Towards a redundancy elimination algorithm for underspecified descriptions

Alexander Koller and Stefan Thater

Department of Computational Linguistics
Universität des Saarlandes, Saarbrücken, Germany
{*koller,stth*}*@coli.uni-sb.de*

Abstract

This paper proposes an efficient algorithm for the *redundancy elimination* problem: Given an underspecified semantic representation (USR), compute an USR which has fewer readings, but still describes at least one representative of each semantic equivalence class of the original readings. The algorithm operates on underspecified chart representations which are derived from dominance graphs; it can be applied to the USRs computed by large-scale grammars. To our knowledge, it is the first redundancy elimination algorithm which maintains underspecification, rather than just enumerating non-redundant readings.

1 Introduction

Underspecification is a standard approach to dealing with scope ambiguities in computational semantics [12,6,7,2]. The basic idea is to not enumerate all possible semantic representations for each syntactic analysis, but to derive a single compact *underspecified representation (USR)*. This simplifies semantics construction, and current algorithms support the efficient enumeration of readings from an USR [10].

In addition, underspecification has the potential for eliminating incorrect or redundant readings by inferences based on context or world knowledge, without even enumerating them. For instance, sentences with scope ambiguities often have readings which are semantically equivalent. In this case, we typically need to retain only one reading from each equivalence class. This situation is illustrated by the following two sentences from the Rondane treebank, which is distributed with the English Resource Grammar (ERG; [5]), a broad-coverage HPSG grammar.

(1) For travellers going to Finnmark there is a bus service from Oslo to Alta through Sweden. (Rondane 1262)

(2) We quickly put up the tents in the lee of a small hillside and cook for the first time in the open. (Rondane 892)

For the two example sentences, the ERG (Version 01-2006) derives USRs with seven and six quantifiers, respectively, that correspond to various types of noun phrases (including proper names and pronouns). The USR for (1) describes 3960 readings, which are all semantically equivalent to each other. On the other hand, the USR for (2) has 480 readings, which fall into two classes of mutually equivalent readings, characterised by the relative scope of “the lee of” and “a small hillside.”

This paper presents an algorithm for the *redundancy elimination* problem: Given an USR, compute an USR which has fewer readings, but still describes at least one representative of each equivalence class – without enumerating any readings. This algorithm computes the one or two representatives of the semantic equivalence classes in the above examples, so subsequent modules don’t have to deal with all the other equivalent readings. It also closes the gap between the large number of readings predicted by the grammar and the intuitively perceived much lower degree of ambiguity of these sentences. Finally, it can be helpful for a grammar designer because it is much more feasible to check whether two readings are linguistically reasonable than 480.

We model equivalence in terms of rewrite rules that permute quantifiers without changing the semantics of the readings. The particular USRs we work with are underspecified chart representations, which can be computed from dominance graphs (or USRs in some other underspecification formalisms) efficiently [10]. The algorithm can deal with many interesting cases, but is incomplete in the sense that the resulting USR may still describe multiple equivalent readings.

To our knowledge, this is the first algorithm in the literature for redundancy elimination on the level of USRs. There has been previous research on *enumerating* only some representatives of each equivalence class [13,4], but these approaches don’t maintain underspecification: After running their algorithms, we have a set of readings rather than an underspecified representation.

Plan of the paper. We will first define dominance graphs and review the necessary background theory in Section 2. We will then give a formal definition of equivalence and derive some first results in Section 3. Section 4 presents the redundancy elimination algorithm. Finally, Section 5 concludes and points to further work.

2 Dominance Graphs

The basic underspecification formalism we assume here are *labelled dominance graphs* [1]. Dominance graphs are equivalent to leaf-labelled normal dominance constraints [7], which have been discussed extensively in previous literature.

Definition 2.1 A *(compact) dominance graph* is a directed graph $(V, E \uplus D)$ with two kinds of edges, *tree edges E* and *dominance edges D*, such that:

(i) the graph (V,E) defines a collection of node disjoint trees of height 0 or 1. We call the trees in (V,E) the *fragments* of the graph.

(ii) if (v,v') is a dominance edge in D, then v is a hole and v' is a root in G. A node v is a *root* (in G) if v does not have incoming tree edges; otherwise, v is a *hole*.

A *labelled dominance graph* over a ranked signature Σ is a triple $G = (V, E \uplus D, L)$

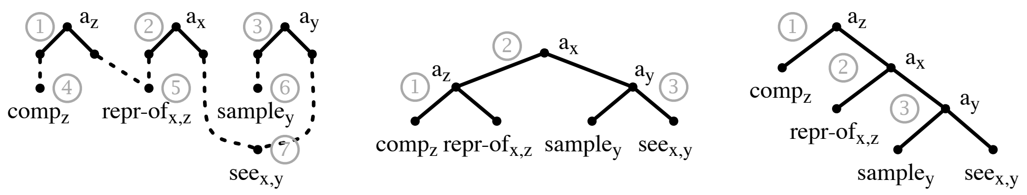

Fig. 1. A dominance graph that represents the five readings of the sentence "a representative of a company saw a sample" (left) and two (of five) configurations.

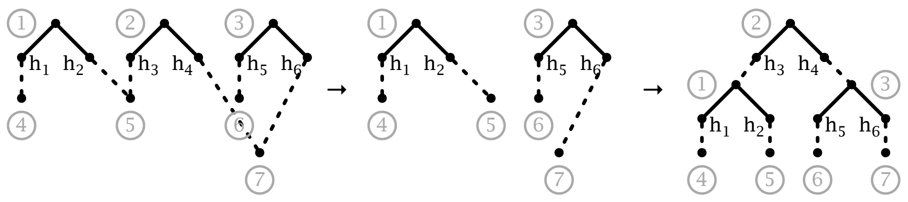

Fig. 2. An example computation of a solved form.

such that $(V, E \uplus D)$ is a dominance graph and $L : V \rightsquigarrow \Sigma$ is a partial *labelling function* which assigns a node v a label with arity n iff v is a root with n outgoing tree edges. Nodes without labels (i.e., holes) must have outgoing dominance edges.

We will write $v{:}f(v_1,\dots,v_k)$ for a fragment whose root v is labelled with f and whose holes are $v_1,\dots,v_k$. We will write $R(F)$ for the root of the fragment F, and we will typically just say *graph* instead of *labelled dominance graph*.

An example of a labelled dominance graph is shown to the left of Fig. 1. Tree edges are drawn as solid lines, and dominance edges are drawn as dotted lines, directed from top to bottom. This graph can serve as an USR for the sentence "a representative of a company saw a sample" if we demand that the holes are "plugged" by roots while realising the dominance edges as dominance, as in the two (of five) *configurations* shown to the right [7]. Configurations encode semantic representations of the sentence, and we freely read configurations as ground terms over Σ.

2.1 Solving dominance graphs

Algorithms for *solving* a dominance graph in order to compute the readings it describes typically compute its *minimal solved forms* [1,3]. In this paper, we restrict ourselves to *hypernormally connected* graphs (defined below), for which one can show that all solved forms are minimal and bijectively correspond to configurations.

Let G, G' be dominance graphs. We say that G is *in solved form* iff it is a forest, and G is a *solved form of* G' if G is in solved form and more specific than G' i.e., G and G' have the same labels and tree fragments, and the reachability relation of G extends that of G'. G' is *solvable* if it has a solved form G. If G' is hypernormally connected, then each hole in G has exactly one outgoing dominance edge, and G can be mapped to a configuration by identifying the two ends of each dominance edge; conversely, we can find a unique solved form for each configuration. The graph to the left of Fig. 2 shows one of the (minimal) solved forms of the example graph, which corresponds to the configuration in the middle of Fig. 1.

```
COMPUTE-CHART($G$)
 1  if there is an entry for $G$ in the chart
 2     then return true
 3  $free \leftarrow$ FREE-FRAGMENTS($G$)
 4  if $free = \emptyset$
 5     then return false
 6  if $G$ contains only one fragment
 7     then return true
 8  for each $F \in free$
 9  do $split \leftarrow$ SPLIT($G, F$)
10       for each $S \in$ WCCS($G - F$)
11       do if COMPUTE-CHART($S$) = false
12             then return false
13       add $(G, split)$ to the chart
14  return true
```

$\{1,2,3,4,5,6,7\}$:	$\langle 1, h_1 \mapsto \{4\}, h_2 \mapsto \{2,3,5,6,7\}\rangle$
	$\langle 2, h_3 \mapsto \{1,4,5\}, h_4 \mapsto \{3,6,7\}\rangle$
	$\langle 3, h_5 \mapsto \{5\}, h_6 \mapsto \{1,2,4,5,7\}\rangle$
$\{2,3,5,6,7\}$:	$\langle 2, h_3 \mapsto \{5\}, h_4 \mapsto \{3,6,7\}\rangle$
	$\langle 3, h_5 \mapsto \{6\}, h_6 \mapsto \{2,5,7\}\rangle$
$\{3,6,7\}$:	$\langle 3, h_5 \mapsto \{6\}, h_6 \mapsto \{7\}\rangle$
$\{2,5,7\}$:	$\langle 2, h_3 \mapsto \{5\}, h_4 \mapsto \{7\}\rangle$
$\{1,4,5\}$:	$\langle 1, h_1 \mapsto \{4\}, h_2 \mapsto \{5\}\rangle$
$\{1,2,4,5,7\}$:	$\langle 1, h_1 \mapsto \{4\}, h_2 \mapsto \{2,5,7\}\rangle$
	$\langle 2, h_3 \mapsto \{1,4,5\}, h_4 \mapsto \{7\}\rangle$

Fig. 3. The chart solver and an example chart computed for the dominance graph in Fig. 2.

The key concept of the solver we build upon is that of a *free fragment* [3]. A fragment F in a solvable graph G is free iff there is a solved form in which F is at the root. It can be shown that a fragment is free iff it has no incoming dominance edges and its holes are in different biconnected components of the graph i.e., they are disconnected if the root of the fragment is removed from the graph [3]. Removing a free fragment from a graph splits the graph into different weakly connected components (wccs) – one for each hole. Thus each free fragment F induces a *split* of G, which consists of a reference to F and a mapping of the other fragments to the hole to which they are connected. For instance, the example graph has three free fragments: 1, 2, and 3. By removing fragment 2, the graph is decomposed into two wccs, which are connected to the holes h_3 and h_4, respectively (see Fig. 2).

The solver [10] is shown in Fig. 3. It computes a chart-like data structure which assigns sets of splits to subgraphs. For each subgraph it is called on, the solver computes the free fragments, the splits they induce, and calls itself recursively on the wccs of each split. It records subgraphs and splits in the chart, and will not repeat work for a subgraph it has encountered before. The algorithm returns true iff the original graph was solvable. The chart tells us how to build the minimal solved forms of the graph: For each subgraphs, pick any split, compute a solved form for each wcc recursively, and plug them into the given hole of the split's root fragment. As an example, the chart for the graph in Fig. 1 is shown to the right of Fig. 3.

Notice that the chart which the solver computes, while possibly exponentially larger than the original graph, is still exponentially smaller than the entire set of readings because common subgraphs (such as $\{2,5,7\}$ in the example) are represented only once. Thus the chart can still serve as an underspecified representation.

2.2 Hypernormally connected dominance graphs

A *hypernormal path* [1] in a graph G is a path in the undirected version G_u of G that does not use two dominance edges that are incident to the same hole. We say that G is *hypernormally connected (hnc)* iff each pair of nodes is connected by a simple hypernormal path in G. Hnc graphs are equivalent to *chain-connected* dominance constraints [9], and are closely related to *dominance nets* [11]. The results in this paper are restricted to hnc graphs, but this does not limit the applicability of our results: an empirical study suggests that all dominance graphs that are generated by current large-scale grammars are (or should be) hnc [8].

The key property of hnc dominance graphs is that their solved forms correspond to configurations, and we will freely switch between solved forms and their corresponding configurations. Another important property of hnc graphs which we will use extensively in the proofs below is that it is possible to predict which holes of fragments can dominate other fragments in a solved form.

Lemma 2.2 *Let G be a hnc graph with free fragment F. Then all weakly connected components of $G - F$ are hnc.*

Proposition 2.3 *Let F_1, F_2 be fragments in a hnc dominance graph G. If there is a solved form S of G in which $R(F_1)$ dominates $R(F_2)$, then there is exactly one hole h of F_1 which is connected to $R(F_2)$ by a simple hypernormal path which doesn't use $R(F_1)$. In particular, h dominates $R(F_2)$ in S.*

Proof. Let's say that F_1 dominates F_2 in some solved form S. There is a run of the solver which computes S. This run chooses F_1 as a free fragment before it chooses F_2. Let's call the subgraph in which the split for F_1 is chosen, G'. G' is hnc (Lemma 2.2), so in particular there is a simple hypernormal path from the hole h of F_1 which is in the same wcc as F_2 to $R(F_2)$; this path doesn't use $R(F_1)$. On the other hand, assume there were another hole h' of F_1 which is connected to $R(F_2)$ by a path that doesn't use $R(F_1)$. Then the path via $R(F_2)$ would connect h and h' even if $R(F_1)$ were removed, so h and h' would be in the same biconnected component of G, in contradiction to the assumption that F_1 is free in G'.

For the second result, note that F_2 is assigned to the hole h in the split for F_1. □

The following definition captures the complex condition in Prop. 2.3:

Definition 2.4 Let G be a hnc dominance graph. A fragment F_1 in G is called a *possible dominator* of another fragment F_2 in G iff it has exactly one hole h which is connected to $R(F_2)$ by a simple hypernormal path which doesn't use $R(F_1)$. We write $\mathsf{ch}(F_1, F_2)$ for this unique h.

3 Equivalence

Equivalence is traditionally defined as the relation between formulas which have the same interpretation. However, even first-order equivalence is an undecidable problem, thus an algorithm which checks for semantic equivalence of different configurations of a graph can't possibly be efficient. On the other hand, we do not need to solve the full semantic equivalence problem, as we only want to compare formulas that are readings of the same sentence i.e., different configurations of the same USR. Such formulas only differ in the way that the fragments are combined. We can therefore approximate equivalence by using a *rewrite system* that permutes fragments and defining equivalence of configurations as mutual rewritability as usual.

By way of example, consider again the two (equivalent) configurations shown in Fig. 1. We can obtain the second configuration from the first one by applying the following rewrite rule, which rotates the nodes 1 and 2:

$$\mathsf{a}_x(\mathsf{a}_z(P,Q),R) \to \mathsf{a}_z(P,\mathsf{a}_x(Q,R)) \tag{3}$$

The formulas on both sides of the arrow are semantically equivalent in first-order logic for any choice of the subformulas P, Q, and R. Thus the equivalence of the two configurations with respect to our one-rule rewrite system implies that they are also semantically equivalent.

While we will require that the rewriting approximation is *sound* i.e., rewrites formulas into equivalent formulas, we cannot usually hope to achieve *completeness* i.e., there will be semantic equivalences that are not modelled by the rewriting equivalence. However, we believe that the rewriting-based system will still prove to be useful in practical applications, as the permutation of quantifiers is exactly the kind of variability that an underspecified description allows.

We formalise this rewriting-based notion of equivalence as follows. The definition uses the abbreviation $\overline{x_{[1,k)}}$ for $x_1,\dots,x_{k-1}$, and $\overline{x_{(k,n]}}$ for $x_{k+1},\dots,x_n$.

Definition 3.1 A *permutation system R* is a system of rewrite rules over a signature Σ of the following form:

$$f_1(\overline{x_{[1,i)}}, f_2(\overline{y_{[1,k)}}, z, \overline{y_{(k,m]}}), \overline{x_{(i,n]}}) \;\to\; f_2(\overline{y_{[1,k)}}, f_1(\overline{x_{[1,i)}}, z, \overline{x_{(i,n]}}), \overline{y_{(k,m]}})$$

The *permutability relation* $P(R)$ is the binary relation $P(R) \subseteq (\Sigma \times \mathbb{N})^2$ which contains exactly the pairs $((f_1,i),(f_2,k))$ and $((f_2,k),(f_1,i))$ for each such rewrite rule.

As usual, we say that two terms are *equivalent* with respect to R, $s \approx_R t$, iff there is a sequence of rewrite steps and inverse rewrite steps that rewrite s into t. We say that R is *sound* with respect to a semantic notion of equivalence $\equiv$ if $\approx_R \subseteq \equiv$. If G is a graph over Σ and R a permutation system, then we write $SC_R(G)$ for the set of equivalence classes $\mathrm{Conf}(G)/\approx_R$, where $\mathrm{Conf}(G)$ is the set of configurations of G.

A rewrite system (let's call it R_{fol}) which is sound for the standard equivalence relation of first-order logic could use rule (3) and the three other permutations of two existential quantifiers, plus the following rule for universal quantifiers:

$$\mathsf{every}_x(X,\mathsf{every}_y(Y,Z)) \to \mathsf{every}_y(Y,\mathsf{every}_x(X,Z))$$

The other three permutations of universal quantifiers, as well as the permutations of universal and existential quantifiers, are not sound.

It is possible to compute $SC_R(G)$ by solving G and using a theorem prover for equational reasoning to compute the equivalence classes of the configurations, but this is very inefficient. To replace this by a computation on the USR, we must be able to recognise whether two fragments of a graph can be permuted in all configurations of the graph. This is not possible in general: If we don't know in advance

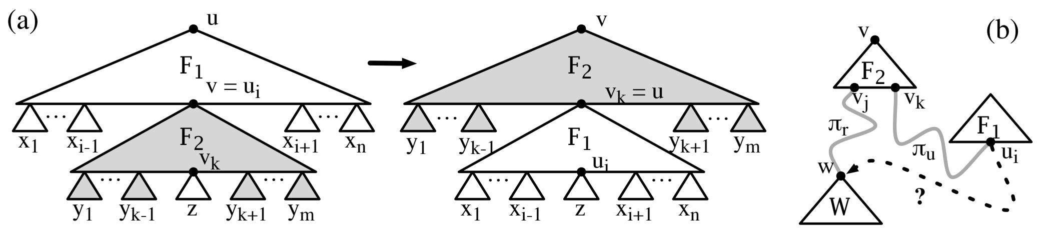

Fig. 4. Diagrams for the proof of Lemma 3.3

which hole of one fragment the other fragment can plug, we can't know whether the two fragments can be permuted. However, in a *hnc* graph, the hole of a fragment which another fragment can plug is determined uniquely (because of Lemma 2.3), and can be recognised without solving the graph.

Definition 3.2 Let R be a permutation system. Two fragments F_1 and F_2 with root labels f_1 and f_2 in a graph G are called *R-permutable* iff they are possible dominators of each other and $((f_1, \mathsf{ch}(F_1,F_2)), (f_2, \mathsf{ch}(F_2,F_1))) \in P(R)$.

Lemma 3.3 *Let R be a permutation system, let $F_1 = u{:}f_1(u_1,\dots,u_n)$ and $F_2 = v{:}f_2(v_1,\dots,v_m)$ be R-permutable fragments in the hnc graph G, such that F_2 is free, and let C_1 be a configuration of G in which u is the father of v. Then:*

(a) It is possible to apply a R-rewrite step or an inverse R-rewrite step to C_1 at u; call the resulting tree C_2.

(b) C_2 is also a configuration of G.

(c) $C_2 \approx_R C_1$.

Proof. Let $i = \mathsf{ch}(F_1,F_2)$ and $k = \mathsf{ch}(F_2,F_1)$; we know that $((f_1,i),(f_2,k)) \in P(R)$.

(a) F_1 is a possible dominator of F_2, so u_i is plugged with v in C_1 (Lemma 2.3). Thus the (possibly inverse) rule which justified the tuple $((f_1,i),(f_2,k))$ is applicable at u.

(b) We must verify that every dominance edge in G is realised by C_2. As Fig. 4a shows, all dominance edges that do not go out of a hole of F_1 are still trivially realised by C_2. Now let's consider dominances out of the holes of F_1.

- Dominance edges out of any u_j with $j \neq i$ are still satisfied (see the figure).
- Dominance edges from u_i to a node in z are still satisfied (see the figure).
- Dominance edges from u_i to v: Such edges cannot exist in G as F_2 is free.
- Dominance edges from u_i to a node w in some y_j with $j \neq k$: Such edges cannot exist either. F_2 is a possible dominator of the fragment W whose root w is, so there is a simple hypernormal path π_w from $\mathsf{ch}(F_2,W)$ to w which doesn't use v; $\mathsf{ch}(F_2,W) = v_j$ because v_j dominates w in C_1 (Lemma 2.3). On the other hand, F_2 is a possible dominator of F_1, so there is a simple hypernormal path π_u from v_k to u_i which doesn't use v. Now if there were a dominance edge from u_i to w in G, then v_j and v_k would be in the same biconnected component (they would be connected via $\pi_u \circ (u_i,w) \circ \pi_w^{-1}$ if v were removed), which contradicts the freeness of F_2 (see Fig. 4b). □

4 Underspecified redundancy elimination

Now we can finally consider the problem of strengthening an USR in order to remove redundant readings which are equivalent to other readings. We will define an algorithm which gets as its input a graph G, a chart as computed by COMPUTE-CHART, and a permutability relation $P(R)$. It will then remove splits from the chart, to the effect that the chart represents fewer solved forms of the original graph, but at least one representative from each class in $SC_R(G)$ remains. The subgraph sharing of the original chart will be retained, so the computed chart is still an USR.

The key concept in the redundancy elimination algorithm is that of a *permutable split*. Intuitively, a split of G is called permutable if its root fragment F is permutable with all other fragments in G which could end up above F. Because of Lemma 3.3, we can then always pull F to the root by a sequence of rewrite steps. This means that for any configuration of G, there is an equivalent configuration whose root is F – i.e., by choosing the split for F, we lose no equivalence classes.

Definition 4.1 Let R be a permutation system. A split S of a graph G is called *R-permutable* iff the root fragment F of S is R-permutable with all other fragments in G which are possible dominators of F in G.

In the graph of Fig. 1, all three splits are R_{fol}-permutable: For each of the upper fragments, the other two upper fragments are possible dominators, but as all three fragments are labelled with existential quantifiers and R_{fol} contains all permutations of existential quantifiers, the fragments are permutable with each other. And indeed, we can pick any of the three fragments as the root fragment, and the resulting split will describe a representative of the single equivalence class of the graph.

Proposition 4.2 *Let G be a hnc graph, and let S be a permutable split of G. Then $SC(S) = SC(G)$.*

Proof. If G is unsolvable, the claim is trivially true. Otherwise, let C be an arbitrary configuration of G; we must show that $S = (F, h_1 \mapsto G_1, \ldots, h_n \mapsto G_n)$ has a configuration C' which is equivalent to C.

Let's say that the fragments which properly dominate F in C are $F_1, \ldots, F_n$ ($n \geq 0$), ordered in such a way that F_i dominates F_j in C for all $i < j$. Each F_i is a possible dominator of F, by Prop. 2.3. Because S is permutable, this means that each F_i is permutable with F in G. By applying Lemma 3.3 n times (first to F and F_n, then to F and F_{n-1}, and so on), we can compute a configuration C' of G in which F is at the root and such that $C' \approx_R C$. But C is a configuration of S, which proves the theorem. $\square$

This suggests the following redundancy elimination algorithm:

```
REDUNDANCY-ELIMINATION($Ch, G, R$)
1  for each subgraph $G'$ in $Ch$
2      do if $G'$ has an $R$-permutable split $S$
3            then remove all splits for $G'$ except for $S$ from $Ch$
```

Because of Prop. 4.2, the algorithm is correct in that for each configuration C of G, the reduced chart still has a configuration C' with $C \approx_R C'$. The particular choice of S doesn't affect the correctness of the algorithm (but may change the number of remaining configurations). However, the algorithm is not complete in the sense that the reduced chart can have no two equivalent configurations. We will illustrate this below. We can further optimize the algorithm by deleting subgraphs (and their splits) that are not referenced anymore by using reference counters. This doesn't change the set of solved forms of the chart, but may further reduce the chart size.

In the running example, we would run REDUNDANCY-ELIMINATION on the chart in Fig. 3. As we have seen, all three splits of the entire graph are permutable, so we can pick any of them e.g., the split with root fragment 2, and delete the splits with root fragments 1 and 3. This reduces the reference count of some subgraphs (e.g. $\{2,3,5,6,7\}$) to 0, so we can remove these subgraphs too. The resulting chart is shown below, which represents a single solved form (the one shown in Fig. 2).

$$\begin{aligned}\{1,2,3,4,5,6,7\} &: \langle 2, h_2 \mapsto \{1,4\}, h_4 \mapsto \{3,6,7\}\rangle \\ \{1,4\} &: \langle 1, h_1 \mapsto \{4\}\rangle \\ \{3,6,7\} &: \langle 3, h_5 \mapsto \{6\}, h_6 \mapsto \{7\}\rangle\end{aligned}$$

Now consider variations of the graph in Fig. 1 in which the quantifier labels are different; these variant graphs have exactly the same chart, but fewer fragment pairs will be permutable. If all three quantifiers are universal, then the configurations fall into two equivalence classes which are distinguished by the relative scope of the fragments 1 and 2. The algorithm will recognise that the split with root fragment 3 is permutable and delete the splits for 1 and 2. The resulting chart has two solved forms. Thus the algorithm is still complete in this case. If, however, the fragments 1 and 2 are existential quantifiers and the fragment 3 is universal, there are three equivalence classes, but the chart computed by the algorithm will have four solved forms. The problem stems from the fact that neither of the existential quantifiers is permutable as long as the universal quantifier is still in the same subgraph; but the two configurations in which 2 dominates 3 are equivalent.

Runtime analysis. Given a graph G with n nodes and m edges, we can compute a table which specifies for each pair u, v of root nodes whether there is a unique hole of u from which v can be reached via a simple hypernormal path which doesn't use u, and which hole this is. A naive algorithm for doing this iterates over all u and v and then performs a depth-first search through G, which takes time $O(n^2(n+m))$, which is a negligible runtime in practice.

Given this table, we can determine the possible dominators of each fragment in time $O(n)$ (because there are at most $O(n)$ possible dominators). Thus it takes time $O(n)$ to decide whether a split is permutable, and time $O(n \cdot S)$, where S is the number of splits in the chart, to run the entire elimination algorithm. The reference counting optimisation adds nothing to this asymptotic runtime, as each split may trigger at most one reference count update for each hole of the split's root fragment.

5 Conclusion

We have presented an algorithm for redundancy elimination on underspecified chart representations. It checks for each subgraph in the chart whether it has a *permutable split*; if yes, it removes all other splits for this subgraph. This reduces the set of described readings, while making sure that at least one representative of each original equivalence class remains while maintaining underspecification. Equivalence is defined with respect to a certain class of rewriting systems which approximates semantic equivalence of the described formulas and fits well with the underspecification setting. The algorithm runs in polynomial time in the size of the chart.

The algorithm is useful in practice: it reduces the USRs for (1) and (2) from the introduction to one and two solved forms, respectively. In fact, initial experiments with the Rondane treebank suggest that it reduces the number of readings of a typical sentence by an order of magnitude. It does this efficiently: Even on USRs with billions of readings, for which the enumeration of readings would take about a year, it finishes after a few seconds. However, the algorithm is not complete in the sense that the computed chart has no more equivalent readings. We have some ideas for achieving this kind of completeness, which we will explore in future work. Another line in which the present work could be extended is to allow equivalence with respect to arbitrary rewrite systems.

References

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

[2] Blackburn, P. and J. Bos, “Representation and Inference for Natural Language. A First Course in Computational Semantics,” CSLI Publications, 2005.

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

[4] Chaves, R. P., *Non-redundant scope disambiguation in underspecified semantics*, in: *Proceedings of the 8th ESSLLI Student Session*, Vienna, 2003, pp. 47–58.

[5] Copestake, A. and D. Flickinger, *An open-source grammar development environment and broad-coverage english grammar using HPSG*, in: *Proc. of LREC*, 2000.

[6] Copestake, A., D. Flickinger, C. Pollard and I. Sag, *Minimal recursion semantics: An introduction.*, Journal of Language and Computation (2004), to appear.

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

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

[9] Koller, A., J. Niehren and S. Thater, *Bridging the gap between underspecification formalisms: Hole semantics as dominance constraints*, in: *Proc. of EACL-03*, 2003.

[10] Koller, A. and S. Thater, *The evolution of dominance constraint solvers*, in: *Proc. of ACL-05 Workshop on Software*, Ann Arbor, 2005.

[11] Niehren, J. and S. Thater, *Bridging the gap between underspecification formalisms: Minimal recursion semantics as dominance constraints*, in: *Proc. of ACL-03*, 2003.

[12] van Deemter, K. and S. Peters, “Semantic Ambiguity and Underspecification,” CSLI, 1996.

[13] Vestre, E., *An algorithm for generating non-redundant quantifier scopings*, in: *Proc. of EACL*, Berlin, 1991, pp. 251–256.