-based generation of referring expressions. Computational Linguistics 29(1) (2003)
18. Paige, R., Tarjan, R.: Three partition refinement algorithms. SIAM J. Comput. 16(6), 973–989 (1987)
19. Saha, D.: An incremental bisimulation algorithm. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science, LNCS, vol. 4855, pp. 204–215. Springer Berlin / Heidelberg (2007)
20. Stone, M.: On identifying sets. In: Proc. of the 1st INLG (2000)
21. Stone, M., Webber, B.: Textual economy through close coupling of syntax and semantics. In: Proc. of the 9th INLG workshop. pp. 178–187 (1998)
22. Viethen, J., Dale, R.: Algorithms for generating referring expressions: Do they do what people do? In: Proc. of the 4th INLG (2006)