International Conference on Rewriting Techniques and Applications 2010 (Edinburgh), pp. 177-192
http://rewriting.loria.fr/rta/

COMPUTING RELATIVE NORMAL FORMS IN REGULAR TREE LANGUAGES

ALEXANDER KOLLER [1] AND STEFAN THATER [1]

[1] Saarland University
Saarbrücken, Germany
E-mail address: koller@mmci.uni-saatland.de
E-mail address: stth@coli.uni-saarland.de

ABSTRACT. We solve the problem of computing, out of a regular language L of trees and a rewriting system R, a regular tree automaton describing the set $L' \subseteq L$ of trees which cannot be R-rewritten into a tree in L. We call the elements of L' the relative normal forms of L. We apply our algorithm to the problem of computing weakest readings of sentences in computational linguistics, by approximating logical entailment with a rewriting system, and give the first efficient and practically useful algorithm for this problem. This problem has been open for 25 years in computational linguistics.

1. Introduction

One key task in computational linguistics is to represent the meaning of a natural language sentence using some formal representation *(reading)*, and to model inference on the level of natural language [1] as inference on the corresponding meaning representations. The classical approach [2] uses logical languages, such as first or higher order predicate logic, to represent sentence meanings. But when the sentence is *ambiguous*, it is often infeasible to explicitly enumerate all the different readings: The number of readings is worst-case exponential in the length of the sentence, and it is not uncommon for a sentence to have millions of readings if they contain a number of ambiguous constituents.

The standard technique to address this issue is *underspecification* [3, 4, 5, 6]: all readings of an ambiguous sentence are represented by a single, compact *underspecified representation (USR)*, such as a dominance graph [7]. Individual readings can be enumerated from an USR if necessary, but this step is postponed for as long as possible. This offers a partial solution to the problem of managing ambiguity. However, it is much less clear how to *disambiguate* a sentence, i.e. to determine the reading that the speaker actually intended in the given context.

In the absence of convincing disambiguation techniques, it has been proposed to work with the *weakest readings* of a sentence in practical applications [8]: If the readings are a set of formulas (say, of predicate logic), the weakest readings are those readings that are minimal with respect to logical entailment. From an application perspective, the weakest readings capture the "safe" information that is common to all possible readings; they are also linguistically interesting [9]. Because there are so many readings, it is infeasible to compute all readings and test all pairs for entailment using a theorem prover. However, although the problem has been open for over 25 years [9, 10], the best

Key words and phrases: normal forms, tree automata, incomplete inference, computational linguistics.

Proceedings of the 21st International Conference on Rewriting Techniques and Applications, Edinburgh, July, 2010
Editors: Christopher Lynch
LIPIcs - Leibniz International Proceedings in Informatics. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.RTA.2010.177

known algorithm [11] is still quadratic in the number of readings and therefore too slow for practical use.

In this paper, we solve this problem for a sound but incomplete approximation of entailment by means of a rewrite system. Technically, we consider the problem of computing, from a regular tree language L of trees and a rewrite system R, a regular tree automaton representing the subset $L' \subseteq L$ consisting of all trees that cannot be R-rewritten into a tree in L. We call the elements of L' the *relative normal forms* of L with respect to R. To do this, we first represent the one-step rewriting relation of R in terms of a linear *context tree transducer*, which extends ordinary tree transducers by rewriting an entire context in each derivation step instead of a single symbol. We then compute the pre-image of L under this transducer, and intersect L with the complement of the pre-image. We show that an automaton accepting the pre-image can be computed in linear time if L is represented by a deterministic automaton. For a certain special case, which holds in our application, we show that we can even obtain a *deterministic* automaton for the pre-image in linear time. Altogether, we obtain an algorithm for computing weakest readings that is quadratic in the size of the tree automaton describing L, instead of quadratic in the size of L.

Despite the incompleteness, the approximation of entailment as rewriting is sufficient in our application: For one specific rewrite system, our algorithm computes a mean of 4.5 weakest readings per sentence in a text corpus, down from about three million readings that the sentences have on average. It takes 20 ms per sentence on average to do this. Thus, we see our algorithm as a practical solution to the problem of computing weakest readings in computational linguistics. On the other hand, our algorithm handles arbitrary linear rewriting systems and is therefore much more generally applicable. For instance, our earlier work on redundancy elimination [12], which was based on tree automata intersection as well, falls out as a special case; and we anticipate that further approximative inference techniques for natural language will be developed based on this paper in the future.

Plan of the paper. In Sect. 2, we define the problem and review dominance graphs and tree automata. We then define context tree transducers, use them to compute the pre-image of L under R, and analyze the complexity of the algorithm in Sect. 3. We go through an example from the application in Sect. 4, and conclude in Sect. 5.

2. Definitions

We start by reviewing some dominance graph theory and defining the problem we want to solve.

2.1. Dominance graphs

Semantic ambiguity, which is present when a natural-language sentence can have more than one possible meaning, is a serious problem in natural language processing with large-scale grammars. For instance, the mean number of possible meaning representations per sentence in the Rondane text corpus [13] is about $5 \cdot 10^9$. It is obviously impractical to enumerate all of these meaning representations. Instead, computational linguists typically use *underspecification* approaches, in which a single compact *description* of the possible meanings is computed instead of all possible semantic readings.

One formal approach to underspecification, which we use in this paper, is that of using *dominance graphs* [7]. Dominance graphs assume that the individual semantic representations of the sentence – say, formulas of predicate logic – are represented as trees, and then describe sets of trees

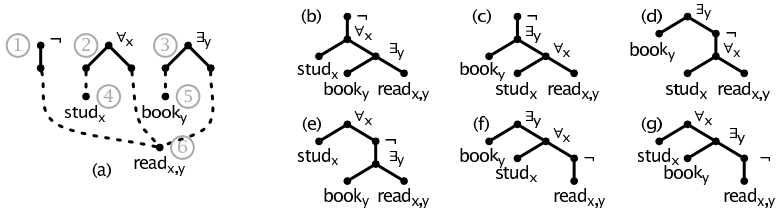

Figure 1: A dominance graph that represents the six readings of the sentence *every student did not read a book* (a) and its six configurations (b – g).

by specifying parent and ancestor relationships between nodes. They are equivalent to leaf-labeled normal dominance constraints [4].

We assume finite ranked signatures $\Sigma, \Sigma', \ldots$ of tree constructors f with arities $\mathsf{ar}(f)$. We define a *(finite constructor) tree* t over Σ to be a function $t : D \to \Sigma$, where D is a tree domain (i.e., a finite subset of $\mathbb{N}^*$ that is closed under prefix and left sibling), such that every node $u \in D$ has $\mathsf{ar}(t(u))$ many children. Alternatively, we can see each tree as a ground term over Σ. We write T_Σ for the set of trees over Σ.

Definition 2.1. A *(compact) labeled dominance graph* over a ranked signature Σ is a quadruple $G = (V, E \uplus D, L, <)$, where $(V, E \uplus D)$ is a directed graph, $L : V \rightsquigarrow \Sigma$ is a partial *(node) labeling function* and $< \subseteq V \times V$ a strict linear order on V, such that

(1) the graph (V, E) defines a collection of node disjoint trees of height 0 or 1 (we call the edges in E *tree edges*, the trees *fragments*, the roots of the fragments *roots* and all other nodes *holes*);
(2) if $(v, v') \in D$, then v is a hole and v' is a root in G (we call the edges in D *dominance edges*);
(3) the labeling function L assigns a node v a label with arity n iff v is a root with n outgoing tree edges (we write $f|_n$ to indicate that f has arity n);
(4) every hole has at least one outgoing dominance edge.

We write W_G for the roots and L_G for the labeling function of G, and we will say that v is a *hole of* u if $(u, v) \in E$. We will typically just say *dominance graph* for "compact labeled dominance graph".

Dominance graphs can be seen as descriptions of sets of trees, which can be obtained from the graph by "plugging" roots into holes so that dominance edges are realized as dominance. We call these trees the *configurations* of the graph. An example graph (for the sentence "every student did not read a book") and its six configurations are shown in Fig. 1, where we draw tree edges as solid lines and dominance edges as dotted lines, directed from top to bottom. The signature includes the symbols $\neg|_1$, $\forall_x|_2$, and $\mathsf{stud}_x|_0$; we read the trees over this signature as simplified formulas of predicate logic, taking $\forall_x(P, Q)$ to abbreviate $\forall x(P \to Q)$ and $\exists_x(P, Q)$ for $\exists x(P \wedge Q)$. Atomic formulas such as $stud(x)$ are abbreviated by single function symbols such as $stud_x$ of arity 0. Now the six configurations, (b) – (g), are the six trees whose nodes are the (labeled) roots of the graph, such that all dominance edges in the graph are realized as reachability in the tree.

Formally, we define a *plugging* of a dominance graph $G = (V, E \uplus D, L, <)$ to be an injective partial function $p : V \rightsquigarrow V$ mapping each hole to a root. We can apply a plugging to a dominance graph to obtain a directed graph $p(G) = (V', E')$ such that $V' = W_G$ and $E' = \{(v, p(v')) \mid (v, v') \in E\}$. We call $p(G)$ an *unlabeled configuration* of G iff (i) $p(G)$ is a tree, and (ii) if $(v, v') \in D$, then $p(v)$ dominates v' in $p(G)$, i.e., there is a directed path from $p(v)$ to v' in the tree $p(G)$. By taking W_G as a ranked signature (with $\mathsf{ar}(u) = \mathsf{ar}(L_G(u))$) and ordering the children of each node according to $<$, we can read $p(G)$ as a finite constructor tree $t \in T_{W_G}$. An unlabeled configuration t can be mapped to

a finite constructor tree $L_G(t) \in T_\Sigma$ – called a *labeled configuration* – by labeling each node $u \in V'$ with $L_G(u)$. The configurations in Fig. 1 are all labeled. We say that a graph is *configurable* if it has a (labelled or unlabelled) configuration.

Throughout this paper, we restrict ourselves to *hypernormally connected* dominance graphs. We say that G is *hypernormally connected (hnc)* iff each pair of nodes is connected by a simple *hypernormal path*. A hypernormal path [7] in a dominance graph G is a path in the undirected version of G that does not use two dominance edges that are incident to the same hole. Hnc graphs have a number of desirable properties. For instance, the problem of deciding whether a dominance graph has a configuration is NP-complete in general, but polynomial if the graph is hnc. Furthermore, hnc dominance graphs can be translated into equivalent tree automata (see below). Note that virtually all dominance graphs that are used in linguistics applications are hnc [14].

2.2. Weakest readings

In order to perform inferences on the semantic representations for a sentence, it is desirable to identify the "correct" semantic representation from among all the (many) possibilities. Unfortunately, there are no satisfying models that would allow this. One alternative that has been proposed as a workaround is to compute the *weakest readings* – that is, the least informative semantic representations [8]. This idea exploits the fact that a set of predicate logic formulas is partially ordered with respect to logical entailment; the weakest readings are then the (configurations representing the) minimal elements of this order. In Fig. 1, (f) entails (g), (b) entails (c), and so on; (d) and (g) are incomparable, and indeed, (d) and (g) are the weakest readings of the dominance graph in Fig. 1a. The problem that motivates this paper is how to efficiently compute the weakest readings of a dominance graph.

A brute-force approach to this problem would be to compute all labeled configurations of a dominance graph, and then to run a theorem prover for each pair of configurations to establish the entailment order. Although this is clearly impractically slow when real-world sentences have an average of several billions of readings, the best known algorithm [11] is essentially just an optimization of this approach, and in particular is quadratic in the number of configurations.

Here we take a different approach. We will work with a sound but incomplete approximation of entailment using a rewriting system. Notice that the entailment between (f) and (g) can be explained by the fact that (f) can be rewritten into (g) by applying the rewrite rule

$$\exists_y(P, \forall_x(Q,R)) \to \forall_x(Q, \exists_y(P,R))^1 \tag{2.1}$$

if x does not occur in P. In positive contexts, the right-hand side of this rule is always entailed by the left-hand side; in this sense, the rule is sound. Now (g) can be recognized as a weakest reading because it cannot be rewritten into another tree which is also a configuration of the dominance graph.

In order to obtain a sound model of first-order entailment, we must make the rewriting system sensitive to the logical polarity of the subformula at which we apply a rewrite rule: If we were to apply (2.1) to the configuration (c), we would rewrite it into (b), which is logically *stronger* than (c). That is, we must restrict (2.1) such that it can only be used for subformulas that occur in a logically positive context.

[1] $\forall_x, \exists_y \in \Sigma$ are uninterpreted binary constructors. To ensure finiteness of the rewrite systems we use, we assume there is only a finite set of variables $x, y, \ldots$ in the language of semantic representations. P, Q, R are ordinary variables of the rewrite system.

More generally, we assume a finite alphabet Ann of *annotations*; we want to assign a single annotation to every node of a tree in T_Σ. We assign a *starting annotation* $a_0 \in \mathsf{Ann}$ to the root of each tree, and use an *annotator function* $\mathsf{ann} : \mathsf{Ann} \times \Sigma \times \mathbb{N} \to \mathsf{Ann}$ to compute the annotations for the other nodes: If some node u is annotated with a and has label $f \in \Sigma$, then we annotate the i-th child of u with $\mathsf{ann}(a, f, i)$ for all $1 \leq i \leq \mathsf{ar}(f)$. We then define an *annotated rewriting system* R as a finite set of pairs $a : r$, where $a \in \mathsf{Ann}$ and r is a rewrite rule over Σ. A tree $t \in \Sigma$ can be rewritten into a tree t', $t \to_R t'$, if there is a rule $a : r$ and a node u in t with annotation a such that t rewrites into t' by applying r at node u. We write $\to_R^*$ for the reflexive, transitive closure of $\to_R$.

Using this terminology, we can capture logical polarities by using $\mathsf{Ann} = \{+, -\}$, $a_0 = +$, and ann such that $\mathsf{ann}(+, \forall_x, 1) = -$, $\mathsf{ann}(+, \forall_x, 2) = +$, and so on. We can then rephrase (2.1) more precisely as follows:

$$+ : \exists_y(P, \forall_x(Q, R)) \to \forall_x(Q, \exists_y(P, R)) \tag{2.2}$$

This rule will still rewrite (f) into (g) because it is applied at the root (with annotation $+$), but it will not rewrite (c) into (b), because the redex has annotation $-$. We will extend (2.2) to a full rewrite system, which correctly characterizes (d) and (g) as the only two weakest readings, in Section 4.

Now observe that if we have an annotated rewriting system in which every rule makes the formula logically weaker, then all weakest readings will have the property that it is not possible to rewrite them into some other configuration. More generally, we will say that all weakest readings are in *relative normal form* with respect to the set of all configurations.

Definition 2.2. Let L be a set of trees over some signature Σ, and let R be an (annotated) rewrite system over Σ. We say that a tree $t \in L$ is in *relative normal form* with respect to R iff there is no tree $t' \in L$ such that $t \to_R t'$. We write $\mathrm{RNF}_R(L)$ for the relative normal forms in L with respect to R.

In the example, (d) and (g) are in relative normal form because they are in normal form, i.e. they cannot be rewritten at all. However, in general a tree may be in relative normal form without being in normal form, if all possible results of rewrites are not in L. For example, consider the set $L = \{f(f(g(h(a)))), f(g(f(h(a)))), f(f(h(g(a))))\}$ and a rewrite system R that consists of the single rule $f(g(x)) \to g(f(x))$. The tree $f(f(g(h(a))))$ rewrites to $f(g(f(h(a)))) \in L$, and is therefore not in relative normal form. However, while we could further rewrite this tree into $g(f(f(h(a))))$, the result is no longer in L, so $f(g(f(h(a))))$ is in relative normal form. The tree $f(f(h(g(a))))$ does not contain a redex in the first place, and is therefore also in relative normal form.

2.3. Dominance graphs as tree automata

The problem that we solve in this paper is to find an efficient algorithm for computing the relative normal forms in regular tree languages with respect to an annotated rewriting system. This solves the problem of computing the weakest readings of a dominance graph because the configuration sets of dominance graphs are regular tree languages, and it is known how to compute tree automata for accepting them. We will now recall some definitions regarding tree automata and transducers, and then sketch the translation of dominance graphs as tree automata.

Let Σ be a finite ranked signature as above, and let $\mathscr{X}$ be a finite set of (variable) symbols. We write $T_\Sigma(\mathscr{X})$ for $T_{\Sigma \cup \{a|_0 \mid a \in \mathscr{X}\}}$. If $\mathscr{X}_m$ is a set of m variables, we write $\mathrm{Con}^{(m)}(\Sigma)$ for the *contexts* with m holes, i.e. those trees in $T_\Sigma(\mathscr{X}_m)$ in which each element of $\mathscr{X}_m$ occurs exactly once. If $C \in \mathrm{Con}^{(m)}(\Sigma)$, then $C[t_1, \ldots, t_m] = C[t_1/x_1, \ldots, t_m/x_m]$, where $x_1, \ldots, x_m$ are the variables from left to right.

Definition 2.3. A *top-down tree transducer* from Σ to Δ is a 5-tuple $M = (Q, \Sigma, \Delta, q_0, \delta)$, where Q is a finite set of states, Σ and Δ are ranked signatures, and $q_0 \in Q$ is the initial state. The rules in δ are of

$$q_{\{1,2,3,4,5,6\}}(\neg(x_1)) \to \neg(q_{\{2,3,4,5,6\}}(x_1))$$
$$q_{\{1,2,3,4,5,6\}}(\forall_x(x_1,x_2)) \to \forall_x(q_{\{4\}}(x_1), q_{\{1,3,5,6\}}(x_2))$$
$$q_{\{1,2,3,4,5,6\}}(\exists_y(x_1,x_2)) \to \exists_y(q_{\{5\}}(x_1), q_{\{1,2,4,6\}}(x_2))$$
$$q_{\{3,5,6\}}(\exists_y(x_1,x_2)) \to \exists_y(q_{\{5\}}(x_1), q_{\{6\}}(x_2))$$
$$q_{\{2,3,4,5,6\}}(\forall_x(x_1,x_2)) \to \forall_x(q_{\{4\}}(x_1), q_{\{3,5,6\}}(x_2))$$
$$q_{\{2,3,4,5,6\}}(\exists_y(x_1,x_2)) \to \exists_y(q_{\{5\}}(x_1), q_{\{2,4,6\}}(x_2))$$
$$q_{\{1,3,5,6\}}(\exists_y(x_1,x_2)) \to \exists_y(q_{\{5\}}(x_1), q_{\{1,6\}}(x_2))$$
$$q_{\{1,3,5,6\}}(\neg(x_1)) \to \neg(q_{\{3,5,6\}}(x_1))$$
$$q_{\{1,2,4,6\}}(\forall_x(x_1,x_2)) \to \forall_x(q_{\{4\}}(x_1), q_{\{1,6\}(x_2)})$$
$$q_{\{1,2,4,6\}}(\neg(x_1)) \to \neg(q_{\{2,4,6\}}(x_1))$$
$$q_{\{2,4,6\}}(\forall_x(x_1,x_2)) \to \forall_x(q_{\{4\}}(x_1), q_{\{6\}}(x_2))$$
$$q_{\{1,6\}}(\neg(x_1)) \to \neg(q_{\{6\}}(x_1))$$
$$q_{\{4\}}(\mathsf{stud_x}) \to \mathsf{stud_x}$$
$$q_{\{5\}}(\mathsf{book_y}) \to \mathsf{book_y}$$
$$q_{\{6\}}(\mathsf{read_{x,y}}) \to \mathsf{read_{x,y}}$$

Figure 2: A tree automaton accepting the labeled configurations of the dominance graph in Fig. 1(a).

the form $q(f(x_1,\ldots,x_n)) \to C[q_1(x_{i_1}),\ldots,q_m(x_{i_m})]$, where $f \in \Sigma$, $q,q_1,\ldots,q_m \in Q$, $C \in \mathrm{Con}^{(m)}(\Delta)$, and $x_{i_k} \in \{x_1,\ldots,x_n\}$ for all k.

If t is a tree in $T_{\Sigma\cup\Delta\cup Q}$, then we say that M derives t' in one step from t, $t \to_M t'$, if there are a context C', trees $t_1,\ldots,t_n$ and a transition rule $q(f(x_1,\ldots,x_n)) \to C[q_1(x_{i_1}),\ldots,q_m(x_{i_m})]$ such that $t = C'[q(f(t_1,\ldots,t_n))]$ and $t' = C'[C[q_1(t_{i_1}),\ldots,q_m(t_{i_m})]]$. The *derivation relation* $\to^*$ of M is the reflexive, transitive closure of $\to$. The *translation relation* τ_M of M is

$$\tau_M = \{(t,t') \mid t \in T_\Sigma \text{ and } t' \in T_\Delta \text{ and } q_0(t) \to^* t'\}.$$

A tree transducer is called *linear* if no variable occurs twice, and *non-deleting* if every variable occurs at least once on the right-hand side of each rule. It is called *deterministic* if for every $q \in Q$ and $f \in \Sigma$, there is at most one rule whose left-hand side is $q(f(x_1,\ldots,x_n))$.

A *top-down tree automaton* over Σ is a top-down transducer $A = (Q,\Sigma,\Sigma,q_0,\delta)$ such that every rule in δ is of the form $q(f(x_1,\ldots,x_n)) \to f(q_1(x_1),\ldots,q_n(x_n))$. We write $\mathscr{L}(A) = \{t \mid (t,t) \in \tau_A\}$ for the *language* accepted by A.

A *bottom-up tree automaton* is a 4-tuple $A = (Q,\Sigma,Q_F,\delta)$ in which $Q_F \subseteq Q$ is the set of *final states* and the transition rules in δ are of the form $f(q_1(x_1),\ldots,q_n(x_n)) \to q(f(x_1,\ldots,x_n))$. Derivations and languages are defined in analogy to the top-down case, see [15] for details. A bottom-up automaton is called deterministic if for every $f \in \Sigma$ and $q_1,\ldots,q_n \in Q$ there is at most one rule whose left-hand side is $f(q_1(x_1),\ldots,q_n(x_n))$.

For every top-down automaton A, there is a bottom-up automaton A' with $\mathscr{L}(A) = \mathscr{L}(A')$. For every bottom-up automaton A, there is a deterministic bottom-up automaton A' with $\mathscr{L}(A) = \mathscr{L}(A')$.

Now any hnc dominance graph G can be translated into a top-down tree automaton A_G that accepts the language of all labeled configurations, and into a deterministic top-down tree automaton A_G^u that accepts the language of all unlabeled configurations of G [12]. The states of these automata correspond to hnc subgraphs of G, and the transition rules encode decompositions of this subgraph into smaller subgraphs by removing a *free* root and its holes. A root u in a configurable graph G is called free iff G has a configuration whose root is u; it can be tested in linear time whether a root in a configurable hnc graph is free [16].[2]

The tree automaton we obtain for the labeled configurations of the graph in Fig. 1 (a) is shown in Fig. 2. The first rule states that we can choose $\neg$ as the root of a configuration, and obtain the subgraph $\{2,3,4,5,6\}$ by removing it. We can then choose $\forall_x$ as the root in this subgraph, splitting it into two weakly connected components, $\{4\}$ and $\{3,5,6\}$, and so on. This automaton

[2]The algorithm in [16] is defined in terms of *solved forms*. Our definition is equivalent for hnc dominance graphs.

accepts exactly the six labeled configurations of the original dominance graph. In practice, the tree automata computed for dominance graphs remain small; for instance, the graphs obtained for the sentences in the Rondane treebank [13] contain on average 14 roots and have $5 \cdot 10^9$ configurations, whereas the automata have 320 rules and can be computed in 20 ms on average [12].

3. Computing relative normal forms

We will now show how relative normal forms of regular tree languages with respect to a linear rewriting system can be computed. We will first sketch the basic idea and show an (inefficient) solution based on pre-images of regular tree languages under regular tree translations. We will then introduce *context tree transducers*, which allow us to use linear transducers and obtain an efficient algorithm.

Throughout, we limit ourselves to linear rewriting systems, which are sufficient for our application.

3.1. Relative normal forms as non-pre-images

As we defined above, a tree t in some set L of trees is in relative normal form iff there is no other tree t' in L into which t can be rewritten in one step. This can have two possible reasons: Either t is in normal form, i.e. there is no rewrite step that can be applied to it at all; or t can be rewritten, but no possible result of these rewrite steps is in L.

The key idea in this paper is to model the one-step rewriting relation of a rewriting system R with a top-down tree transducer M_R, such that $t \to_R t'$ iff $(t,t') \in \tau_R$. Given such a transducer, we can then determine the relative normal forms as those trees that cannot be rewritten with the transducer.

Lemma 3.1. *Let L be a set of trees, let R be a rewriting system, and let M be a transducer such that $t \to_R t'$ iff $(t,t') \in \tau_M$. Then*

$$\mathrm{RNF}_R(L) = L \cap \overline{\tau_M^{-1}(L)}.$$

For the case where L is a regular language of trees, the intersection can be computed efficiently using a construction on the tree automata. Complements of regular tree languages can also be computed on the automata themselves, although this requires computing the determinization of the tree automataton if it is nondeterministic, which may take exponential time. The question is how we can encode one-step rewriting in a tree transducer, and how we can compute the pre-image of L under τ_M.

For the first question, we can directly build a top-down transducer to encode the one-step rewriting relation. The transducer has two states, q and $\overline{q}$. The state q indicates that the transducer will apply a rewrite rule at some node below the current node; it is the start state. The state $\overline{q}$ indicates that the transducer will not apply any rewrite rule at the nodes below the current node; we require that all leaves of the tree are read in this state.

The transducer has two types of transitions. First, when the transducer is in state $\overline{q}$, it must copy the current symbol into the output tree; it may also choose to do this in state q:

$$\begin{array}{lcll} \overline{q}(f(x_1,\ldots,x_n)) & \to & f(\overline{q}(x_1),\ldots,\overline{q}(x_n)) & \text{for all } f \in \Sigma \\ q(f(x_1,\ldots,x_n)) & \to & f(\overline{q}(x_1),\ldots,q(x_i),\ldots,\overline{q}(x_n)) & \text{for some } 1 \le i \le n \text{ and all } f \in \Sigma \end{array} \tag{3.1}$$

In addition, in state q, M may choose to apply a rewrite rule and switch to state $\bar{q}$. Let's say we have a linear rewrite rule $f(g(x_1,x_2),x_3) \to g(x_1, f(x_2,x_3))$. We can represent an application of this rule with the following transition rules:

$$\begin{array}{lcl} q(f(x,y)) & \to & g(\bar{q}_{g,1}(x), f(\bar{q}_{g,2}(x), \bar{q}(y))) \\ \bar{q}_{g,1}(g(x,y)) & \to & \bar{q}(x) \\ \bar{q}_{g,2}(g(x,y)) & \to & \bar{q}(y), \end{array} \tag{3.2}$$

where crucially there are no other transitions for $\bar{q}_{g,1}$ and $\bar{q}_{g,2}$ than these, i.e. by using these states we simultaneously enforce the left-hand subtree below the f node to have root label g, and copy its two subtrees into the x_1 and x_2 positions of the rewriting rule.

In other words, it is possible to capture the one-step rewriting relation as the translation relation of a top-down tree transducer. But now consider how to compute $\tau_M^{-1}(L)$ for a regular tree language L. One possible idea is to consider a top-down tree transducer M_L such that $\tau_{M_L} = \{(t,t) \mid t \in L\}$; it is clear that such a transducer exists. We can then concatenate M and M_L; $\tau_M^{-1}(L)$ is the domain of $M \circ M_L$.

For deterministic tree transducers, it is known that the domain of a tree transducer is a regular tree language, and how to compute it ([17], Theorem 4.1; [18], p. 693). However, these algorithms are only applicable to *deterministic* transducers, and the transducers defined above are not deterministic (in state q, they may choose to either copy or rewrite). Furthermore, even if the transducers could be made deterministic, they compute regular tree automata of exponential size if the transducer is not linear. Indeed, the transducers shown above are not linear, because the first rule for the rewriting case duplicates x. This makes the use of these algorithms unattractive in our case.

3.2. Context tree transducers

However, although the above transducers can have non-linear rules, the extent of the non-linearity is limited: Every time a non-linear rule is applied, some other rules must be applied which will delete most of the copied trees, and retain only disjoint parts of them. This means that the machinery which the domain automaton construction uses to accommodate non-linear transducers is not necessary in our setting.

Consider the transition rules in (3.2). The only reason why we need to copy the g subtree twice is that we were not able to specify that the transducer should read the context $f(g(x_1,x_2),x_3)$ in the input tree directly. If we had a way to directly use this context on the left-hand side, each of x_1, x_2, and x_3 could appear only once on the right-hand side, and thus we could get away with a linear transducer. We will now extend the definition of top-down transducers in such a way that they can accept contexts on the left-hand side.

Definition 3.2. A *(top-down) context tree transducer* from Σ to Δ is a 5-tuple $M = (Q, \Sigma, \Delta, q_0, \delta)$. δ is a finite set of transition rules of the form $q(C[x_1,\ldots,x_n]) \to D[q_1(x_{i_1}),\ldots,q_m(x_{i_m})]$, where $C \in \mathrm{Con}^{(n)}(\Sigma)$, $D \in \mathrm{Con}^{(m)}(\Delta)$, $q, q_1, \ldots, q_m \in Q$, and $x_{i_k} \in \{x_1,\ldots,x_n\}$ for all k.

If t is a tree in $T_{\Sigma\cup\Delta\cup Q}$, then we say that M derives t' in one step from t, $t \to t'$, if there is a context C', trees $t_1,\ldots,t_n$, and a transition rule $q(C[x_1,\ldots,x_n]) \to_M D[q_1(x_{i_1}),\ldots,q_m(x_{i_m})]$ such that $t = C'[q(C[t_1,\ldots,t_n])]$ and $t' = C'[D[q_1(t_{i_1}),\ldots,q_m(t_{i_m})]]$. The *derivation relation* $\to_M^*$ is the reflexive, transitive closure of $\to_M$. The *translation relation* τ_M of M is

$$\tau_M = \{(t,t') \mid t \in T_\Sigma \text{ and } t' \in T_\Delta \text{ and } q_0(t) \to^* t'\}.$$

A context tree transducer is called *linear* if no variable occurs twice, and *non-deleting* if every variable occurs at least once on the right-hand side of each rule.

A *context tree automaton* is a context tree transducer with $\Sigma = \Delta$ and transition rules of the form $q(C[x_1,\ldots,x_n]) \to C[q_1(x_1),\ldots,q_n(x_n)]$. We take the *language* $\mathscr{L}(A)$ of a context tree automaton A to be the domain of τ_A.

Every top-down transducer is trivially also a top-down context transducer. Conversely, not every translation relation of a context transducer can be represented as the translation relation of an ordinary top-down transducer. For instance, the relation $\{(f(a,b),b)\}$ is the translation relation of a context transducer with rules $q(f(a,x)) \to q'(x)$ and $q'(b) \to b$. However, a top-down transducer must either output b when it reads the f or when it reads the b; either way, it must also accept an input tree $f(b,b)$. Every translation relation of a context transducer can also be computed by the "transformation language" of [19]. More specifically, context tree transducers are equivalent to extended left-hand side tree transducers (xTs) [20]. It is clear that an xT can encode a context transducer by specifying all constructors in the context in its tree pattern. Furthermore, an xT can be simulated in a context transducer by having a transition rule for every context C that satisfies the tree pattern on the left-hand side of each xT rule.

On the other hand, context *automata* are equivalent to ordinary tree automata. This can be shown as for regular tree grammars.

3.3. Rewriting with context tree transducers

Given a linear rewriting system R, it is now straightforward to produce a context tree transducer M_R whose translation relation is the one-step rewriting relation of R. First, M_R uses the rules in (3.1) to be able to copy parts of the input tree to the output unchanged. Second, for each rewrite rule $C[x_1,\ldots,x_n] \to C'[x_{i_1},\ldots,x_{i_n}]$, M_R contains a transition rule as follows:

$$q(C[x_1,\ldots,x_n]) \to C'[\overline{q}(x_{i_1}),\ldots,\overline{q}(x_{i_n})]. \tag{3.3}$$

Unlike the transducer in Section 3.1, this transducer is now linear. We will exploit this in Section 3.4 to obtain a more efficient algorithm for computing a pre-image.

But before we do this, let us extend the construction of context tree transducers for rewriting systems to annotated rewriting systems. Let's say we have an annotation alphabet Ann, an annotator function ann, and a linear annotated rewriting system R over Σ and Ann. We can obtain a context tree transducer M_R for the one-step rewriting relation of R by keeping track of the current annotation of nodes in the input tree in the state. In particular, we split the state q into states $q^{a_1},\ldots,q^{a_n}$ where $\mathsf{Ann} = \{a_1,\ldots,a_n\}$. We retain a single state $\overline{q}$, as no further rewriting can take place in this state, and the annotation is therefore irrelevant.

The initial state of M_R is q^{a_0}, where a_0 is the starting annotation. We then have the following versions of the transition rules in (3.1); these rules copy symbols to the output tree and keep track of the current annotation.

$$\begin{array}{lcll} \overline{q}(f(x_1,\ldots,x_n)) & \to & f(\overline{q}(x_1),\ldots,\overline{q}(x_n)) & \text{for all } f \in \Sigma \\ q^a(f(x_1,\ldots,x_n)) & \to & f(\overline{q}(x_1),\ldots,q^{\mathsf{ann}(a,f,i)}(x_i),\ldots,\overline{q}(x_n)) & \text{for some } 1 \le i \le n \text{ and all } f \in \Sigma \end{array} \tag{3.4}$$

In addition, M_R contains the following version of the transition rules in (3.3). It applies the rewriting rule $a : C[x_1,\ldots,x_n] \to C'[x_{i_1},\ldots,x_{i_n}]$ if the transducer is at a node in the input tree which is annotated with a.

$$q^a(C[x_1,\ldots,x_n]) \to C'[\overline{q}(x_{i_1}),\ldots,\overline{q}(x_{i_n})]. \tag{3.5}$$

Lemma 3.3. *Let t,t' be trees, and let R be a linear annotated rewrite system. Then $t \to_R t'$ iff $(t,t') \in \tau_{M_R}$.*

3.4. Pre-images under linear context tree translations

Finally, we show how to compute the pre-image of a regular tree language under the translation relation of a linear context tree transducer.

Proposition 3.4. *Let L be a regular tree language, and let M be a linear context tree transducer. Then $\tau_M^{-1}(L)$ is a regular tree language.*

Proof. Let $M = (P, \Sigma, \Delta, p_0, \delta)$, and let $B = (Q, \Delta, q_0, \gamma)$ be a top-down tree automaton with $\mathscr{L}(B) = L$. We construct a context tree automaton $A = (P \times Q, \Sigma, \Sigma, \langle p_0, q_0\rangle, \eta)$ with $\mathscr{L}(A) = \tau_M^{-1}(L)$.

Let $p(C[x_1, \ldots, x_n]) \to D[p_1(x_{i_1}), \ldots, p_n(x_{i_n})]$ be in δ. Furthermore, let $q(D[x_1, \ldots, x_n]) \to_B^* D[q_1(x_1), \ldots, q_n(x_n)]$, where we extend $\to_B$ to a binary relation on $T_{Q \cup \Delta \cup \{x_1, \ldots, x_n\}}$ by using $x_i \to_B x_i$ for all i. Then we let A contain the transition rule

$$\langle p, q\rangle(C[x_1, \ldots, x_n]) \to C[\langle p_{k_1}, q_{k_1}\rangle(x_1), \ldots, \langle p_{k_n}, q_{k_n}\rangle(x_n)],$$

where $k_{i_j} = j$ for all j.

Intuitively, A should read a context C if M can translate this into a context which B accepts. We keep track of the states in which M and B are during this process in A's state. If A is in a state $\langle p, q\rangle$, M must translate C from state p; it will output a context D, which B must accept from state q. During its run, M assigns states $p_1, \ldots, p_n$ to the holes of C; similarly, B assigns states $q_1, \ldots, q_n$ to the holes of D. Because M is linear, the holes of C and D correspond to each other bijectively, and we can build the new states $\langle p_i, q_i\rangle$ in which A must then read the subtrees below the holes of C. ∎

3.5. Complexity

In the worst case, the algorithm in the proof of Prop. 3.4 constructs a tree automaton with at most $|P| \cdot |Q|$ states and at most $|\delta| \cdot |Q| \cdot m$ transition rules, where m is the maximum number of state tuples $(q_1, \ldots, q_n)$ which B can assign to the holes of any context D on the right-hand side of a rule in M. If B is deterministic, we have $m = 1$, i.e. we can construct the pre-image automaton in time $O(|B| \cdot |M|) = O(|B| \cdot |R|)$. This means that by exploiting the linearity, we avoid the exponential blow-up in the automaton size from the domain automaton construction in [17]. If B is nondeterministic, m may be exponential in the size of M, where the exponent is the maximum number of variables on the right-hand side of a transition rule.

One further complication is that the construction in Lemma 3.1 requires us to compute the complement of the pre-image automaton A. Even if B is deterministic, A may not be, and so the automaton for $\overline{\mathscr{L}(A)}$ may be exponentially larger because A must be determinized. However, the rewriting systems and tree automata that we use in our application have certain properties that make the deterministic pre-image automata small as well. We first define these special properties, and then prove the complexity result.

Definition 3.5. The *left-hand size* of a rewriting system R over Σ is the maximum number of constructors from Σ that is used on the left-hand side of a rule in R.

We call a top-down tree automaton *both way deterministic* if it is deterministic and the bottom-up tree automaton that is obtained by reversing all transition rules is also deterministic.

Now we can prove the key complexity result for our application.

Proposition 3.6. *Let B be a both way deterministic top-down tree automaton, and let R be a linear rewriting system of left-hand size at most 2. Then it is possible to compute a deterministic bottom-up tree automaton A such that $\mathscr{L}(A) = \tau_{M_R}^{-1}(\mathscr{L}(B))$ in time $O(|B| \cdot |R|)$.*

Proof. First, we build the nondeterministic top-down context automaton N with $\mathscr{L}(N) = \tau_{M_R}^{-1}(\mathscr{L}(B))$ as in the proof of Prop. 3.4. This automaton has three types of rules, of the following forms:

(1) $\langle p^a, q\rangle(f(x_1,\ldots,x_n)) \to f(\langle \bar{p}, q_1\rangle(x_1),\ldots,\langle p^{a'}, q_i\rangle(x_i),\ldots,\langle \bar{p}, q_n\rangle(x_n))$
(2) $\langle \bar{p}, q\rangle(f(x_1,\ldots,x_n)) \to f(\langle \bar{p}, q_1\rangle(x_1),\ldots,\langle \bar{p}, q_n\rangle(x_n))$
(3) $\langle p^a, q\rangle(C[x_1,\ldots,x_n]) \to C[\langle \bar{p}, q_1\rangle(x_1),\ldots,\langle \bar{p}, q_n\rangle(x_n)]$

Rules of types 1 and 2 encode decisions of the transducer to copy symbols, whereas rules of type 3 encode a decision to rewrite C into some other context.

In a second step, we build an ordinary nondeterministic bottom-up tree automaton N' such that $\mathscr{L}(N') = \mathscr{L}(N)$. This can be done by breaking the transition rules for contexts with more than one constructor up into ordinary transition rules that read single constructors, and reversing the direction of all arrows. We can do this for the three rule types as follows:

(1) $f(\langle \bar{p}, q_1\rangle(x_1),\ldots,\langle p^{a'}, q_i\rangle(x_i),\ldots,\langle \bar{p}, q_n\rangle(x_n)) \to \langle p^a, q\rangle(f(x_1,\ldots,x_n))$
(2) $f(\langle \bar{p}, q_1\rangle(x_1),\ldots,\langle \bar{p}, q_n\rangle(x_n)) \to \langle \bar{p}, q\rangle(f(x_1,\ldots,x_n))$
(3) If C is of the form $f(x, g(y,z))$ and there is a type 2 rule in N' of the form $g(\langle \bar{p}, q_2\rangle(x_2), \langle \bar{p}, q_3\rangle(x_3)) \to \langle \bar{p}, q'\rangle(g(x_2,x_3))$, then the rule $\langle p^a, q\rangle(f(x_1, g(x_2,x_3))) \to f(\langle \bar{p}, q_1\rangle(x_1), g(\langle \bar{p}, q_2\rangle(x_2), \langle \bar{p}, q_3\rangle(x_3)))$ gets broken up into the following two rules:

$$g(\langle \bar{p}, q_2\rangle(x_2), \langle \bar{p}, q_3\rangle(x_3)) \to \langle p^g, q'\rangle(g(x_2,x_3))$$
$$f(\langle \bar{p}, q_1\rangle(x_1), \langle p^g, q'\rangle(y)) \to \langle p^a, q\rangle(f(x_1,y)).$$

The form $f(g(x,y),z)$ is analogous. If C is of the form $f(x,y)$, then we simply reverse the rule into $f(\langle \bar{p}, q_1\rangle(x_1), \langle \bar{p}, q_2\rangle(x_2)) \to \langle p^a, q\rangle(f(x_1,x_2))$.

Finally, we determinize N' into a deterministic bottom-up automaton A such that $\mathscr{L}(A) = \mathscr{L}(N')$ and A does not contain states that are not reachable in a bottom-up run of the automaton. According to the standard construction, we have $Q_A \subseteq \mathscr{P}(Q_{N'})$; but which of these states are actually reachable? First, for any $\mathfrak{q} \in Q_A$, if $\langle p, q\rangle \in \mathfrak{q}$ and $\langle p', q'\rangle \in \mathfrak{q}$, then $q = q'$: Because B, if read as a bottom-up transducer, is deterministic, the q on the right-hand sides of type 1 and 2 rules in N' is uniquely determined by f and the $q_1,\ldots,q_n$, and the type 3 rules are constructed to maintain this invariant too. Furthermore, we know that for every $\mathfrak{q} \in Q_A$, there is a $q \in Q_B$ such that $\langle \bar{p}, q\rangle \in \mathfrak{q}$, by induction using the type 2 rules. We also know that there is at most one p^f for $f \in \Sigma$ such that there is a q with $\langle p^f, q\rangle \in \mathfrak{q}$: namely the one for the most recent f that was read with a type 3 rule. Finally, $\mathfrak{q}$ may contain an arbitrary number of pairs of the form $\langle p^a, q\rangle$ for annotations a.

This means that $|Q_A| = O(|Q_B| \cdot |\Sigma| \cdot 2^{|\mathsf{Ann}|})$ where Ann is the annotation alphabet, i.e. the size of A's state alphabet is linear in that of B. In addition, A has at most as many transition rules as N'. If B has k rules and m is the maximum arity of symbols in Σ, this amounts to $k \cdot m$ rules of type 1, k rules of type 2, and at most $2 \cdot |R| \cdot |Q_B|$ rules of type 3. That is, the size of A's rule set is linear in k. Because both N and N' can be computed from B in linear time (as we argued above), this means that we compute A in linear time. ∎

In our application to scope underspecification, the tree automata A_G^u for the unlabeled configurations of a hnc dominance graph (e.g., the unlabeled version of the automaton in Fig. 2) are both way deterministic, and the rewriting rules we use (such as (2.2)) only permute two adjacent constructors, i.e. they are all of left-hand size two. In other words, for any A_G^u we can compute a deterministic automaton for the pre-image language in linear time. It is then straightforward to compute the complement automaton $\bar{A}$ and intersect it with A_G^u, obtaining an automaton A_W as the end result; this last step can take time $O(|A_G^u| \cdot |\bar{A}|) = O(|A_G^u|^2)$ in the worst case. Altogether we obtain an algorithm for

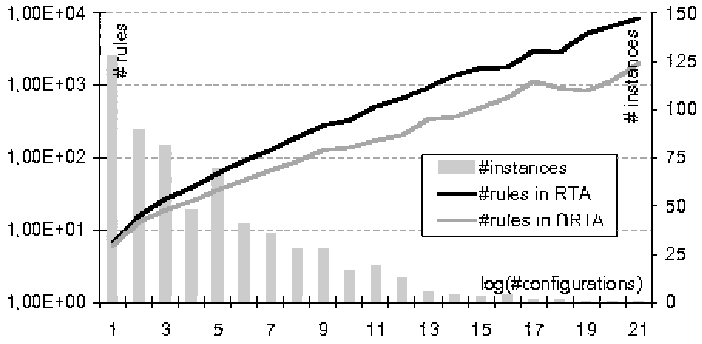

Figure 3: Sizes of automata in the Rondane treebank.

computing weakest readings that is quadratic in $|A^u_G|$, which is a huge improvement over the best previous algorithm, which was quadratic in $|\mathscr{L}(A^u_G)|$.

We have implemented a version of this algorithm which is further optimized to exploit certain properties of dominance graphs, and evaluated it empirically [21]. We ran the algorithm on the tree automata obtained from the dominance graphs for a subset of 623 sentences in the Rondane corpus [13]. Each of these dominance graphs describes a set of formulas of a variant of higher-order predicate logic, for which we chose suitable rewriting rules for approximating entailment. We find that the mean number of configurations represented by the automata drops from about three million for the original automata to 4.5 for the resulting automata A_W, and we reduce 67% of the sentences to a single weakest reading. The entire computation can be performed in about 20 ms per sentence on average. This means that although modeling weakest readings in terms of a rewriting system is incomplete with respect to true logical entailment, the approximation and our algorithm are highly useful on practical data.

It is interesting to observe that although the intersection construction can potentially make A_W larger than A^u_G, this does not actually happen in practice. This is shown in Fig. 3: The black line plots the sizes of the A^u_G, whereas the grey line plots the sizes of the A_W. The horizontal axis represents groups of sentences, where each group i contains all sentences with $\lceil e^{i-1} \rceil$ to $\lfloor e^i \rfloor$ readings; the vertical axis plots the mean number of transition rules in the automata (note the logarithmic scale). The grey bars indicate the number of sentences in each group. As the figure shows, the A_W tend to be much smaller than the original automata for each group. Averaged over all automata obtained from the subcorpus, the original automata have 180 transitions, whereas the result automata have 68; 87% of the automata are smaller after the intersection. It remains an open question to explain why the intersection decreases the automaton size so consistently.

4. An example

We finish by demonstrating our algorithm on the initial problem of computing the weakest readings of the dominance graph in Fig. 1(a). We assume the following annotated rewriting system, in which (2.2) is repeated as (4.1):

$$+ : \exists_y(P, \forall_x(Q,R)) \to \forall_x(Q, \exists_y(P,R)) \tag{4.1}$$

$$- : \forall_x(P, \exists_y(Q,R)) \to \exists_y(Q, \forall_x(P,R)) \tag{4.2}$$

$$+ : \neg(\exists_y(P,Q)) \to \exists_y(P, \neg(Q)) \tag{4.3}$$

$$+ : \forall_x(P, \neg(Q)) \to \neg(\forall_x(P,Q)) \tag{4.4}$$

This rewrite system translates into a top-down context tree transducer M_R with the following transition rules; we show the type 1 and 2 rules only for $\exists_y$.

$$p^+(\exists_y(x_1,\forall_x(x_2,x_3))) \to \forall_x(\overline{p}(x_2),\exists_y(\overline{p}(x_1),\overline{p}(x_3)))$$
$$p^+(\forall_x(x_1,\exists_y(x_2,x_3))) \to \exists_y(\overline{p}(x_2),\forall_x(\overline{p}(x_1),\overline{p}(x_3)))$$
$$p^+(\forall_x(x_1,\neg(x_2))) \to \neg(\forall_x(\overline{p}(x_1),\overline{p}(x_2)))$$
$$p^+(\neg(\exists_y(x_1,x_2))) \to \exists_y(\overline{p}(x_1),\neg(\overline{p}(x_2)))$$
$$p^+(\exists_y(x_1,x_2)) \to \exists_y(\overline{p}(x_1),p^+(x_2)) \qquad p^+(\exists_y(x_1,x_2)) \to \exists_y(p^+(x_1),\overline{p}(x_2))$$
$$p^-(\exists_y(x_1,x_2)) \to \exists_y(\overline{p}(x_1),p^-(x_2)) \qquad p^-(\exists_y(x_1,x_2)) \to \exists_y(p^-(x_1),\overline{p}(x_2))$$
$$\overline{p}(\exists_y(x_1,x_2)) \to \exists_y(\overline{p}(x_1),\overline{p}(x_2)) \qquad \ldots$$
$$\overline{p}(\mathsf{stud}_x) \to \mathsf{stud}_x \qquad \overline{p}(\mathsf{book}_y) \to \mathsf{book}_y \qquad \overline{p}(\mathsf{read}_{x,y}) \to \mathsf{read}_{x,y}$$

We can now consider an automaton A_G representing the configurations of a dominance graph, as in Fig. 2, and compute a nondeterministic top-down context automaton N with $L(N) = \tau_{M_R}^{-1}(L(A_G))$, as in the proof of Prop. 3.4. In the example, N looks as follows. Note that we are only showing transitions between productive states, and we abbreviate the state $\langle q_{G'}, p^a\rangle$ as $q^a_{G'}$. Note also that the construction in Prop. 3.4 would usually be executed on the unlabeled version A^u_G of A_G, but we show the labeled version here because A_G is already both ways deterministic in this example, and easier to read.

$$q^+_{1,\ldots,6}(\exists_y(x_1,\forall_x(x_2,x_3))) \to \exists_y(\overline{q}_5(x_1),\forall_x(\overline{q}_4(x_2),\overline{q}_{1,6}(x_3)))$$
$$q^+_{1,\ldots,6}(\neg(\exists_y(x_1,x_2))) \to \neg(\exists_y(\overline{q}_5(x_1),\overline{q}_{2,4,6}(x_2))) \qquad q^+_{1,\ldots,6}(\forall_x(x_1,x_2)) \to \forall_x(\overline{q}_4(x_1),q^+_{1,3,5,6}(x_2))$$
$$q^+_{1,\ldots,6}(\neg(x_1)) \to \neg(q^-_{2,\ldots,6}(x_1)) \qquad \overline{q}_{1,\ldots,6}(\exists_y(x_1,x_2))) \to \exists_y(\overline{q}_5(x_1),\overline{q}_{1,2,4,6}(x_2))$$
$$\overline{q}_{1,\ldots,6}(\forall_x(x_1,x_2)) \to \forall_x(\overline{q}_4(x_1),\overline{q}_{1,3,5,6}(x_2)) \qquad \overline{q}_{1,\ldots,6}(\neg(x_1)) \to \neg(\overline{q}_{2,\ldots,6}(x_1))$$
$$q^-_{2,\ldots,6}(\forall_x(x_1,\exists_y(x_2,x_3))) \to \forall_x(\overline{q}_4(x_1),\exists_y(\overline{q}_5(x_2),\overline{q}_6(x_3)))$$
$$\overline{q}_{2,\ldots,6}(\exists_y(x_1,x_2))) \to \exists_y(\overline{q}_5(x_1),\overline{q}_{2,4,6}(x_2)) \qquad \overline{q}_{2,\ldots,6}(\forall_x(x_1,x_2)) \to \forall_x(\overline{q}_4(x_1),\overline{q}_{3,5,6}(x_2))$$
$$q^+_{1,3,5,6}(\neg(\exists_y(x_1,x_2))) \to \neg(\exists_y(\overline{q}_5(x_1),\overline{q}_6(x_2)))$$
$$\overline{q}_{1,3,5,6}(\exists_y(x_1,x_2))) \to \exists_y(\overline{q}_5(x_1),\overline{q}_{1,6}(x_2)) \qquad \overline{q}_{1,3,5,6}(\neg(x_1)) \to \neg(\overline{q}_{3,5,6}(x_1))$$
$$\overline{q}_{1,2,4,6}(\forall_x(x_1,x_2)) \to \forall_x(\overline{q}_4(x_1),\overline{q}_{1,6}(x_2)) \qquad \overline{q}_{1,2,4,6}(\neg(x_1)) \to \neg(\overline{q}_{2,4,6}(x_1))$$
$$\overline{q}_{2,4,6}(\forall_x(x_1,x_2)) \to \forall_x(\overline{q}_4(x_1),\overline{q}_6(x_2)) \qquad \overline{q}_{3,5,6}(\exists_y(x_1,x_2))) \to \exists_y(\overline{q}_5(x_1),\overline{q}_6(x_2))$$
$$\overline{q}_{1,6}(\neg(x_1)) \to \neg(\overline{q}_6(x_1)) \qquad \overline{q}_4(\mathsf{stud}_x) \to \mathsf{stud}_x \qquad \overline{q}_5(\mathsf{book}_y) \to \mathsf{book}_y \qquad \overline{q}_6(\mathsf{read}_{x,y}) \to \mathsf{read}_{x,y}$$

Finally, we compute a deterministic ordinary bottom-up automaton A with $L(A) = L(N)$ as in the proof of Prop. 3.6. This involves breaking up rules whose left-hand sides are nontrivial contexts up into ordinary rules, reorienting the transitions into bottom-up rules, and determinizing the resulting automaton. The states of the determinized automaton are sets that contain states of N and states q^f_G that were introduced when breaking up the context rules; we suppress the set brackets for singleton sets below. Notice that as we claimed in the proof of Prop. 3.6, if any two of $\overline{q}_{G'}$ and $q^+_{G''}$ or $q^-_{G''}$ are in the same state set, then $G' = G''$; and there is at most one state $q^f_{G'}$ for any $f \in \Sigma$ in each such state set. A looks as follows:

$$\exists_y(\overline{q}_5(x_1), \{q^{\forall x}_{1,2,4,6}, \overline{q}_{1,2,4,6}\}(x_2)) \to \{q^{+}_{1,\ldots,6}, \overline{q}_{1,\ldots,6}\}(\exists_y(x_1,x_2))$$
$$\neg(\{q^{\exists y}_{2,\ldots,6}, \overline{q}_{2,\ldots,6}\}(x_1)) \to \{q^{+}_{1,\ldots,6}, \overline{q}_{1,\ldots,6}\}(\neg(x_1))$$
$$\forall_x(\overline{q}_4(x_1), \{q^{+}_{1,3,5,6}, \overline{q}_{1,3,5,6}\}(x_2)) \to \{q^{+}_{1,\ldots,6}, \overline{q}_{1,\ldots,6}\}(\forall_x(x_1,x_2))$$
$$\neg(\{q^{-}_{2,\ldots,6}, \overline{q}_{2,\ldots,6}\}(x_1)) \to \{q^{+}_{1,\ldots,6}, \overline{q}_{1,\ldots,6}\}(\neg(x_1))$$
$$\exists_y(\overline{q}_5(x_1), \overline{q}_{2,4,6}(x_2))) \to \{q^{\exists y}_{2,\ldots,6}, \overline{q}_{2,\ldots,6}\}(\exists_y(x_1,x_2))$$
$$\forall_x(\overline{q}_4(x_1), \{q^{\exists y}_{3,5,6}, \overline{q}_{3,5,6}\}(x_2)) \to \{q^{-}_{2,\ldots,6}, \overline{q}_{2,\ldots,6}\}(\forall_x(x_1,x_2))$$
$$\forall_x(\overline{q}_4(x_1), \overline{q}_{1,6}(x_2))) \to \{q^{\forall x}_{1,2,4,6}, \overline{q}_{1,2,4,6}\}(\forall_x(x_1,x_2))$$
$$\neg(\overline{q}_{2,4,6}(x_1)) \to \overline{q}_{1,2,4,6}(\neg(x_1))$$
$$\neg(\{q^{\exists y}_{3,5,6}, \overline{q}_{3,5,6}\}(x_1)) \to \{q^{+}_{1,3,5,6}, \overline{q}_{1,3,5,6}\}(\neg(x_1))$$
$$\forall_x(\overline{q}_4(x_1), \overline{q}_6(x_2)) \to \overline{q}_{2,4,6}(\forall_x(x_1,x_2))$$
$$\exists_y(\overline{q}_5(x_1), \overline{q}_6(x_2))) \to \{q^{\exists y}_{3,5,6}, \overline{q}_{3,5,6}\}(\exists_y(x_1,x_2))$$
$$\neg(\overline{q}_6(x_1)) \to \overline{q}_{1,6}(\neg(x_1)) \qquad \mathsf{stud}_x \to \overline{q}_4(\mathsf{stud}_x) \qquad \mathsf{book}_y \to \overline{q}_5(\mathsf{book}_y) \qquad \mathsf{read}_{x,y} \to \overline{q}_6(\mathsf{read}_{x,y})$$

A is a deterministic automaton which accepts four trees, namely the configurations (b), (c), (e), and (f) in Fig. 1. Therefore we can obtain an automaton which accepts exactly the weakest readings of the graph in Fig. 1 – i.e., (d) and (g) – by intersecting A_G and the complement automaton $\overline{A}$.

5. Conclusion

In this paper, we have presented an algorithm for computing those members of a regular tree language L that are in relative normal form with respect to an annotated rewriting system R. We have shown how to compute these elements by computing the pre-image of the tree language under a transducer encoding the one-step rewriting relation, and then intersecting the language with the complement of this pre-image. By defining *context* tree transducers, we were able to compute the pre-image in linear time if L is given in terms of a deterministic automaton; for the special case where R has left-hand sides of size at most two, we could show that even the deterministic automaton for the pre-image is linear in size. This restriction holds in our application to computational linguistics, where our results provide an approximate, but practically useful solution to the problem of computing weakest readings.

From the perspective of our application, our results open up a whole new class of rewriting-based inferences on natural-language meaning representations which can now be processed efficiently. We will explore such inferences in the future. One line of research that seems particularly intriguing is to deal with cases where multiple trees that are equivalent with respect to the underlying rewrite system are left over in the language of the final tree automaton. Such cases can happen when the rewrite system is not confluent. It would be interesting to investigate the practical impact of augmenting the permutation system, e.g. with the Knuth-Bendix completion procedure. This trades off a reduction in the number of relative normal forms (due to improved confluence) against an increase in the size of the rewrite system.

Acknowledgments. We would like to thank the reviewers and particularly Joachim Niehren for their extremely helpful comments, which influenced this paper substantially and improved it a lot.

References

[1] Dagan, I., Glickman, O., Magnini, B.: The PASCAL recognising textual entailment challenge. In Quiñonero-Candela, J., Dagan, I., Magnini, B., d'Alché Buc, F., eds.: Machine Learning Challenges. Volume 3944 of Lecture Notes in Computer Science. Springer (2006) 177–190

[2] Montague, R.: The proper treatment of quantification in ordinary English. In Thomason, R., ed.: Formal Philosophy. Selected Papers of Richard Montague. Yale University Press, New Haven (1974)

[3] van Deemter, K., Peters, S.: Semantic Ambiguity and Underspecification. CSLI (1996)

[4] Egg, M., Koller, A., Niehren, J.: The Constraint Language for Lambda Structures. Logic, Language, and Information **10** (2001) 457–485

[5] Copestake, A., Flickinger, D., Pollard, C., Sag, I.: Minimal recursion semantics: An introduction. Journal of Language and Computation (2005)

[6] Blackburn, P., Bos, J.: Representation and Inference for Natural Language. A First Course in Computational Semantics. CSLI Publications (2005)

[7] Althaus, E., Duchier, D., Koller, A., Mehlhorn, K., Niehren, J., Thiel, S.: An efficient graph algorithm for dominance constraints. Journal of Algorithms **48** (2003) 194–219

[8] Bos, J.: Let's not argue about semantics. In: Proceedings of LREC. (2008) 2835–2840

[9] Kempson, R., Cormack, A.: Ambiguity and quantification. Linguistics and Philosophy **4** (1981) 259–309

[10] Hobbs, J.: An improper treatment of quantification in ordinary English. In: Proceedings of the 21st ACL. (1983)

[11] Gabsdil, M., Striegnitz, K.: Classifying scope ambiguities. In: Proceedings of the First Intl. Workshop on Inference in Computational Semantics. (1999)

[12] Koller, A., Regneri, M., Thater, S.: Regular tree grammars as a formalism for scope underspecification. In: Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL-08: HLT), Columbus, Ohio (2008)

[13] Oepen, S., Toutanova, K., Shieber, S., Manning, C., Flickinger, D., Brants, T.: The LinGO Redwoods treebank: Motivation and preliminary applications. In: Proceedings of the 19th International Conference on Computational Linguistics (COLING'02). (2002) 1253–1257

[14] Fuchss, R., Koller, A., Niehren, J., Thater, S.: Minimal recursion semantics as dominance constraints: Translation, evaluation, and analysis. In: Proc. of ACL, Barcelona (2004)

[15] Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata` (2007)

[16] Bodirsky, M., Duchier, D., Niehren, J., Miele, S.: An efficient algorithm for weakly normal dominance constraints. In: ACM-SIAM Symposium on Discrete Algorithms. (2004)

[17] Engelfriet, J.: Top-down tree transducers with regular look-ahead. Math. Systems Theory **10** (1977) 289–303

[18] Engelfriet, J., Maneth, S.: A comparison of pebble tree transducers with macro tree transducers. Acta Informatica **39**(9) (2003) 613–698

[19] Maneth, S., Berlea, A., Perst, T., Seidl, H.: Xml type checking with macro tree transducers. In: PODS '05: Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, New York, NY, USA, ACM (2005) 283–294

[20] Graehl, J., Knight, K., May, J.: Training tree transducers. Computational Linguistics **34**(3) (2008)

[21] Koller, A., Thater, S.: Computing weakest readings. In: Proceedings of the 48th ACL. (2010)