An Efficient Enumeration Algorithm for Canonical Form Underspecified Semantic Representations

Mehdi Manshadi, James Allen, Mary Swift
University of Rochester
Rochester, NY
{mehdih, james, swift}@cs.rochester.edu

Abstract. We give polynomial-time algorithms for satisfiability and enumeration of underspecified semantic representations in a *canonical form*. This canonical form brings several underspecification formalisms together into a uniform framework (Manshadi et al., 2008), so the algorithms can be applied to any underspecified representation that can be converted to this form. In particular, our algorithm can be applied to Canonical Form Minimal Recursion Semantics (CF-MRS). An efficient satisfiability and enumeration algorithm has been found for a subset of MRS (Niehren and Thater, 2003). This subset, however, is not broad enough to cover all the meaningful MRS structures occurring in practice. CF-MRS, on the other hand, provably covers all MRS structures generated by the MRS semantic composition process.

Keywords: Formal Semantics, Underspecification, Minimal Recursion Semantics, Canonical Form Underspecified Representation.

1 Introduction

Underspecification in semantic representation is about encoding semantic ambiguities in a semantic representation. Efficient enumeration of all possible readings of an underspecified semantic representation is a topic that has interested researchers since the introduction of underspecification in semantic representation. Hobbs and Shieber (1987) is one of the earliest works on this topic. The underspecification formalism that they use is based on a traditional underspecified logical form (Woods 1978), which is neither flat nor constraint-based. Most of the recent semantic formalisms, however, use a flat, constraint-based representation of natural language semantics, such as *Minimal Recursion Semantics* (Copestake et al., 2001), *Hole Semantics* (Bos 1996), and *Dominance Constraints* (Egg et al., 2001).

Recently there has been some work on finding efficient algorithms for determining whether an underspecified representation has a reading or not (the *satisfiability* problem) and for enumerating all the possible readings of a satisfiable representation (the *enumeration* problem). Althaus et al. (2003) shows that the satisfiability problem for Dominance Constraints formalism in its general form is NP-complete. Niehren and Thater (2003) define a subset of dominance constraints called *dominance nets*, and show that an algorithm given by Bodirsky et al. (2004) can be used to generate the readings of a dominance net. Furthermore, they define a translation of *Minimal*

Recursion Semantics (MRS) to dominance constraints. As an analogy to dominance nets, they also define a subset of MRS called *MRS nets* and prove that there is a bijection between the readings of a MRS net and the readings of its corresponding dominance net. This shows that the above mentioned algorithm can be used for enumeration of MRS nets, a big subset of MRS. They do not, however, make any claim about the coverage of MRS nets. By studying the output of *the LinGO English Resource Grammar (ERG)* (Copestake and Flickinger 2000) on the *Redwoods* Treebank (Oepen et al., 2002), Fuchss et al. (2004) claim that all the non-net MRS structures are semantically "incomplete". In other words, they claim that the concept of net is broad enough to cover all semantically complete MRS structures. This claim, however, later was invalidated (Thater 2007). That is there are examples of coherent English sentences whose MRS structure is not a net (see section 6). As a result no efficient enumeration algorithm has been found that covers all MRS structures occurring in practice.

In recent work, Manshadi et al. (2008) define another subset of MRS structures called *Canonical Form* MRS (*CF-MRS*) and prove that it covers all the well-formed MRS structures generated by the *MRS semantic composition algorithm* (Copestake et al. 2005). Motivated by the definition of CF-MRS, they define a *Canonical Form Underspecified Representation (CF-UR)* and claim that this representation can be translated back and forth to some other underspecification formalisms, such as Dominance Constraints and Hole Semantics.

In this paper, we give a polynomial-time algorithm for satisfiability and enumeration of CF-UR. This directly results in an efficient algorithm for solving CF-MRS. Since CR-MRS has been proved to cover every well-formed MRS generated by the MRS semantic composition process, our algorithm covers coherent non-net examples that previously proposed algorithms do not.

The structure of this paper is as follows. We give an informal introduction in to CF-UR in (2.1) and the formal definition in (2.2). We define *dependency graph* (3) and present the algorithms to solve a *first-order* CF-UR (4) and the CF-UR in its general form (5). (6) discusses the related work in detail; it specifically addresses the difference between MRS net and CF-MRS.

2 Canonical Form Underspecified Representation

2.1 An informal introduction

We first explain the concept of CF-UR through an example. Consider the sentence *Every dog probably chases some cat*. Two of its readings are shown in figure (1), and the corresponding CF-UR in graphical form is shown in figure (2).

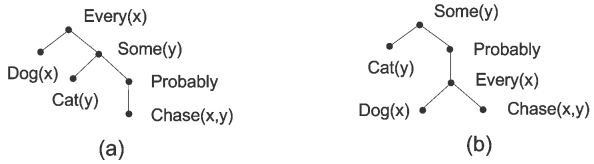

Figure 1. Two readings of the sentence Every dog probably chases some cat.

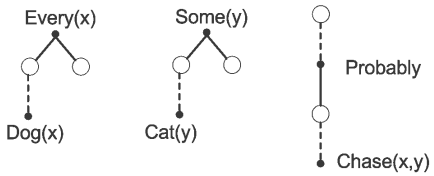

Figure 2. CF-UR graph

As shown in this figure, the graph has two types of node: *label node* and *hole node,* and two types of edge: *solid edge* and *dotted edge*. The graph is directed, although the directions are not shown explicitly when the graph is a tree or a forest. There are three kind of label nodes: first-order predicates (such as *Dog(x)*, *Chase(x,y)*), operators (such as *Probably*) and quantifiers (such as *Every(x)*). Every quantifier node has two outgoing solid edges to two hole nodes: one for its restriction (*restriction hole*) and one for its body (*body hole*). Operators such as *Probably* also have one or more outgoing solid edges to distinct hole nodes. There is only one hole node in the graph which does not have any incoming edge. This is called the *top hole*. As seen in figure (2), the number of hole nodes and label nodes in the graph are equal. In order to build the readings of a CF-UR, we must plug the label nodes into the hole nodes. But not every plugging[2] is desirable. The dotted edges in a CF-UR represent the *qeq* (equality modulo quantifiers, Copestake et al., 2005) *constraints*. A qeq constraint from hole *h* to label *l* is satisfied, if either the label *l* directly plugs into the hole *h* or, *h* is filled by a quantifier node and *l* plugs into the body hole of that quantifier, or there is another quantifier which plugs into the body hole of the first one and *l* plugs into the body of the second one and so on. Given a CF-UR, a tree is built by removing all the constraint edges and plugging every label to a distinct hole. The tree is called a fully scoped structure iff it satisfies all the qeq constraints. For example, figures (1a,b) both satisfy the four constraints of the CF-UR in figure (2); hence they are fully scoped structures of this CF-UR.

To have a valid reading of an underspecified representation every variable must be in the scope of its quantifier. We call this *dependency constraint*. A node *P(...)* is *dependent* on a quantifier node *Q(x)* if *x* is an argument of *P(...)*. A fully scoped structure satisfies this constraint iff *Q(x)* outscopes[3] (i.e. is an ancestor of) *P(...)* in the tree. A fully scoped structure of a CF-UR is called a *reading* or a *solution* iff it satisfies all the dependency constraints. Manshadi et al. (2008) prove that every well-formed MRS structure generated by the MRS's semantic composition process is in a canonical form (hence called CF-MRS), which is a notational variant of CF-UR. Furthermore, they show that in a CF-MRS, qeq relationships and outscoping constraints are equivalent; that is if the qeq constraints in a CF-UR are treated as outscoping constraints, the CF-UR will still have the same set of readings. As mentioned before, the dependency constraints are also outscoping relations; therefore similar to the Dominance Constraints formalism, the dependency constraints are made explicit (figure 3), and all the constraints in CF-UR are treated as outscoping relations.

---

[2] We borrowed the term plugging from Hole Semantics. Manshadi et al. (2008) call the plugging a label assignment.

[3] Note that outscoping (or dominance) is considered to be a reflexive and transitive relation.

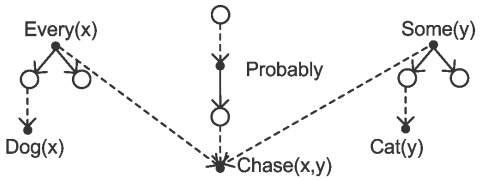

Figure 3. UR graph with explicit dependencies

2.2 The formal definition

Consider F the set of labeled formulas of the following types:

- *Quantification:* A formula of the form *l:Q(x, hr, hb)* where *l* is the label, *Q* is the generalized quantifier, *x* is the first order variable quantified by *Q*, and *hr* and *hb* are the holes for the *restriction* and the *body* of the quantifier.
- *Operators:* A formula of the form *l:P(x1, x2, ..., h1, h2, ...)* where *P* is an operator, *x1, x2, ...* are first order arguments of *P* and *h1, h2, ...* are holes for the higher order arguments.
- *Predications:* A formula of the form *l:P(x1, x2, ...)* where *P* is a first order predicate and *x1, x2, ...* are its arguments.

A CF-UR is the triple $U = <F, h_T, C>$ in which F is a set of labeled formulas, h_T is a unique hole which does not occur in any argument position in F, called the *top hole*, and $C=C_q \cup C_d$, where C_q is a set of hole to label constraints (corresponding to qeq relationships) and C_d is a set of label to label constraints (corresponding to dependency constraints). We require U to satisfy following conditions (the canonical form conditions):

a) No quantifier labels and no quantifier body holes are involved in any constraint in C_q.

b) Every other hole and label is involved in exactly one constraint in C_q.

A *label assignment* or *plugging P* is a *bijection* between holes and labels. The ordered pair $<U, P>$ is called a *reading* or a *solution* of U iff it satisfies all the constraints in $C= C_q \cup C_d$.

$<U, P>$ satisfies an outscoping constraint $u \leq v$ iff

- when u and v are both labels: $u=P(..., h, ...)$ is in F and $h \leq v$ recursively holds.
- when u is a hole and v is a label: $P(u)=v$ or $P(u)=l$, where $l=P(..., h, \ldots)$ is a labeled formula in F and $h \leq v$ recursively holds.

As an example, the CF-UR for the sentence *Every dog probably chases some cat* is shown below.

$U = <\{l1: Every(x, h1, h2), l2: Dog(x), l3: Some(y, h3, h4), l4: Cat(y), l5: Probably(h5), l6:Chase(x,y)\}, \mathbf{h0}, \{h0 \leq l5, h1 \leq l2, h3 \leq l4, h5 \leq l6\} \cup \{l6 \leq l1, l6 \leq l3\}>$

Figure (3) shows a graphical representation of the CF-UR U, in which the dotted edges represent the constraints (the labels of the formulas are not shown in this figure). Note that the hole nodes of every formula are assumed to be ordered from left to right. A plugging P, which satisfies all the constraints in U, is given below.

$P = \{(h0, l3), (h1, l2), (h2, l6), (h3, l4), (h4, l5), (h5, l1)\}$

P corresponds to the graphical representation of the solution <*U,P*> in figure (1b) above, which is obtained by removing all the dotted edges from *U*'s graph in figure (2) and merging every hole node *h* with the label node *P(h)*.

A CF-UR is called *satisfiable* iff it has at least one solution. The problem of finding all possible solutions of a CF-UR is called the *enumeration* problem.

Since dependency constraints are obvious from the first order arguments of the formulas, for the sake of readability, we often remove the dependency constraints from CF-UR's graph as in figure (2) above. The graph in figure (2) is a forest of three trees; two of which are rooted at the two quantifiers and one rooted at the top hole. Using canonical form conditions (conditions (a) and (b) given above), it is easy to see that this configuration holds in general; that is given a CF-UR *U* with *n* quantifier nodes, if we ignore the dependency constraints (i.e. C_d), *U*'s graph is a forest of exactly *n+1* trees whose roots are the top hole and the quantifiers as in figure (4).

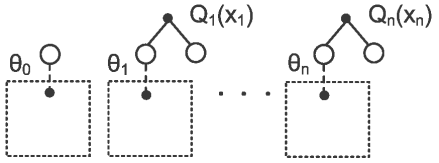

Figure 4. General structure of CF-UR

3 Dependency Graph

In this section we build a mathematical framework to formally present the algorithms and prove their properties. We will first solve the satisfiability and enumeration problems for a CF-UR with only quantifiers and first order predicates (i.e. no operators). We call such a CF-UR a *first-order* CF-UR (figure 5).

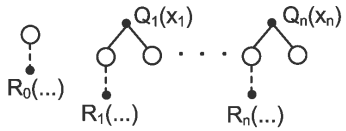

Figure 5. General structure of first-order CF-UR

To generalize this to handle an arbitrary CF-UR *U* with operators (figure 4), we transform *U* into a first-order CF-UR *U'*, which we call *U*'s *reduced form*, by collapsing the trees θ_i *(i=0..n)* into a single first order predicate $R_i(...)$, whose arguments are the union of the first order arguments of all the predicates and operators in the tree θ_i (figure 6). We use the algorithms for first-order CF-UR (given in section 4) to solve *U'* and use the results to solve the original problem (section 5).

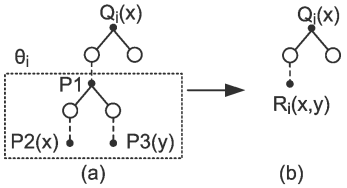

Figure 6. Collapsing the trees into a single node

Consider a first order CF-UR with *n* quantifiers (figure 5). In order to build all its corresponding fully scoped structures, all we need to know is the number of quantifiers. For example, a first order CF-UR with two quantifiers has four possible fully scoped structures shown in figure (7).

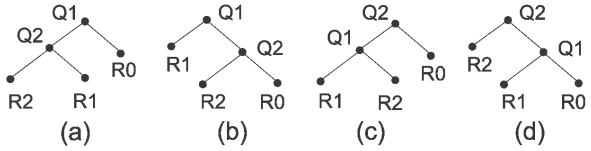

Figure 7. All fully scoped structures for n =2

In order to check whether each fully scoped structure is a solution or not, we only need to check the satisfaction of the dependency constraints. We define the concept of *dependency graph* to represent all these dependencies in a compact form. A *Dependency Graph* (*DG*) is a directed graph with *n+1* nodes labeled *0...n*, corresponding to the nodes R_0 to R_n . The node *i (i>0)* is connected to node *j* by a directed edge *(i, j)* iff R_j is dependent on Q_i. In all the examples in this paper, we assume that the quantifiers are numbered in the order they appear in the sentence. As an example consider the CF-UR in figure (8) for the sentence *Every politician whom somebody knows a child of runs*. The DG for this sentence is given in figure (9c).

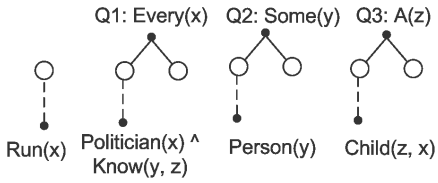

Figure 8. CF-UR for Every politician whom somebody knows a child of runs.

Figures (9a,b) show the DGs for the sentences *Every dog chases a cat* and *Every dog in a room barks* respectively. Note that out of the four possible fully scoped structures for n=2, given in figure (7), (7b,d) are the solutions for the DG in (9a) and (7a,d) are the solutions for DG in (9b).

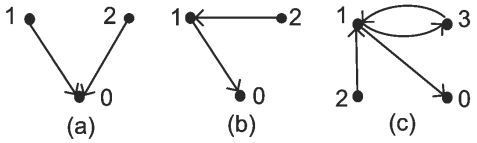

Figure 9. Examples of dependency graph

From the definition, node 0 in every DG has no outgoing edges, so it is a *sink* node. We call the sink node the *heart* of the DG. In all the DGs in figure (9), every node can reach the heart by a directed path. We call this property (that every node in a DG is connected to the heart by a directed path) *heart-connectedness.* Therefore all three DGs in figure (9) are *heart-connected*. This is not a coincidence. Note that if a node is directly connected to the heart by a directed edge, it means that the corresponding noun phrase (NP) is an argument of the heart formula (i.e. the main predicate of the sentence). If a node is connected to the heart by a path of length two, it means that the

corresponding NP is a modifier of an argument of the heart formula, and so on. The heart-connectedness property requires that every NP contribute to the meaning of the sentence, either by filling an argument position of the main predicate or by modifying an argument of the main predicate, and so on; which is a trivial property of every coherent sentence.

4 Scoping in first order CF-UR

Consider a first order CF-UR U with the heart-connected DG G. A solution of U (or simply a solution of G) is an ordered binary tree T with exactly n interior nodes labeled $Q_1..Q_n$ and $n+1$ leaf nodes labeled $R_0...R_n$ such that

- *Qeq constraints:* R_0 is the right-most leaf of T. Every other R_i is the right-most leaf of the tree rooted at the left child of Q_i in T.
- *Dependency constraints:* for every *i, j (i>0)*, if node *i immediately dominates* node *j* in G, then Q_i dominates R_j in T.

Here, by *u immediately dominates v*, we mean *u* is connected to *v* by an edge *(u,v)*. *Dominates* is the *reflexive transitive* closure of *immediately dominates*. We represent the tree rooted at the left child of Q_i in T, T_i and call it the *restriction tree* of Q_i. Similarly, the tree rooted at the right child of Q_i is called the *body tree* of Q_i

For example, the DG given in figure (11a) for the sentence *Every dog in a room of some house barks* has 5 different solutions, two of them given in figure (11b,c).

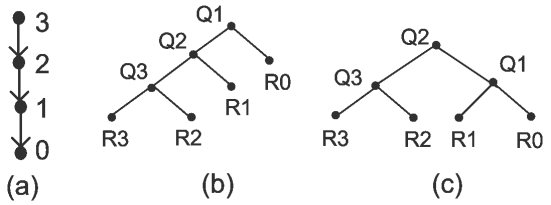

Figure 11. A DG with two of its solutions

It is important to understand some counter intuitive properties of DG. First, the fact that *i* immediately dominates *j* in G, although implying Q_i has to dominate R_j in T, does not necessarily mean that Q_i has to dominate Q_j in T. This is shown in both figures (11b,c) for the nodes *Q2* and *Q3*. Second, if *i transitively* (i.e. not immediately or reflexively) dominates *j* in G, it does not force Q_i to dominate R_j in T, as shown for nodes *Q3* and *R1* in both figures (11b,c).

4.1 Satisfiability algorithm

If the DG G has no *directed cycle* (hence is a *directed acyclic graph* or *DAG*), then there is a *topological order* of the nodes in G, say *n, n-1, ..., 0*. In this case, figure (12) is an interpretation of G. Hence G is trivially satisfiable.

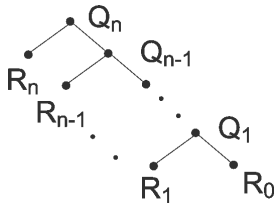

Figure 12. An interpretation for a DAG

Now consider the DG G in figure (9c) as an example of a DG with directed cycles. We can partition G into 3 *Strongly Connected Components* (*SCC*) G_0, G_1, G_2 as shown in figure (13a). A node in an SCC, which connects it to some node outside the SCC, is called a *head* of SCC. In this example, every SCC (except G_0) has exactly one head. We replace every SCC (except G_0) with its head as shown in figure (13b) and call the new graph G'. G' is a DAG, so we can build an interpretation T' of G' as shown in figure (13c). Since we replaced G_1 with a single node, Q_3 is missing in T'. We remove all the outgoing edges of the head (node 1) in G_1 and call the new graph G'_1 as shown in figure (13d). If we treat node 1 as the heart, G'_1 can be seen as a heart-connected DG. Therefore we can recursively apply the same procedure to G'_1 to build a solution T_1 shown in figure (13e). When we return from this recursive procedure, we replace the node R_1 in T' with the tree T_1 and call the new tree T (figure 13f). T is a solution of G.

Note that if G_1 has more than one head, that is if node 3 is also directly connected to the heart, then T would no longer be a solution of G, as R_0 is not in the scope of Q_3 in T. This suggests rejecting every DG containing an SCC with more than one head.

To state this idea formally, consider a DG $G=(V, E)$, with $K+1$ SCC $G_0 \ldots G_K$ where G_0 only contains the heart node. We formally define a head of $G_k=(V_k, E_k)$ as a node u in V_k such that there exists a node v in $V-V_k$ where (u,v) is an edge in E. Since G is heart-connected, every G_k $(k>0)$ has at least one head. We call a DG G *single-headed* iff every SCC G_k has at most one head. Let i_k $(k>0)$ be the head of G_k. We remove all the outgoing edges of i_k in G_k to call the resulting graph G'_k. If we treat i_k as the heart, G'_k can be seen as an independent DG, called the *underlying DG* of G_k.

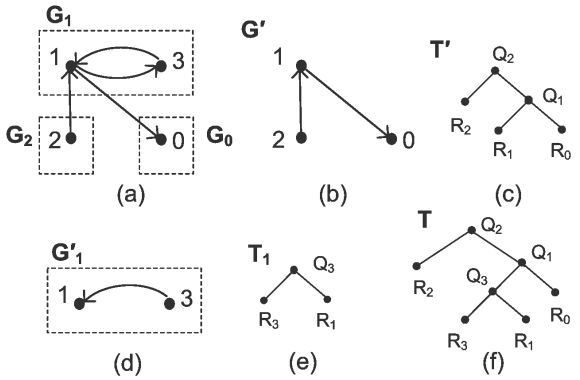

Figure 13. Satisfiability of a DG with cycles.

SAT(G)

1. Find all the SCCs in G ($G_0 \ldots G_K$).
2. If some SCC has more than one head output UNSAT and halt, otherwise:
3. Replace each SCC G_k (k>0) with its head i_k and call the new graph G′.
4. Find a topological order of the nodes in G′, say i_K .. i_1; build an interpretation T′ of G′ as shown in figure (15a).
5. For each SCC G_k with more than one node, remove all the outgoing edges of the head and call it G'_k. Let T_k = SAT(G'_k)
6. Replace every Ri_k in T′ with T_k; call the new tree T as shown in figure (15b).
7. Return T

Figure 14. SATisfiability algorithm for first-order CF-UR

A heart-connected DG *G* is called *recursively single-headed* iff

i) *G* is a single SCC, or
ii) *G* is single-headed and the *underlying* DG of all its SCCs are recursively single-headed.

Note that if a single-headed *G* is heart-connected, all the underlying DGs of *G* are also heart-connected; therefore the concept of recursive single-headedness is well-defined.

Theorem 1. A heart-connected DG is satisfiable if and only if it is recursively single-headed.

Note that theorem 1 has an intuitive linguistic explanation. Since an SCC in the DG of a sentence represents a noun phrase and the head of an SCC actually represents the head of the noun phrase, this theorem says that an underspecified semantic representation has a reading if and only if every noun phrase has a single head.

The algorithm in figure (14) generalizes the procedure introduced in the above example. It returns a tree *T* if *G* is recursively single-headed and outputs UNSAT otherwise. To prove the *if* direction of theorem 1, all we need is to show that if the algorithm returns a tree *T*, *T* is a solution of *G*. We prove this using induction on the depth of the recursion *d*. For *d=0*, *G* is a DAG; hence *T* is trivially a solution of G. Consider a DG *G* for which the depth of recursion is *d (d>0)* and let *T* be the output of the above algorithm. *T* is a solution of *G* since:

- *Qeq constraints: R0* is trivially the right-most leaf of *T*. Also, for every *k>0*, using the induction assumption, T_k is a solution of G'_k with the heart i_k; therefore Ri_k is the right-most leaf of T_k. All other qeq constraints hold by induction assumption.

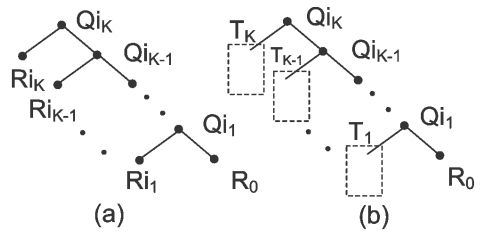

Figure 15. Graphical description of the algorithm

- *Dependency constraints:* consider the nodes *i, j* such that *i* immediately dominates *j* in *G*. If *i, j* are in the same SCC, say G_k, then this dependency constraint holds by the induction assumption (because T_k is a solution of G'_k). If *i, j* are in two different SCCs, say G_k and G_l respectively, then we have $i=i_k$ (because i_k is the single head of G_k) and $i_k<i_l$ (because i_K, i_{K-1}, … is a topological order); therefore Qi_k dominates Qi_l and the whole tree T_l *in T* (figure 15b) and since R_j is in T_l , Q_i dominates R_j in T.

The proof of the *only if* direction is not as straightforward. The proof idea is given below by stating three helpful lemmas. For a given node *v* in *G*, we define *Anc(v)* as the set of nodes that dominate *v* (including *v* itself) and *Dis(v)* as the set of nodes that have a path to the heart without going through *v* (including the heart itself). As *G* is heart-connected, for every *v* in *V*, we have $Anc(v) \cup Dis(v) = V$.

Lemma 1: If Q_i is the root of some arbitrary solution *T* of *G*, then for every $j \neq i$ in *Anc(i)*, Q_j is in the restriction tree of Q_i and for every $k>0$ in *Dis(i)*, Q_k is in the body tree of Q_i.

Proof: Consider $j \in Anc(i)$. We use induction on the length of the path *P* (shown as $|P|$) from *j* to *i* to show that Q_j is in the restriction of Q_i . First, let $|P|=1$, that is *j* immediately dominates *i*. Q_i is the root and R_i (the restriction of Q_i) has to be in the scope of Q_j, therefore Q_j has to be in the restriction tree of Q_i. Now Let $|P|=n+1$ $(n \geq 1)$ and *k* be the node immediately after *j* on the path *P*. According to the induction assumption, Q_k is in the restriction tree of Q_i. On the other hand, R_k must be in the scope of Q_j therefore Q_j has to be in the restriction tree of Q_i too. A similar argument applies when $j \in Dis(i)$. □

From this lemma, a node Q_i can be the root of some solution *G* only if *Anc(i)* and *Dis(i)* are disjoint. We call this property the *root condition*.

Lemma 2. If a subgraph G' of *G* (with the same heart) is unsatisfiable, then *G* is unsatisfiable.

Although simple, the above lemma is very helpful as it allows us to consider only the problematic part of the DG and ignore the rest. Now consider a DG *G* with an SCC G_1 which has at least two heads, say i_1 and j_1 (figure 16). The nodes i_1 and j_1 are connected to some node(s) outside SCC, say *i* and *j* respectively. As *G* is heart-connected, *i* and *j* are connected to the heart by paths P_1 and P_2. First, consider the case where the two paths intersect only at the heart node. Consider only the part of *G* which includes the SCC G_1 and the paths P_1 and P_2; call it G' (figure 16).

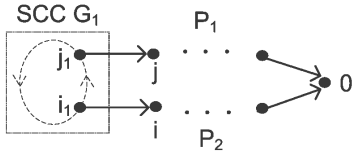

Figure 16. An unsatisfiable DG

G' is unsatisfiable because none of the nodes in this graph satisfy the root condition. Therefore from lemma (2), *G* is unsatisfiable. A similar argument can be given for the case where P_1 and P_2 intersect at some other nodes as well. This completes the proof that *G* is satisfiable only if it is single-headed. To prove that *G* needs to be recursively single-headed, we use following lemma, which directly results from lemma 1.

Lemma 3. If i_k is the head of some SCC G_k and T is an arbitrary solution of G, for every node $j \neq i_k$ in G_k, Q_j is in the restriction tree of Qi_k in T.

From this lemma, the nodes in every SCC G_k form a smaller satisfiability problem whose DG *contains* the underlying DG of G_k (i.e. G'_k) and whose heart is the same as G'_k's heart (i.e. Ri_k). Therefore from lemma 2 the property of being single-headed must recursively hold for G'_k.

The algorithm given in figure (14) divides the satisfiability of G into K subproblems, satisfiability of $G'_1 \ldots G'_K$, where $|V'_1|+\ldots+|V'_K|<|V|$ ($|G|$ is the size of G and |V| is the number of nodes in G). The cost of this breaking is linear in |G|. Therefore if $T(|G|)$ is the running time of the algorithm, we have:

$$T(|G|) = O(|G|)+T(|G'_1|)+\ldots+T(|G'_K|)$$

Using induction on |V|, the worst-case complexity of the algorithm is quadratic in size of G. More precisely the running time is $O(|V|\cdot|G|)$, where $|G| = |V|+|E|$ if we represent the graph using adjacency list.

4.2 Enumeration algorithm

Using lemma 1, we showed that Q_i can be a root of some solution T of G only if *Anc(i)* and *Dis(i)* are disjoint. Consider the subgraph of G induced by the nodes in *Dis(i)* and call it G_b (figure 17 is an example where $i=1$).

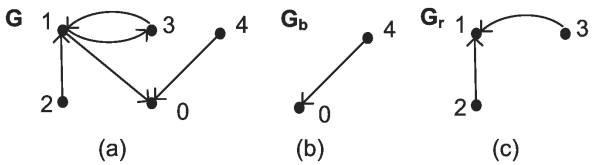

Figure 17. G, G_r, and G_b for Every politician whom somebody had a chat with voted for the bill.

From lemma 1, it can be seen that the body tree of Q_i in T has to be a solution of G_b. Now, consider the subgraph of G induced by the nodes in *Anc(i)*. We remove all the outgoing edges of i and call the resulting DG, G_r (figure 17c). From lemma 1, the restriction tree of Q_i in T has to be a solution of G_r. As a result, Q_i is a root of some solution of G if and only if it satisfies the root condition and its corresponding G_r and G_b are both satisfiable. Note that since G is heart-connected, both G_r and G_b are heart-connected. Figure (18) summarizes the enumeration algorithm.

EnumerateFO (G)

1. *If G is not satisfiable fail.*
2. *If G has only one node, return G.*
3. *Find R, the set of nodes i in G which satisfy the root condition.*
4. *Non-deterministically pick a node i in R:*
5. *Build G_r and G_b*
6. *Let T_r = EnumerateFO(G_r) & T_b = EnumerateFO(G_b)*
7. *Build the tree T rooted at Q_i, with T_r and T_b as restriction and body tree of Q_i respectively.*
8. *Return T*

Figure 18. Enumeration for first-order CF-UR

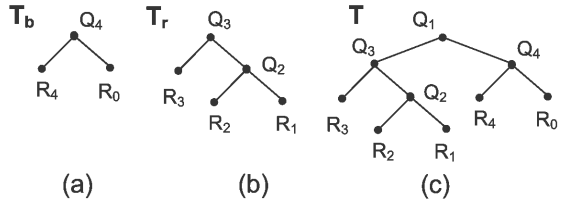

Figure 19. An example of enumeration algorithm

As an example, if the recursive call to *EnumerateFO()* for G_r and G_b in figure (17), returns the trees T_r and T_b in figure (19a,b), the final solution tree T would be the one in figure (19c). An argument similar to the proof of the *if* direction of theorem 1 can be given to prove the soundness of the algoithm. The completeness can be prove by induction. If T is a solution of a heart-connected DG G rooted at Q, according to the above discssions and lemma 1, Q must satisfy the root condition (hence it will be picked as the root at some branch of the algorithm). On the other hand, the restriction and the body tree of Q must be solutions of G_r and G_b (hence they are built at step 5 based on induction asumption), therefore T will be generated by the algorithm.

The enumeration algorithm breaks the problem into subproblems, but this time the cost of this breaking is quadratic in $|G|$ (because we check the satisfiablity at each step). Therefore the time complexity of the overall procedure is cubic in size of G per *solution*. The time-complexity can be improved though. It can be shown that the satisfiability check at each step is not necessary. In fact, we can remove step 1 of the algorithm and if the enumeration algorithm ever fails (i.e. it encounters an empty R) we declare G as unsatisfiable. After this simplification, the running time would be $O(|V|\cdot|G|)$. Space does not permit us to give the technical details of the proofs given here. We reserve those for a longer paper.

5 Scoping with operators

The following theorem shows that to check the satisfiability of a CF-UR, it is enough to check the satisfiability of its reduced form.

Theorem 2. A CF-UR is satisfiable if and only if its reduced form is satisfiable.

The *only if* direction is trivial. The *if* direction is true because for every solution of the reduced form, at least one solution of the original CF-UR can be built by taking the solution T of the reduced form, expanding every node R_i to its original tree θ_i (see figures 4-6) and simply assigning every label in the θ_i to the hole to which the label is qeq. We call such solutions *basic* solutions. For example, figure (20a) shows the CF-UR for the sentence *Every dog which probably chases some cat does not bark* with its reduced form in figure (20b). Figures (20c,d,e) show one solution of the reduced form, its expanded version, and the corresponding basic solution of the original MRS respectively. In general, corresponding to each solution of the reduced form, there is more than one solution of the original CF-UR. For example there are three more solutions corresponding to (20c) as shown in (20f,g,h).

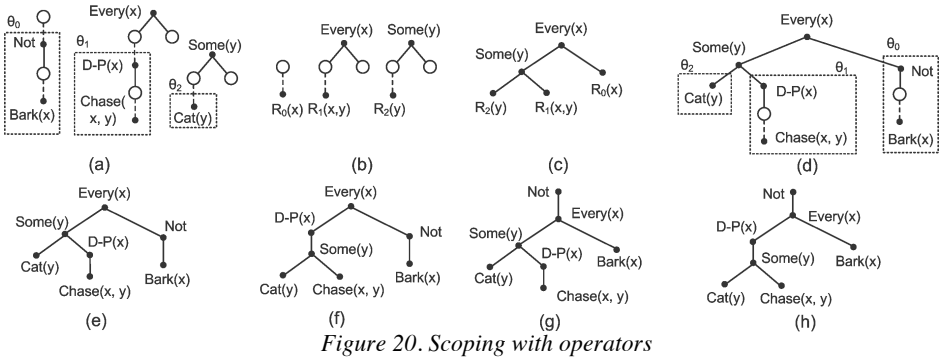

Figure 20. Scoping with operators

The solutions other than basic ones are built by taking a basic solution and moving quantifiers with their restriction trees inside operators (figure 21).

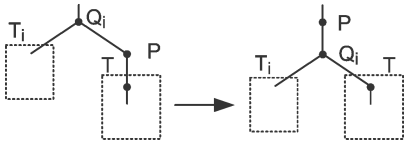

Figure 21. Moving quantifiers inside operators

A quantifier Q_i can move inside an operator P only if P is not dependent on Q_i. The enumeration algorithm for a general CF-UR is shown in figure (22). For every basic solution T', *EnumerateB(T′, n)* is called to build all the corresponding non-basic solutions. In order to prevent generating a single solution in more than one way, quantifiers in a basic solution are ordered in a post order fashion and are picked by *EnumerateB()* based on this order using argument m (see condition (a) in step 2 of *EnumerateB()*). Trivially the algorithm is sound. To see why it is complete, consider an arbitrary solution T (e.g. figure 20h); move quantifiers with their restriction trees (based on a preorder) all the way up in T until it hits another quantifier node; the resulting tree T' would be a basic solution (figure 20e).

Enumerate(U) *// U: a CF-UR with n quantifiers*

1. *Let G = DG of U′ (Reduced-form of U)*
2. *Let T = EnumerateFO(G)*
   T′ = Basic solution corresponding to T
3. *Call EnumerateB(T′, n)*

EnumerateB(T, m)

1. *Output T*
2. *Non-deterministically pick a quantifier Q_i;*
   a) whose order k is at most m;
   b) whose body node is an operator P;
   c) where P is not dependent on Q_i
3. *Move Q_i inside P and call the new tree T′.*
4. *Call EnumerateB(T′, k)*

Figure 22. General enumeration algorithm

Every branch of *EnumerateB()* takes linear time and *uniquely* generates a solution; therefore *EnumerateB()* runs in linear time per solution; therefore *Enumerate()* runs in quadratic time per solution as a result of the call to *EnumerateFO()*.

In these algorithms, we only considered single-hole operators. The extension to general case is straightforward; if an operator *P* has more than one hole, when moving a quantifier inside *P*, we non-deterministically pick a child of *P* and move the quantifier inside *P* along that child. We define an order on the children of *P* (e.g. from left to right) to prevent generating a single tree in exponentially many ways.

6 Related Work

There has been some work on satisfiability and enumeration of underspecified representation in the context of dominance constraints (Althaus 2003, Bodirsky 2004). Crucially, the concept of *solution* in that context has a different definition from the standard definition of reading in formal semantics, which we gave in section 2. In dominance constraints formalism, the standard notion of reading is referred to as the *constructive solution* or *configuration*. In the following discussion, we will use the term DC *solution* to refer to dominance constraints notion of solution and the term *reading* or *constructive solution* to refer to the standard notion of solution.

The main difference between a DC solution and a constructive solution is that there could be nodes in the DC solution that do not correspond to any label in the actual underspecified representation, but in a constructive solution every node corresponds to some label in the underspecified representation. As a result there are examples of underspecified representations that have DC solutions but no constructive solution. In fact, the problem of finding the DC solutions is easier than the problem of finding constructive solutions. However, even finding DC solutions for dominance constraints in general is NP-complete.

As a result, Althaus et al. (2003) define a subset of dominance constraints called *normal* dominance constraints and show that this subset can be solved in polynomial time. Bodirsky et al. (2004) expand the definition of normal dominance constraints to *weakly* normal dominance constraints and show that this larger subset still can be solved in polynomial-time. However, finding the constructive solutions of both normal and weakly normal dominance constraint is still NP-complete. This means that Bodirsky's algorithm cannot be used to find constructive solutions of weakly normal dominance constraints. Niehren and Thater (2003) define the concept of dominance net, a subset of weakly normal dominance constraints, and show that for this subset, Bodirsky's algorithm can be used to enumerate the constructive solutions.

As an analogy with dominance nets, they define a subset of MRS called MRS nets and show that Bodirsky's algorithm can be used to enumerate its readings. The concept of net, however, is too restricted. Figure (23) shows the three schemas that can occur in a net. In this figure dependency constraints are shown explicitly and all the constraint edges are interpreted as outscoping relations.

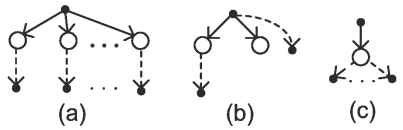

Figure 23. Net schemas

The first schema corresponds to the operators in CF-UR where every hole has exactly one outgoing constraint. The second schema corresponds to the quantifier nodes where the restriction has one outgoing constraint edge and the body has no outgoing edge. The net condition requires that the quantifier node has exactly one outgoing dependency edge, which means there must be exactly one predicate (other than quantifier's restriction) dependent on every quantifier. This is where the limitation of nets comes from: not every natural language sentence satisfies this restriction. For example consider the CF-UR in figure (8). Figure (24) shows the same CF-UR with explicit dependency constraints.

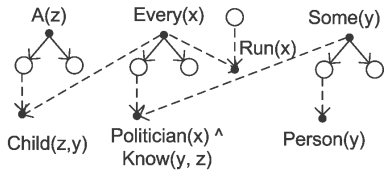

Figure 24. A non-net CF-UR

As seen in this figure, the quantifier *Every* has two outgoing dependency edges, therefore it is not a net. Note that in CF-UR, there is no restriction on the number of outgoing dependency edges of a quantifier, therefore CF-UR covers the non-net structures such as the above example.

The schema in figure (23c), on the other hand, does not have any counterpart in CF-UR. This means that CF-UR lacks a certain kind of structure covered by nets. However, within the context of practical MRS structures this is not a limitation for CF-UR. In fact, this schema only occurs when translating MRS structures into dominance constraints. Since the dominance constraints formalism does not allow a free hole, the top hole of an MRS is replaced with a dummy operator *Prop* with a single hole. To enforce that this predicate has the widest scope (i.e. be the root of every reading), the hole is connected to every other label by a dominance edge (cf. Thater 2007). This results in one structure of schema 3 in every MRS net. No such a transformation is needed when MRS structures are represented in canonical form, therefore such a structure never occurs in CF-MRS. As a result, within the domain of practical MRS structures, nets are a strict subset of canonical form structures. After all, CF-MRS is proved to cover all well-formed MRS structures generated by the MRS semantic composition process.

7 Conclusion

We have presented algorithms for satisfiability and enumeration of CF-UR, an underspecified semantic representation in a canonical form. CF-UR is a notational variant of CF-MRS, which is the set of all well-formed MRS structures that can be generated by the MRS semantic composition algorithm. CF-MRS is broader than MRS nets, a previously defined subset of MRS for which satisfiability and enumeration algorithms have been found; broad enough to cover all the MRS structures occurring in practice.

In addition, CF-UR brings several different formalisms together into a uniform framework. Therefore, the proposed algorithms can be applied to any underspecified representation that can be transformed into CF-UR. For example, by using the concept of dependency graph, it is straightforward to show that the enumeration algorithm given here can replace the traditional wrapping algorithm (Woods 1987) to generate all the readings of a logical form.

The main drawback with both CF-UR and nets is that they do not allow holes to have more than one constraint edge, while some semantic constraints such as island constraints require additional outscoping constraints on the restriction hole of quantifiers. Presenting a version of the algorithms for this extended underspecified representation remains future work.

References

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

Bos, J. (1996) Predicate logic unplugged. In Proc. 10th Amsterdam Colloquium, pages 133–143.

Bodirsky M., Duchier D., Niehren J., and Miele S.. 2004. An efficient algorithm for weakly normal dominance constraints. In ACM-SIAM Symposium on Discrete Algorithms. The ACM Press.

Copestake, A. and Flickinger, D. (2000) An open-source grammar development environment and broad-coverage English grammar using HPSG. In Conference on Language Resources and Evaluations.

Copestake, A., Lascarides, A. and Flickinger, D. (2001) An Algebra for Semantic Construction in Constraint-Based Grammars. ACL-01. Toulouse, France.

Copestake A., Flickinger D., Pollard, C., and Sag, I. (2005) Minimal Recursion Semantics: An Introduction. Research on Language and Computation, 3 (4):281-332.

Egg M., Koller A., and Niehren J. (2001) The constraint language for lambda structures. Journal of Logic, Language, and Information, 10:457–485.

Fuchss, R., Koller, A., Niehren, J. and Thater, S. (2004) Minimal Recursion Semantics as Dominance Constraints: Translation, Evaluation, and Analysis. ACL-04. Barcelona, Spain.

Hobbs, J. and Shieber, S. M. (1987) An Algorithm for Generating Quantifier Scopings. Computational Linguistics 13, pp. 47–63.

Manshadi, M., Allen J., and Swift, M. (2008) Toward a Universal Underspecifed Semantic Representation. 13th Conference on Formal Grammar (FG 2008), Hamburg, Germany.

Niehren, J. and Thater, S. (2003) Bridging the Gap Between Underspecification Formalisms: Minimal Recursion Semantics as Dominance Constraints. ACL-03. Sapporo, Japan.

Oepen, S., Callahan, E. Flickinger, D., and Manning, C. (2002) LinGO Redwoods. A Rich and Dynamic Treebank for HPSG. In Beyond PARSEVAL, LREC 2002, Las Palmas, Spain.

Thater, S. (2007) Minimal Recursion Semantics as Dominance Constraints: Graph-Theoretic Foundation and Application to Grammar Engineering. PhD Thesis, Universität des SaarlandesWoods,

W. A. (1978). "Semantics and Quantification in Natural Language Questiom Answering." Advances in Computers 17: 1-87.