Efficient Construction of Underspecified Semantics under Massive Ambiguity

Jochen Dörre*
Institut für maschinelle Sprachverarbeitung
University of Stuttgart

Abstract

We investigate the problem of determining a compact underspecified semantical representation for sentences that may be highly ambiguous. Due to combinatorial explosion, the naive method of building semantics for the different syntactic readings independently is prohibitive. We present a method that takes as input a syntactic parse forest with associated constraint-based semantic construction rules and directly builds a *packed semantic structure*. The algorithm is fully implemented and runs in $O(n^4 log(n))$ in sentence length, if the grammar meets some reasonable 'normality' restrictions.

1 Background

One of the most central problems that any NL system must face is the ubiquitous phenomenon of ambiguity. In the last few years a whole new branch developed in semantics that investigates *underspecified* semantic representations in order to cope with this phenomenon. Such representations do not stand for the real or intended meaning of sentences, but rather for the possible options of interpretation. Quantifier scope ambiguities are a semantic variety of ambiguity that is handled especially well by this approach. Pioneering work in that direction has been (Alshawi 92) and (Reyle 93).

More recently there has been growing interest in developing the underspecification approach to also cover syntactic ambiguities (cf. (Pinkal 95; EggLebeth 95; Schiehlen 96)). Schiehlen's approach is outstanding in that he fully takes into account syntactic constraints. In (Schiehlen 96) he presents an algorithm which directly constructs a single underspecified semantic structure from the ideal "underspecified" syntactic structure, a parse forest.

On the other hand, a method for producing "packed semantic structures", in that case "packed quasi-logical forms", has already been used in the Core Language Engine, informally described in (Alshawi 92, Chap. 7). However, this method only produces a structure that is virtually isomorphic to the parse forest, since it simply replaces parse forest nodes by their corresponding semantic operators. No attempt is made to actually apply semantic operators in the phase where those "packed QLFs" are constructed. Moreover, the packing of the QLFs seems to serve no purpose in the processing phases following semantic analysis. Already the immediately succeeding phase "sortal filtering" requires QLFs to be unpacked, i.e. enumerated.

Contrary to the CLE method, Schiehlen's method actively packs semantic structures, even when they result from distinct syntactic structures, extracting common parts. His method, however, may take time exponential w.r.t. sentence length. Already the semantic representations it produces can be exponentially large, because they grow linear with the number of (syntactic) readings and that can be exponential, e.g., for sentences that exhibit the well-known attachment ambiguity of prepositional phrases. It is therefore an interesting question to ask, whether we can compute compact semantic representations from parse forests without falling prey to exponential explosion.

The purpose of the present paper is to show that construction of compact semantic representations like in Schiehlen's approach from parse forests is not only possible, but also cheap, i.e., can be done in polynomial time.

To illustrate our method we use a simple DCG grammar for PP-attachment ambiguities, adapted from (Schiehlen 96), that yields semantic representations (called UDRSs) according to the Underspecified Discourse Representation Theory (Reyle 93; KampReyle 93). The grammar is shown in Fig. 1.

*This research has been carried out while the author visited the Programming Systems Lab of Prof. Gert Smolka at the University of Saarland, Saarbrücken. Thanks to John Maxwell, Martin Müller, Joachim Niehren, Michael Schiehlen, and an anonymous reviewer for valuable feedback and to all at PS Lab for their helpful support with the OZ system.

```
start(DRS) --> s([_,_,ltop],[],DRS).

s([Event,VerbL,DomL],DRS_i,DRS_o) -->
  np([X,VerbL,DomL],DRS_i,DRS1),
  vp([Event,X,VerbL,DomL],DRS1,DRS_o).

s([Event,VerbL,DomL],DRS_i,DRS_o) -->
  s([Event,VerbL,DomL],DRS_i,DRS1),
  pp([Event,VerbL,DomL],DRS1,DRS_o).

vp([Ev,X,VerbL,DomL],DRS_i,DRS_o) -->
  vt([Ev,X,Y,VerbL,DomL],DRS_i,DRS1),
  np([Y,VerbL,DomL],DRS1,DRS_o).

np([X,VbL,DomL],DRS_i,DRS_o) -->
  det([X,NounL,VbL,DomL],DRS_i,DRS1),
  n([X,NounL,DomL],DRS1,DRS_o).

n([X,NounL,DomL],DRS_i,DRS_o) -->
  n([X,NounL,DomL],DRS_i,DRS1),
  pp([X,NounL,DomL],DRS1,DRS_o).

pp([X,L,DomL],DRS_i,DRS_o) -->
  prep(Cond,X,Y),
  np([Y,L,DomL],[L:Cond|DRS_i],DRS_o).
```

```
vt([Ev,X,Y,L,_DomL],DRS_i,DRS) --> [saw],
  {DRS=[L:see(Ev,X,Y)|DRS_i]}.

det([X,Lab,VerbL,_],DRS_i,DRS) -->
  [a],
  {DRS=[lt(Lab,ltop),lt(VerbL,Lab),
        Lab:X|DRS_i],
   gensym(l,Lab),gensym(x,X)}.

det([X,ResL,VbL,DomL],DRS_i,DRS) -->
  [every],
  {DRS=[lt(L,DomL),lt(VbL,ScpL),ResL:X,
        L:every(ResL,ScpL)|DRS_i],
   gensym(l,L),gensym(l,ResL),
   gensym(l,ScpL),gensym(x,X)}.

np([X,_,_],DRS_i,DRS) --> [i],
  {DRS=[ltop:X,anchor(X,speaker)|DRS_i],
   gensym(x,X)}.

n([X,L,_],DRS,[L:man(X)|DRS]) --> [man].
n([X,L,_],DRS,[L:hill(X)|DRS]) --> [hill].

prep(on(X,Y),X,Y) --> [on].
prep(with(X,Y),X,Y) --> [with].
```

Figure 1: Example DCG

The UDRSs constructed by the grammar are flat lists of the UDRS-constraints $l \leq l'$ (subordination (partial) ordering between labels; Prolog representation: `lt`(l,l')), l : *Cond* (condition introduction in subUDRS labeled l), $l : X$ (referent introduction in l), l : *GenQuant*(l',l'') (generalised quantifier) and an anchoring function. The meaning of a UDRS as a set of denoted DRSs can be explained as follows.[1] All conditions with the same label form a subUDRS and labels occurring in subUDRSs denote locations (holes) where other subUDRSs can be plugged into. The whole UDRS denotes the set of well-formed DRSs that can be formed by some plugging of the subUDRSs that does not violate the ordering $<$. Scope of quantifiers can be underspecified in UDRSs, because subordination can be left partial.

In our example grammar every nonterminal has three arguments. The 2nd and the 3rd argument represent a UDRS list as a difference list, i.e., the UDRS is "threaded through". The first argument is a list of objects occurring in the UDRS that play a specific role in syntactic combinations of the current node.[2] An example of a UDRS, however a packed UDRS, is shown later on in §5.

To avoid the dependence on a particular grammar formalism we present our method for a constraint-based grammar abstractly from the actual constraint system employed. We only require that semantic rules relate the semantic 'objects' or structures that are associated with the nodes of a local tree by employing constraints. E.g., we can view the DCG rule $s \longrightarrow np\ vp$ as a relation between three 'semantic construction terms' or variables **SemS**, **SemNP**, **SemVP** equivalent to the constraints

```
SemS = [[Event,VerbL,DomL,TopL],DRS_i,DRS_o]
SemNP = [[X,VerbL,DomL,TopL],DRS_i,DRS1]
SemVP = [[Event,X,VerbL,DomL,TopL],DRS1,DRS_o]
```

Here is an overview of the paper. §2 gives the preliminaries and assumptions needed to precisely state the problem we want to solve. §3 presents the abstract algorithm. Complexity considerations follow in §4. Finally, we consider implementation issues, present results of an experiment in §5, and close with a discussion.

2 The Problem

As mentioned already, we aim at calculating from given parse forests the same compact semantic structures that have been proposed by (Schiehlen 96), i.e. structures that make explicit the common parts of different syntactic readings, so that subsequent semantic processes can use this generalised information. As he does, we assume a constraint-based grammar, e.g. a DCG (PereiraWarren 80) or HPSG (PollardSag 94) , in which syntactic constraints and constraints that determine a resulting semantic representation can be seperated and parsing can be performed using the syntactic constraints only.

Second, we assume that the set of syntax trees can be compactly represented as a parse forest (cf. (Earley 70; BillotLang 89; Tomita 86)). Parse forests are rooted labeled directed acyclic graphs with AND-nodes (standing for context-free branch-

[1]Readers unfamiliar with DRT should think of these structures as some Prolog terms, representing semantics, built by unifications according to the semantic rules. It is only important to notice how we extract common parts of those structures, irrespective of the structures' meanings.

[2]E.g., for an NP its referent, as well as the upper and lower label for the current clause and the top label.

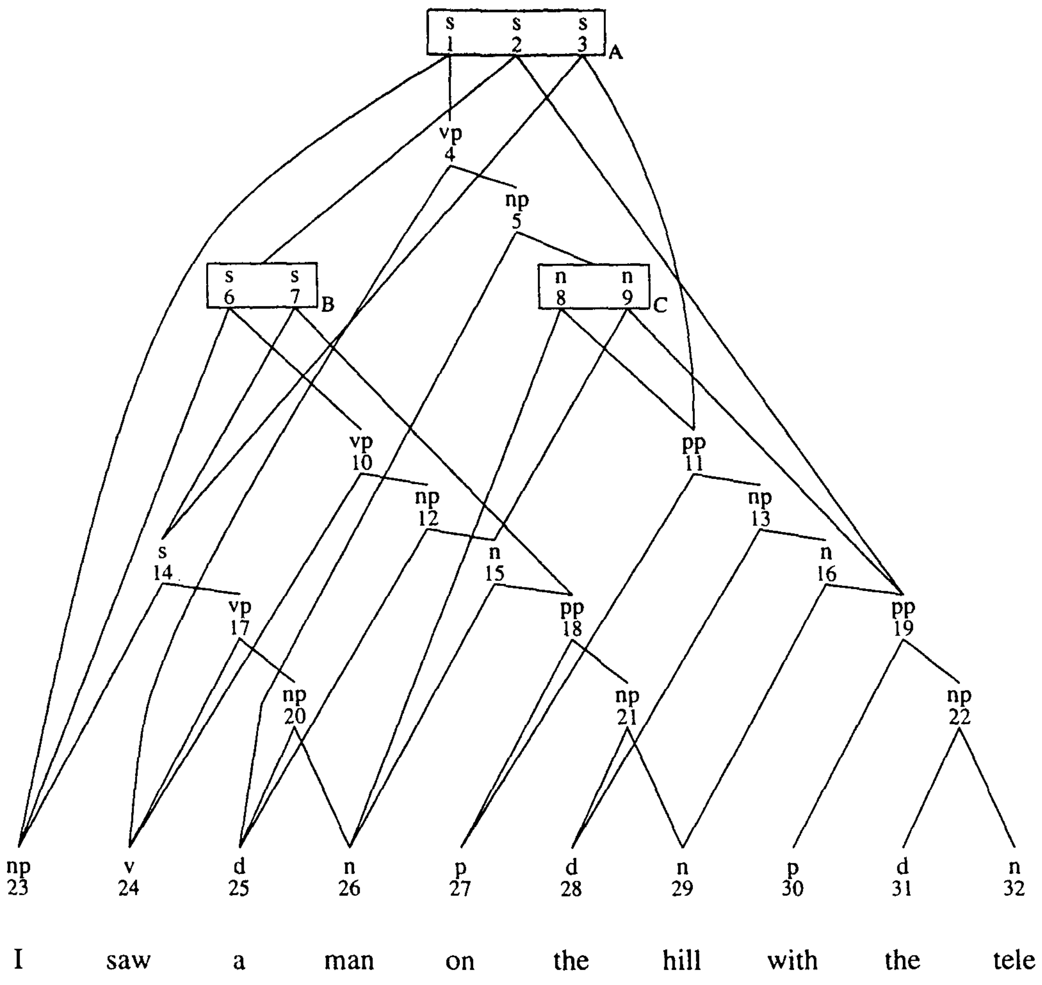


Figure 2: Example of a parse forest

ing) and OR-nodes (standing for alternative subtrees), that can be characterised as follows (cf. Fig. 2 for an example).[3]

1. The terminal yield as well as the label of two AND-nodes are identical, if and only if they both are children of one OR-node.
2. Every tree reading is a valid parse tree.

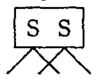
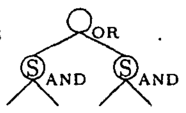
Tree readings of such graphs are obtained by replacing any OR-node by one of its children. Parse forests can represent an exponential number of phrase structure alternatives in $o(n^3)$ space, where n is the length of the sentence. The example uses the 3 OR-nodes (A, B, C) and the AND-nodes 1 through 32 to represent 5 complete parse trees, that would use 5×19 nodes.

Third, we assume the rule-to-rule hypothesis, i.e., that the grammar associates with each local tree a 'semantic rule' that specifies how to construct the mother node's semantics from those of its children.

Hence, input to the algorithm is

- a parse forest
- an associated semantic rule for every local tree (AND-node together with its children) therein
- and a semantic representation for each leaf (coming from a semantic lexicon).

To be more precise, we assume a constraint language $\mathcal{C}$ over a denumerable set of variables $\mathcal{X}$, that is a sublanguage of Predicate Logic with equality and is closed under conjunction, disjunction, and variable renaming. Small greek letters ϕ, ψ will henceforth denote constraints (open formulae) and letters X, Y, Z (possibly with indeces) will denote variables. Writing $\phi(X_1, \ldots, X_k)$ shall indicate that $X_1, \ldots, X_k$ are the free variables in the constraint ϕ. Frequently used examples for constraint languages are the language of equations over first-order terms

[3]The graphical representation of an OR-node is a box surrounding its children, i.e. the AND-OR-graph structure of  is .

for DCGs,[4] PATR-style feature-path equations, or typed feature structure description languages (like the constraint languages of ALE (Carpenter 92) or CUF (DörreDorna 93)) for HPSG-style grammars.

Together with the constraint language we require a constraint solver, that checks constraints for satisfiability, usually by transforming them into a normal form (also called 'solved form'). Constraint solving in the DCG case is simply unification of terms.

The semantic representations mentioned before are actually not given directly, but rather as a constraint on some variable, thus allowing for partiality in the structural description. To that end we assume that every node in the parse forest ν has associated with it a variable X_ν that is used for constraining the (partial) semantic structure of ν. The semantics of a leaf node μ is hence given as a constraint $\phi_\mu(X_\mu)$, called a *leaf constraint.*

A final assumption that we adopt concerns the nature of the 'semantic rules'. The process of semantics construction shall be a completely monotonous process of gathering constraints that *never leads to failure.* We assume that any associated (instantiated) semantic rule $r(\nu)$ of a local tree (AND-branching) $\nu(\nu_1,\ldots,\nu_k)$ determines ν's semantics $\Sigma(\nu)$ as follows from those of its children:

$$\Sigma(\nu) = \exists X_{\nu_1}\ldots\exists X_{\nu_k}(\phi_{r(\nu)}(X_\nu, X_{\nu_1},\ldots,X_{\nu_k}) \wedge \Sigma(\nu_1) \wedge \ldots \wedge \Sigma(\nu_k)).$$

The constraint $\phi_{r(\nu)}(X_\nu, X_{\nu_1},\ldots,X_{\nu_k})$ is called the *rule constraint* for ν. It is required to only depend on the variables $X_\nu, X_{\nu_1},\ldots,X_{\nu_k}$. Note that if the same rule is to be applied at another node, we have a different rule constraint.

Note that any $\Sigma(\nu)$ depends only on X_ν and can be thought of as a unary predicate. Now, let us consider semantics construction for a single parse tree for the moment. The leaf constraints together with the rules define a semantics constraint $\Sigma(\nu)$ for every node ν, and the semantics of the full sentence is described by the Σ-constraint of the root node, $\Sigma(root)$. In the Σ-constraints, we actually can suppress the existential quantifiers by adopting the convention that any variable other than the one of the current node is implicitly existentially bound on the formula toplevel. Name conflicts, that would force variable renaming, cannot occur. Therefore $\Sigma(root)$ is (equivalent to) just a big conjunction of all rule constraints for the inner nodes and all leaf constraints.

Moving to parse forests, the semantics of an OR-node $\nu(\nu_1,\ldots,\nu_k)$ is to be defined as

$$\Sigma(\nu) = \exists X_{\nu_1}\ldots\exists X_{\nu_k}(\Sigma(\nu_1) \wedge X_\nu{=}X_{\nu_1} \vee \ldots \vee \Sigma(\nu_k) \wedge X_\nu{=}X_{\nu_k}),$$

specifying that the set of possible (partial) semantic representations for ν is the union of those of ν's children. However, we can simplify this formula once and for all by assuming that for every OR-node there is only one variable X_ν that is associated with it *and all of its children.* Using the same variable for $\nu_1\ldots\nu_k$ is unproblematic, because no two of these nodes can ever occur in a tree reading. Hence, the definition we get is

$$\Sigma(\nu) = \Sigma(\nu_1) \vee \ldots \vee \Sigma(\nu_k).$$

Now, in the same way as in the single-tree case, we can directly "read off" the Σ-constraint for the whole parse forest representing the semantics of all readings. Although this constraint is only half the way to the packed semantic representation we are aiming at, it is nevertheless worthwhile to consider its structure a little more closely. Fig. 3 shows the structure of the Σ-constraint for the OR-node B in the example parse forest.

In a way the structure of this constraint directly mirrors the structure of the parse forest. However, by writing out the constraint, we loose the sharings present in the forest. A subformula coming from a shared subtree (as $\Sigma(18)$ in Fig. 3) has to be stated as many times as the subtree appears in an unfolding of the forest graph. In our PP-attachment example the blowup caused by this is in fact exponential.

On the other hand, looking at a Σ-constraint as a piece of syntax, we can *represent* this piece of syntax in the same manner in which trees are represented in the parse forest, i.e. we can have a representation of $\Sigma(root)$ with a structure isomorphic to the forest's graph structure.[5] In practice this difference becomes a question of whether we have full control over the representations the constraint solver employs (or any other process that receives this constraint as input). If not, we cannot contend ourselves with the *possibility* of compact representation of constraints, but rather need a means to enforce this compactness on the constraint level. This means that we have to introduce some form of functional abstraction into the constraint language (or anything equivalent that allows giving names to complex constraints and referencing to them via their names). Therefore we enhance the constraint language as follows. We allow to our disposition a second set of variables, called names, and two special forms of constraints

1. `def(<name>, <constraint>)`	name definition
2. `<name>`	name use

with the requirements, that a name may only be used, if it is defined and that its definition is unique. Thus, the constraint $\Sigma(B)$ above can be written as

$$\begin{aligned}&(\phi_{r(6)} \wedge \ldots \wedge \phi_{26} \wedge \mathrm{N}\\ &\ \vee \phi_{r(7)} \wedge \ldots \wedge \phi_{26} \wedge \mathrm{N}\ )\\ &\ \wedge\ \mathtt{def}(\mathrm{N},\ \phi_{r(18)} \wedge \phi_{27} \wedge \phi_{r(21)} \wedge \phi_{28} \wedge \phi_{29})\end{aligned}$$

[4]DCG shall refer in this paper to a logically pure version, Definite Clause Grammars based on pure PROLOG, involving no nonlogical devices like Cut, var/1, etc.

[5]The packed QLFs in the Core Language Engine (Alshawi 92) are an example of such a representation.

$$\underbrace{\phi_{r(6)} \wedge \phi_{23} \wedge \phi_{r(10)} \wedge \phi_{24} \wedge \phi_{r(12)} \wedge \phi_{25} \wedge \phi_{r(15)} \wedge \phi_{26} \wedge \underbrace{\phi_{r(18)} \wedge \phi_{27} \wedge \phi_{r(21)} \wedge \phi_{28} \wedge \phi_{29}}_{\Sigma(18)}}_{\Sigma(6)}$$

$$\vee$$

$$\underbrace{\phi_{r(7)} \wedge \phi_{r(14)} \wedge \phi_{23} \wedge \phi_{r(17)} \wedge \phi_{24} \wedge \phi_{r(20)} \wedge \phi_{25} \wedge \phi_{26} \wedge \underbrace{\phi_{r(18)} \wedge \phi_{27} \wedge \phi_{r(21)} \wedge \phi_{28} \wedge \phi_{29}}_{\Sigma(18)}}_{\Sigma(7)}$$

Figure 3: Constraint $\Sigma(B)$ of example parse forest

The packed semantic representation as constructed by the method described so far still calls for an obvious improvement. Very often the different branches of disjunctions contain constraints that have large parts in common. However, although these overlaps are efficiently handled on the representational level, they are invisible at the *logical level.* Hence, what we need is an algorithm that factores out common parts of the constraints on the logical level, pushing disjunctions down.[6] There are two routes that we can take to do this efficiently.

In the first we consider only the structure of the parse forest, however ignore the content of (rule or leaf) constraints. I.e. we explore the fact that the parts of the Σ-constraints in a disjunction that stem from nodes shared by all disjuncts must be identical, and hence can be factored out.[7] More precisely, we can compute for every node ν the set $\texttt{must-occur}(\nu)$ of nodes (transitively) dominated by ν that must occur in a tree of the forest, whenever ν occurs. We can then use this information, when building the disjunction $\Sigma(\nu)$ to factor out the constraints introduced by nodes in $\texttt{must-occur}(\nu)$, i.e., we build the factor $\Phi = \bigwedge_{\nu' \in \texttt{must-occur}(\nu)} \Sigma(\nu')$ and a 'remainder' constraint $\Sigma(\nu_i)\backslash\Phi$ for each disjunct.

The other route goes one step further and takes into account the content of rule and leaf constraints. For it we need an operation `generalise` that can be characterised informally as follows.

> For two satisfiable constraints ϕ and ψ, $\texttt{generalise}(\phi, \psi)$ yields the triple ξ, ϕ', ψ', such that ξ contains the 'common part' of ϕ and ψ and ϕ' represents the 'remainder' $\phi\backslash\xi$ and likewise ψ' represents $\psi\backslash\xi$.

The exact definition of what the 'common part' or the 'remainder' shall be, naturally depends on the actual constraint system chosen. For our purpose it is sufficient to require the following properties:

> If $\texttt{generalise}(\phi, \psi) \mapsto (\xi, \phi', \psi')$, then $\phi \vdash \xi$ and $\psi \vdash \xi$ and $\phi \equiv \xi \wedge \phi'$ and $\psi \equiv \xi \wedge \psi'$.

We shall call such a generalisation operation *simplifying* if the normal form of ξ is not larger than any of the input constraints' normal form.

Example: An example for such a generalisation operation for PROLOG's constraint system (equations over first-order terms) is the so-called anti-unify operation, the dual of unification, that some PROLOG implementations provide as a library predicate.[8] Two terms $T1$ and $T2$ 'anti-unify' to T, iff T is the (unique) most specific term that subsumes both $T1$ and $T2$. The 'remainder constraints' in this case are the residual substitutions σ_1 and σ_2 that transform T into $T1$ or $T2$, respectively.

Let us now state the method informally. We use `generalise` to factor out the common parts of disjunctions. This is, however, not as trivial as it might appear at first sight. `Generalise` should operate on solved forms, but when we try to eliminate the names introduced for subtree constraints in order to solve the corresponding constraints, we end up with constraints that are exponential in size. In the following section we describe an algorithm that circumvents this problem.

3 The Algorithm

We call an order $\leq$ on the nodes of a directed acyclic graph $G = (N, E)$ with nodes N and edges E ***bottom-up***, iff whenever $(i, j) \in E$ ("i is a predecessor to j"), then $j \leq i$.

For the sake of simplicity let us assume that any nonterminal node in the parse forest is binary branching. Furthermore, we leave implicit, when conjunctions of constraints are normalised by the constraint solver. Recall that for the generalisation operation it is usually meaningful to operate on

[6]Actually, in the $\Sigma(B)$ example such a factoring makes the use of the name **N** superfluous. In general, however, use of names is actually necessary to avoid exponentially large constraints. Subtrees may be shared by quite different parts of the structure, not only by disjuncts of the same disjunction. In the PP-attachment example, a compression of the Σ-constraint to polynomial size cannot be achieved with factoring alone.

[7](Maxwell IIIKaplan 93) exploit the same idea for efficiently solving the functional constraints that an LFG grammar associates with a parse forest.

[8]`anti_unify` in Quintus Prolog , `term_subsumer` in Sicstus Prolog.

```
Input:  • parse forest, leaf and rule constraints as described above
        • array of variables $X_\nu$ indexed by node s.t. if $\nu$ is a child of OR-node $\nu'$, then $X_\nu = X_{\nu'}$
Data structures:  • an array SEM of constraints and an array D of names, both indexed by node
                  • a stack ENV of def constraints
Output: a constraint representing a packed semantic representation
Method: ENV := nil
        process nodes in a bottom-up order
        doing with node $\nu$:
        if $\nu$ is a leaf then
            SEM[$\nu$] := $\phi_\nu$
            D[$\nu$] := true
        elseif $\nu$ is AND($\nu_1, \nu_2$) then
            SEM[$\nu$] := $\phi_{r(\nu)} \wedge$ SEM[$\nu_1$] $\wedge$ SEM[$\nu_2$]
            if D[$\nu_1$] = true then D[$\nu$] := D[$\nu_2$]
            elseif D[$\nu_2$] = true then D[$\nu$] := D[$\nu_1$]
            else D[$\nu$] := newname
                push def(D[$\nu$], D[$\nu_1$] $\wedge$ D[$\nu_2$]) onto ENV
            end
        elseif $\nu$ is OR($\nu_1, \nu_2$) then
            let GEN, REM1, REM2 such that
                generalise(SEM[$\nu_1$], SEM[$\nu_2$]) $\mapsto$ (GEN, REM1, REM2)
            SEM[$\nu$] := GEN
            D[$\nu$] := newname
            push def(D[$\nu$], REM1 $\wedge$ $D[\nu_1]$ $\vee$ REM2 $\wedge$ $D[\nu_2]$) onto ENV
        end return SEM[$root$] $\wedge$ D[$root$] $\wedge$ ENV
```

Figure 4: Packed Semantics Construction Algorithm

solved forms. However, at least the simplifications $\mathbf{true} \wedge \phi \equiv \phi$ and $\phi \wedge \mathbf{true} \equiv \phi$ should be assumed.

The Packed Semantics Construction Algorithm is given in Fig. 4. It enforces the following invariants, which can easily be shown by induction.

1. Every name used has a unique definition.
2. For any node ν we have the equivalence $\Sigma(\nu) \equiv$ SEM[ν] $\wedge$ $[\![$D[ν]$]\!]$, where $[\![$D[ν]$]\!]$ shall denote the constraint obtained from D[ν] when recursively replacing names by the constraints they are bound to in ENV.
3. For any node ν the constraint SEM[ν] is never larger than the Σ-constraint of any single tree in the forest originating in ν.

Hence, the returned constraint correctly represents the semantic representation for all readings.

4 Complexity

The complexity of this abstract algorithm depends primarily on the actual constraint system and generalisation operation employed. But note also that the influence of the actual semantic operations prescribed by the grammar can be vast, even for the simplest constraint systems. E.g., we can write a DCGs that produce abnormal large "semantic structures" of sizes growing exponentially with sentence length (for a single reading). For meaningful grammars we expect this size function to be linear. Therefore, let us abstract away this size by employing a function $f_G(n)$ that bounds the size of semantic structures (respectively the size of its describing constraint system in normal form) that grammar G assigns to sentences of length n.

Finally, we want to assume that generalisation is simplifying and can be performed within a bound of $g(m)$ steps, where m is the total size of the input constraint systems.

With these assumptions in place, the time complexity for the algorithm can be estimated to be (n = sentence length, N = number of forest nodes)

$$O(g(f_G(n)) \cdot N) \leq O(g(f_G(n)) \cdot n^3),$$

since every program step other than the generalisation operation can be done in constant time per node. Observe that because of Invariant 3. the input constraints to generalise are bounded by f_G as any constraint in SEM.

In the case of a DCG the generalisation operation is `anti_unify`, which can be performed in $o(n \cdot log(n))$ time and space (for acyclic structures). Hence, together with the assumption that the semantic structures the DCG computes can be bounded linearly in sentence length (and are acyclic), we obtain a $O(n \cdot log(n) \cdot N) \leq O(n^4 log(n))$ total time complexity.

SEM[top]:

```
[ltop : x1,
 anchor(x1,'Speaker'),
 l1 : see(e1,x1,x2),
 lt(l2,ltop),
 lt(l1,l2),
 l2 : x2,
 l2 : man(x2),
 A : on(B,x3),
 lt(l3,ltop),
 lt(A,l5),
 l4 : x3,
 l3 : every(l4,l5),
 l4 : hill(x3),
 C : with(D,x4),
 lt(l6,ltop),
 lt(C,l6),
 l6 : x4,
 l6 : tele(x4)]
```

D[top] (a Prolog goal):

```
dEnv(509,1,[B,A,D,C]))
```

ENV (as Prolog predicates):

```
dEnv(506, 1, A) :-
        (   A=[e1,l1]
        ;   A=[x2,l2]
        ).
dEnv(339, 1, A) :-
        (   A=[C,B,C,B]
        ;   A=[x3,l4,_,_]
        ).
dEnv(509, 2, A) :-
        (   A=[e1,l1,x3,l4]
        ;   A=[x2,l2,C,B],
            dEnv(339, 1, [C,B,x2,l2])
        ).
dEnv(509, 1, A) :-
        (   A=[G,F,e1,l1],
            dEnv(506, 1, [G,F])
        ;   A=[E,D,C,B],
            dEnv(509, 2, [E,D,C,B])
        ).
```

Figure 5: Packed UDRS: conjunctive part (left column) and disjunctive binding environment

5 Implementation and Experimental Results

The algorithm has been implemented for the PROLOG (or DCG) constraint system, i.e., constraints are equations over first-order terms. Two implementations have been done. One in the concurrent constraint language OZ (SmolkaTreinen 96) and one in Sicstus Prolog.[9] The following results relate to the Prolog implementation.[10]

Fig. 5 shows the resulting packed UDRS for the example forest in Fig. 2. Fig. 6 displays the SEM part as a graph. The disjunctive binding environment only encodes what the variable referents B and D (in conjunction with the corresponding labels A and C) may be bound to to: one of e1, $x2$, or $x3$ (and likewise the corresponding label). Executing the goal dEnv(509,1,[B,A,D,C]) yields the five solutions:

```
A = l1, B = e1, C = l1, D = e1 ? ;
A = l2, B = x2, C = l1, D = e1 ? ;
A = l1, B = e1, C = l4, D = x3 ? ;
A = l2, B = x2, C = l2, D = x2 ? ;
A = l2, B = x2, C = l4, D = x3 ? ;

no
| ?-
```

Table 1 gives execution times used for semantics construction of sentences of the form I saw a man (on a hill)^n for different n. The machine used for the experiment was a Sun Ultra-2 (168MHz), running Sicstus 3.0#3. In a further experiment an n-ary anti_unify operation was implemented, which improved execution times for the larger sentences, e.g., the 16 PP sentence took 750 msec. These results approximately fit the expectations from the theoretical complexity bound.

[9]The OZ implementation has the advantage that feature structure constraint solving is built-in. Our implementation actually represents the DCG terms as a feature structures. Unfortunately it is an order of magnitude slower than the Prolog version. The reason for this presumably lies in the fact that meta-logical operations the algorithm needs, like generalise and copy_term have been modeled in OZ and not on the logical level were they properly belong, namely the constraint solver.

[10]This implementation is available from http://www.ims.uni-stuttgart.de/~jochen/CBSem.

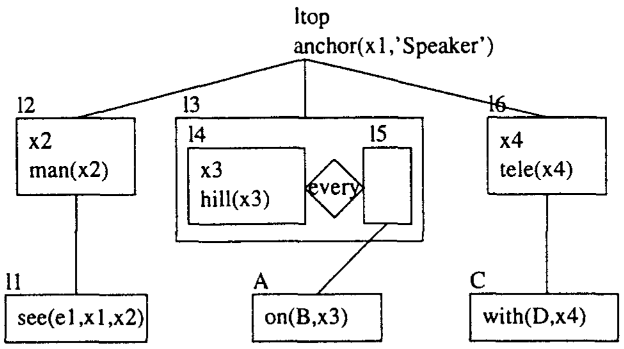


Figure 6: Conjunctive part of UDRS, graphically

n	Readings	AND- + OR-nodes	Time
2	5	35	4 msec
4	42	91	16 msec
6	429	183	48 msec
8	4862	319	114 msec
10	58786	507	220 msec
12	742900	755	430 msec
14	9694845	1071	730 msec
16	129 Mio.	1463	1140 msec

Table 1: Execution times

6 Discussion

Our algorithm and its implementation show that it is not only possible in theory, but also feasible in practice to construct packed semantical representations directly from parse forests for sentence that exhibit massive syntactic ambiguity. The algorithm is both in asymptotic complexity and in real numbers dramatically faster than an earlier approach, that also tries to provide an underspecified semantics for syntactic ambiguities. The algorithm has been presented abstractly from the actual constraint system and can be adapted to any constraint-based grammar formalism.

A critical assumption for the method has been that semantic rules never fail, i.e., no search is involved in semantics construction. This is required to guarantee that the resulting constraint is a kind of 'solved form' actually representing so-to-speak the free combination of choices it contains. Nevertheless, our method (modulo small changes to handle failure) may still prove useful, when this restriction is not fulfilled, since it focuses on computing the common information of disjunctive branches. The conjunctive part of the output constraint of the algorithm can then be seen as an approximation of the actual result, if the output constraint is satisfiable. Moreover, the disjunctive parts are reduced, so that a subsequent full-fledged search will have considerably less work than when directly trying to solve the original constraint system.

References

H. Alshawi (Ed.). *The Core Language Engine.* ACL-MIT Press Series in Natural Languages Processing. MIT Press, Cambridge, Mass., 1992.

S. Billot and B. Lang. The Structure of Shared Forests in Ambiguous Parsing. In *Proceedings of the 27th Annual Meeting of the ACL, University of British Columbia*, pp. 143–151, Vancouver, B.C., Canada, 1989.

B. Carpenter. *ALE: The Attribute Logic Engine User's Guide.* Laboratory for Computational Linguistics, Philosophy Department, Carnegie Mellon University, Pittsburgh PA 15213, December 1992.

J. Dörre and M. Dorna. CUF — A Formalism for Linguistic Knowledge Representation. In J. Dörre (Ed.), *Computational Aspects of Constraint-Based Linguistic Description I, DYANA-2 deliverable R1.2.A.* ESPRIT, Basic Research Project 6852, July 1993.

J. Earley. An Efficient Context-Free Parsing Algorithm. *Communications of the ACM*, 13(2):94–102, 1970.

M. Egg and K. Lebeth. Semantic Underspecification and Modifier Attachment Ambiguities. In J. Kilbury and R. Wiese (Eds.), *Integrative Ansätze in der Computerlinguistik. Beiträge zur 5. Fachtagung der Sektion Computerlinguistik der Deutschen Gesellschaft für Sprachwissenschaft (DGfS)*, pp. 19–24. Düsseldorf, Germany, 1995.

H. Kamp and U. Reyle. *From Discourse to Logic. Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory.* Studies in Linguistics and Philosophy 42. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1993.

J. T. Maxwell III and R. M. Kaplan. The Interface between Phrasal and Functional Constraints. *Computational Linguistics*, 19(4):571–590, 1993.

F. C. Pereira and D. H. Warren. Definite Clause Grammars for Language Analysis—A Survey of the Formalism and a Comparison with Augmented Transition Networks. *Artificial Intelligence*, 13:231–278, 1980.

M. Pinkal. Radical Underspecification. In *Proceedings of the 10th Amsterdam Colloquium*, pp. 587–606, Amsterdam, Holland, December 1995. ILLC/Department of Philosophy, University of Amsterdam.

C. Pollard and I. A. Sag. *Head Driven Phrase Structure Grammar.* University of Chicago Press, Chicago, 1994.

U. Reyle. Dealing with Ambiguities by Underspecification: Construction, Representation, and Deduction. *Journal of Semantics*, 10(2):123–179, 1993.

M. Schiehlen. Semantic Construction from Parse Forests. In *Proceedings of the 16th International Conference on Computational Linguistics*, Copenhagen, Denmark, 1996.

G. Smolka and R. Treinen (Eds.). *DFKI Oz Documentation Series.* German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, 1996. http://www.ps.uni-sb.de/oz.

M. Tomita. *Efficient Parsing for Natural Languages.* Kluwer Academic Publishers, Boston, 1986.