Relaxing Underspecified Semantic Representations for Reinterpretation

Alexander Koller (koller@coli.uni-sb.de)
Dept. of Computational Linguistics, University of the Saarland

Joachim Niehren (niehren@ps.uni-sb.de)
Programming Systems Lab, University of the Saarland

Kristina Striegnitz (kris@coli.uni-sb.de)
Dept. of Computational Linguistics, University of the Saarland

Abstract. Type and sort conflicts in semantics are usually resolved by a process of reinterpretation, which introduces an operator into the semantic representation. We elaborate on the foundations of a recent approach to reinterpretation within a framework for semantic underspecification. In this approach, relaxed underspecified semantic representations are inferred from the syntactic structure, leaving space for subsequent addition of reinterpretation operators. Unfortunately, a structural danger of overgeneration is inherent to the relaxation of underspecified semantic representations. We identify the problem and distinguish structural properties that avoid it. We furthermore develop techniques for proving these properties and apply them to prove the safety of relaxation in a prototypical syntax/semantics interface. In doing so, we present some novel properties of tree descriptions in the constraint language over lambda structures (CLLS).

Keywords: constraints, natural language semantics, reinterpretation, tree descriptions, underspecification

1. Introduction

In natural language semantics, sort or type conflicts as well as assumptions of the context a sentence is uttered in may cause meaning shifts of words or expressions (Bierwisch, 1983; Dölling, 1994; Nunberg, 1995; Pustejovsky, 1995). An example is the following sentence.

(1) Peter began a book.

The problem in this sentence is that Peter can only begin an activity. So in understanding it, we must fill in what, exactly, Peter begins to do with the book – for example, reading it, writing it, etc. This meaning shift leads to readings such as (2) or (3), in which more material has been added, as indicated by the boxes. This process of adding material is called *reinterpretation*.

(2) $\exists x.book(x) \wedge begin(peter, \boxed{\text{read(peter,}}\, x \boxed{)}).$

(3) $\exists x.\mathit{book}(x) \wedge \mathit{begin}(\mathit{peter}, \boxed{\text{write(peter,}}\, x\boxed{)})$

Pustejovsky (1995) assumed that the reinterpretation operator is determined by purely lexical information. Since then it has been noticed that the information determining the reinterpretation process comes actually from a variety of source, including context and world knowledge. This implies that the final decision on a specific reinterpretation operator can only be made after semantic construction. In an extension to Pustejovsky's approach, Lascarides and Copestake (1998) use defaults which can be overridden after semantic construction to account for this fact. Egg (2000) noticed that the actual insertion of a reinterpretation operator can be modeled in an elegant, monotonic way within a framework for semantic underspecification (van Deemter and Peters, 1996; Reyle, 1993; Bos, 1996; Pinkal, 1996; Egg et al., 1998). Instead of deriving and then destructively changing a fully specific semantic representation for a sentence, he changes his semantic construction process to produce a *relaxed* description of the sentence meaning, which contains a "gap" at the reinterpretation site. A reinterpretation operator can then be filled into the gap non-destructively. Such a description could look roughly as in (4).

(4) $\exists x.\mathit{book}(x) \wedge \mathit{begin}(\mathit{peter}, \ \ldots\ x\ \ldots)$

It is possible to derive such relaxed descriptions compositionally, and reinterpretation indeed becomes a monotonic process of making descriptions more specific. However, relaxation is in general a process that bears a danger of overgeneration. Roughly speaking, it is possible that material already present in the description could slip into the gap, giving the relaxed description unintended readings.

This article is a formal investigation of the relaxation operation in the framework of the Constraint Language for Lambda Structures (CLLS) (Egg et al., 1998), where underspecified semantic representations are expressed by tree descriptions subsuming dominance constraints. We define what it means for a relaxation to be *safe* (the overgeneration problem is avoided) and *open* (arbitrary material can be filled into the gap). Then we develop general techniques for reasoning about structural properties of tree descriptions in CLLS – most notably, *chains of fragments* – and use these techniques to prove that every relaxed description produced by the syntax/semantics interface of a certain toy grammar is open and safe. We believe that this result can be lifted to more serious grammars while using the presented proof techniques.

Plan of the article. Section 2 introduces tree descriptions in CLLS. In Section 3, we discuss Egg's underspecification approach to reinterpretation and make the overgeneration problem concrete. In Section 4, we formalize the notions of relaxation, safety, and openness. In Section 5, we define fragments and chains of fragments and prove some useful properties of these objects. Section 6 presents the prototypical toy grammar for deriving relaxed underspecified representations. Finally, we prove that all relaxations that can be derived by this interface are safe and open in Section 7.

2. Semantic Underspecification in CLLS

This section introduces the Constraint Language for Lambda Structures (CLLS) as a formalism for semantic underspecification. Briefly, the idea of underspecification is that because the number of readings of an ambiguous sentence may grows hyper-exponentially with the number of ambiguities, it can make sense to derive only a single, "underspecified" description of the meaning from the syntax and then to work with this description instead of the readings for as long as possible. Readings are enumerated only by need.

CLLS (Egg et al., 2000; Egg et al., 1998; Koller et al., 1998) can be used for the underspecified description of λ-terms. Technically, it is a language of tree descriptions based on dominance constraints, which are used in various applications throughout computational linguistics (Marcus et al., 1983; Vijay-Shanker, 1992; Rambow et al., 1995; Gardent and Webber, 1998). Dominance constraints appear, to a varying degree of explicity, in many frameworks of scope underspecification (Reyle, 1993; Bos, 1996; Muskens, 1995), and their computational aspects are rather well understood (Backofen et al., 1995; Vijay-Shanker et al., 1995; Koller et al., 1998; Duchier and Niehren, 2000; Koller et al., 2000). CLLS is interpreted over λ-*structures*, conservative extensions of tree structures. Below, we will first define λ-structures and then two different ways of describing them: by (conjunctive) logic formulas and by constraint graphs. Then we give a quick overview about the application to scope. Finally, we introduce the first-order language over CLLS, which will turn out later to be useful for talking *about* CLLS descriptions.

2.1. Lambda Structures

A λ-structure represents a λ-term uniquely, up to renaming of bound variables. It is an ordinary first-order tree structure extended with a partial λ-binding function. λ-structures can be drawn as tree-like graphs, with dashed arrows representing λ-binding, as in Figure 1. The

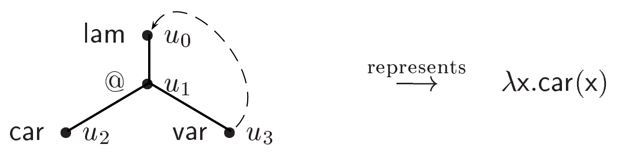


Figure 1. A λ-structure and the λ-term it represents.

tree structures we consider are based on *constructor trees*; that is, the label of a node determines the number of its children. Constructor trees are basically ground terms over a given signature; for instance, the constructor tree underlying the λ-structure in Figure 1 is lam(car@var).

We assume a *signature* $\Sigma = \{\mathsf{lam}, \mathsf{@}, \mathsf{var}, \mathsf{car}, \mathsf{driver}, \dots\}$ of *function symbols* written as f, g. Every function symbol $f \in \Sigma$ has an arity $\mathsf{ar}(f) \geq 0$. We let a, b range over constants – function symbols of arity 0. For describing λ-terms, we assume the following function symbols to belong to Σ: a unary symbol lam for λ-abstraction, the binary symbol @ for application, and the constant var for occurrences of λ-bound variables.

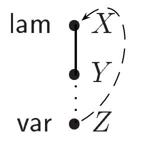
We define an (unlabeled) *tree* in the standard way as a directed graph (V, E) where V is a finite set of *nodes* u, v and $E \subseteq V \times V$ a finite set of *edges* e. The indegree of each node in V is at most 1; each tree has exactly one *root*, i.e. a node with indegree 0. We call a node with outdegree 0 a *leaf* of the tree.

A (finite) *constructor tree* over Σ is a triple (V, E, L) where (V, E) is a tree, $L : V \to \Sigma$ a *node labeling* and $L : E \to \mathbb{N}$ an *edge labeling*, such that any node $u \in V$ has exactly one outgoing edge with label k for each $1 \leq k \leq \mathsf{ar}(L(u))$, and no other outgoing edges. The symbol L is overloaded to serve both as a node and an edge labeling; there will be no danger of confusion.

Definition 2.1. *A (partial)* λ-structure τ *is a quadruple (V,E,L,λ) consisting of a constructor tree* (V, E, L) *over* Σ *and a partial function* $\lambda : L^{-1}(\mathsf{var}) \rightsquigarrow L^{-1}(\mathsf{lam})$ *mapping occurrences of lambda bound variables to their lambda binder.*

We will freely identify the λ-structure (V, E, L, λ) with a first-order structure with domain V and the following relations: binding $\lambda(u) = v$, *dominance* $\lhd^*$, and, for each function symbol $f \in \Sigma$, *labeling.* Dominance and labeling are defined for $u, v, u_1, \dots, u_n \in V$ by:

$u \lhd^* v$	iff there is a path from u to v in (V, E);
$u{:}f(u_1, \dots, u_n)$	iff $L(u) = f$ and $L(u, u_i) = i$ for $1 \leq i \leq n = \mathsf{ar}(f)$.



$$\xrightarrow{\text{represents}} \quad \begin{array}{l} X{:}\mathsf{lam}(Y) \wedge Y \lhd^* Z \wedge \\ Z{:}\mathsf{var} \wedge \lambda(Z){=}X \wedge \\ X{\neq}Z \end{array}$$

Figure 2. A constraint graph representing a overlap-free constraint.

For illustration, we briefly return to the λ-structure for $\lambda x.\mathsf{car}(x)$ in Figure 1.

In this λ-structure, the relations $u_1{:}@(u_2, u_3)$, $u_0 \lhd^* u_2$, and $\lambda(u_3) = u_0$ hold, among others.

Note that not every var node in a partial λ-structure is necessarily mapped to anything by the binding function. This deviates from the standard definition of (total) λ-structures, where all var nodes must be bound, but is necessary in order to maintain some intermediate results in Section 4. However, the underspecified descriptions we will use will generally enforce that all *relevant* var nodes have binders. Note further that trees and λ-structures can be defined by specifying the set of nodes as a "tree domain" – a set of words over the natural numbers. The two definitions are equivalent, but the graph definition is more convenient for the purposes of this paper.

2.2. Descriptions in CLLS

CLLS is a language for describing λ-structures. For the purposes of the present article, we confine ourselves to the sublanguage of CLLS providing dominance, labeling, and λ-binding constraints, but note in passing that the full language also contains *parallelism* and *anaphora* constraints; for details, see (Egg et al., 2000).

Figure 2 shows a description in CLLS, both as a graph and as a logical formula. A λ-structure $\mathcal{M}$ satisfies this description iff X denotes a node in $\mathcal{M}$ that is labeled by lam and has a single child, denoted by Y; this child dominates (is an ancestor of) another node, denoted by Z, which is labeled by var and λ-bound by the node referred to by X. For instance, the λ-structure in Figure 1 satisfies all of these constraints, given that X denotes its root, Y its inner node, and Z its right leaf.

We assume an infinite set Vars of (node) variables which we will, generally, denote X, Y, Z, U, V, W. The syntax of the fragment of CLLS we consider is defined in Figure 3. It provides conjunctions of *atomic constraints* for labeling ($X{:}f(X_1, \ldots, X_n)$), dominance ($X \lhd^* Y$), lambda binding ($\lambda(X){=}Y$), and inequality ($X{\neq}Y$). We freely abbreviate con-

$$\begin{array}{rll} \varphi & ::= & X{:}f(X_1,\dots,X_n) \qquad (f_{|n} \in \Sigma) \\ & | & X \lhd^* Y \\ & | & \lambda(X)=Y \\ & | & X \neq Y \\ & | & \varphi \wedge \varphi'. \end{array}$$

Figure 3. Syntax of a fragment of CLLS.

straints, such as $X \lhd^* Y \wedge Y \lhd^* X$ by $X=Y$ and $X \lhd^* Y \wedge X \neq Y$ by $X \lhd^+ Y$.

CLLS is interpreted over the class of λ-structures, which provide relations for the interpretations of all relation symbols, in the usual Tarskian way (see Section 2.5 for more details). Note that the same notation is used for relation symbols in constraints and the corresponding relations in a λ-structure. They can generally be distinguished by context, as relations are always applied to nodes u whereas relation symbols are applied to variables X.

2.3. Constraint Graphs

For underspecified descriptions, we will only use *overlap-free* constraints which contain sufficiently many inequalities for expressing that any two labeled variables denote distinct nodes.

Definition 2.2. *A constraint φ is called* overlap-free *iff for any two distinct labeling constraints $X{:}f(X_1,\dots,X_n)$ and $Y{:}g(Y_1,\dots,Y_m)$ occurring in φ it holds that $X \neq Y$ is in φ as well (even if $f = g$).*

For easier readability, we will usually draw overlap-free constraints as *constraint graphs*: Figure 2 is an example of an overlap-free constraint and its graph. The nodes of a constraint graph stand for variables of the constraint, and the various types of edges stand for labeling, dominance, and binding constraints. Furthermore, the constraint graph represents an inequality constraint for each pair of labeled nodes; so it really stands for an overlap-free constraint. Despite a superficial (and intended) similarity, it is important to distinguish constraint graphs from λ-structures.

Constraint graphs contain "rigid fragments", which are trees with solid edges whose leaves may or may not be labeled. Because the represented constraint is overlap-free, the variables corresponding to the inner nodes of two fragments must never be mapped to the same node;

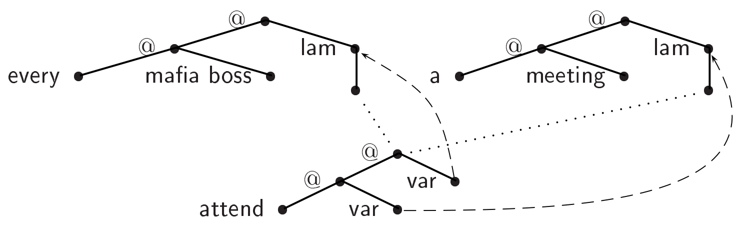


Figure 4. Semantic representation of a scope ambiguity.

that is, fragments may not *overlap*. However, an unlabeled leaf of one fragment may still be mapped to the same node as the root of another. Fragments will be defined formally in Section 5.

2.4. Scope Underspecification

We now illustrate briefly how CLLS can be used to model a scope ambiguity, as in the following example.

(5) Every mafia boss attends a meeting.

In one reading of the sentence, all mafia bosses attend the same meeting, while in the other one, there is not necessarily a single meeting which all of the mafia bosses attend. Under this second reading it would e.g. be possible that every mafia boss has his own meeting.

Its underspecified semantic representation is given in Figure 4. The constraint graph contains three rigid fragments: The fragments at the top represent the two quantifiers "every mafia boss" and "a meeting", and the one at the bottom the verb semantics. Scope ambiguities are characterized by involving two or more quantifiers whose relative scope is not fixed. In Figure 4, this is accounted for by imposing the constraint that both of the quantifier fragments must dominate the third one. As there can be no upward branching in trees, this enforces that one of the quantifier fragments has to be above the other in each tree described by the graph, but the exact ordering is left open.

However, this constraint has not only the two obvious solutions; the satisfying λ-structures could be much larger, as long as they contain the material required in the constraint. In other words, there might be nodes in a solution of a constraint φ which are not referred to by any variable of φ. Note that CLLS differs from most other underspecification formalisms in this respect (Alshawi and Crouch, 1992; Reyle, 1993; Bos, 1996), which is an essential prerequisite for underspecified reinterpretation as considered in this paper.

2.5. First-Order Formulas

CLLS constraints are only conjunctions of atomic constraints. However, it will be useful later to use arbitrary first-order formulas Φ over these atomic constraints. Their function is to facilitate reasoning about underspecified semantic representations; they are not used as underspecified semantic representations themselves, in order to save unnecessary computational complexity and also because our basic notions of safety and chains (e.g. Lemma 4.7 and thus Proposition 4.8) depend on the fact that CLLS does not support quantification over nodes.

The set of (free) node variables of a first-order formula Φ over CLLS constraints is denoted by $\mathit{Var}(\Phi)$. A *variable assignment* into a λ-structure (V, E, L, λ) is a partial function $\alpha : \mathsf{Vars} \rightsquigarrow V$. We write $\mathrm{Dom}(\alpha)$ for the domain of α. A *solution* of a formula Φ consists of a λ-structure $\mathcal{M}$ and a variable assignment α into $\mathcal{M}$ with $\mathit{Var}(\Phi) \subseteq \mathrm{Dom}(\alpha)$ under which Φ evaluates to true. We also say that $(\mathcal{M}, \alpha)$ *satisfies* Φ and write $\mathcal{M}, \alpha \models \Phi$ if $(\mathcal{M}, \alpha)$ is a solution of Φ. We write $\Phi \models \Phi'$ and say that Φ *entails* Φ' if every solution of Φ interpreting all variables in $\mathit{Var}(\Phi')$ is a solution of Φ'.

Incidentally, we will only use the propositional connectives of first-order logic, disjunction and negation, but not quantification. Negation allows to express *disjointness* between two nodes in a tree structure, meaning that neither of them dominates the other, by the following formula $X \bot Y$:

$$X \bot Y =_{\mathrm{def}} \neg X \lhd^* Y \wedge \neg Y \lhd^* X$$

3. Underspecified Reinterpretation

We already saw an example of reinterpretation in the introduction, namely (1). Expressed in λ-terms of higher-order logics, which are the basis of the semantic representation formalism introduced in the previous chapter, the semantics of (1) should look like (6) for instance.

(6) $a(\mathit{book})(\lambda x.\mathit{begin}(\mathit{peter}, \mathit{read}(\mathit{peter}, x)))$

However, traditional semantic construction would rather derive something like (7).

(7) $a(\mathit{book})(\lambda x.\mathit{begin}(\mathit{peter}, x))$

So, what reinterpretation has to do here, intuitively, is to fill in semantic material that was not present on the surface ($\mathit{read}(\mathit{peter}, \bullet)$ in this

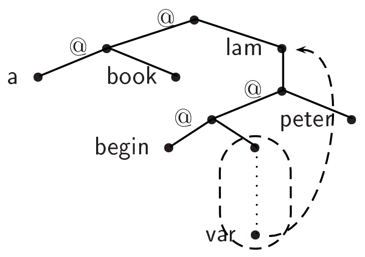


Figure 5. Relaxed semantic representation of (1)

example) at the location of the type conflict. We call this location the *reinterpretation site*, and the additional semantic material, the *reinterpretation operator*.

We now describe Egg's approach to modeling phenomena of this type by exploiting the underspecification mechanisms introduced in the previous section. Then we show that the general relaxation operation underlying this approach may give rise to overgeneration, which introduces the problem that leads the way for the rest of this paper.

3.1. Underspecified Reinterpretation

Recently, Egg (2000) has proposed to describe sentences requiring reinterpretation in an underspecified way, thereby avoiding conflicts. His main idea is to derive sufficiently *relaxed* semantic representations in which gaps, modeled by dominance edges, are left open at all possible reinterpretation sites. The actual reinterpretation step simply is further specialization, i.e instantiation of these relaxation gaps; the approach assumes that suitable reinterpretation operators can be determined by some independent process. For illustration, Egg's semantic construction applied to (1) derives a relaxed semantic representation that essentially looks as in Figure 5. The reinterpretation site, where relaxation took place, is highlighted in this figure by the dashed box.

By introducing a dominance edge at the reinterpretation site, the semantic representation is made less specific. Figure 5 can be seen as a description of (7) as well as (6), since the gap in (5) can be eliminated by identifying the two nodes at its ends.

One advantage of Egg's approach to reinterpretation is that it is compatible with an underspecified treatment e.g. of scope in a straightforward way. Consider the following example which contains a scope ambiguity in addition to a type conflict.

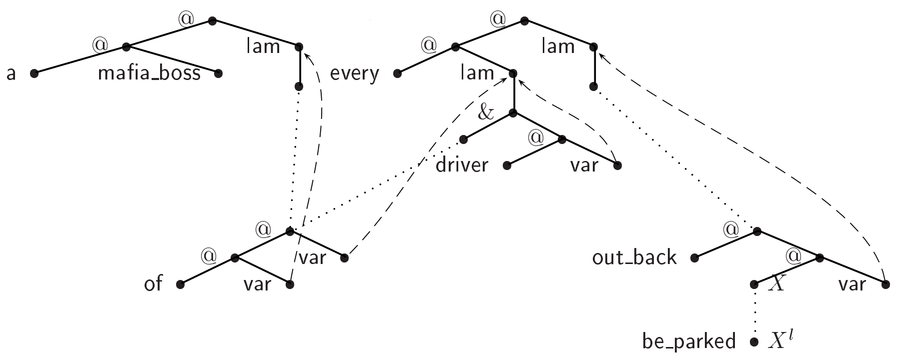


Figure 6. Relaxed underspecified representation of Example (8).

(8) Every driver of a mafia boss is parked out back.

The (relaxed) semantic representation of (8) is presented in Figure 6. In reading the constraint graph in Figure 6, it is first of all helpful to identify its various fragments, most notably the contributions of “a mafia boss”, “every driver”, “of”, and “be parked out back”. The scope ambiguity between “a mafia boss” and “every driver” is modeled by requiring that both fragments must dominate the fragment corresponding to “of”, but leaving their relative position open. Reinterpretation has to coerce “every driver of a mafia boss” into their vehicles. We can do this by filling the appropriate reinterpretation operator into the relaxation gap, i.e. the gap provided by the dominance edge $X \lhd^* X^l$ in the description of the verb semantics. An appropriate reinterpretation operator linking the drivers to their cars is given in Figure 7.

3.2. A Problem?

There are four dominance edges in the description in Figure 6. Three of them, namely the ones connecting “a mafia boss”, “every driver”, “of”, and “be parked out back”, were introduced in order to provide for a correct modeling of scope ambiguities, i.e. they allow to arrange the semantic material that was introduced by semantic construction in different ways into trees. The fourth dominance edge, between X and X^l, was introduced by relaxation to make space for the reinterpretation operator. It is important that *only* reinterpretation operators that are newly added to the description should appear inside the gap. In particular, we do not want solutions where material that was introduced

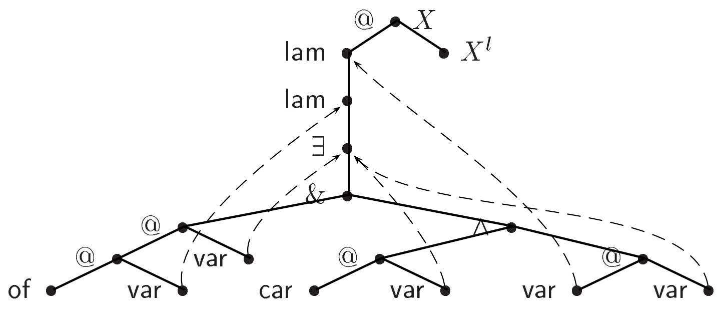


Figure 7. The reinterpretation operator for Figure 6.

by semantic construction appears in the relaxation gap as if there was a scope ambiguity.

This cannot happen in Figure 6 – we say that this description is *safe* –, but is this a general feature of relaxation? Unfortunately not, as the abstract examples in Figure 8 show. Here a relaxation of a constraint graph has a solution where material already present in the original constraint appears at the position of the gap. We call this kind of solution *unintended*.

The problem is that descriptions may contain material that is freely floating around (like e.g. the fragments containing the node labeled with f in Figure 8). In a solution, this material can be mapped to almost any position where the specification of the constraint leaves a gap open. So, it could end up in a relaxation gap, which it should not. This is the potential overgeneration problem of relaxation we mentioned in the introduction: relaxations can have unintended solutions in addition to those solutions we really want; that is, not all relaxations are safe.

There are several conceivable ways out of this problem. One would be to explicitly exclude unintended solutions by adding first-order formulas during relaxation which state that the variables of the unrelaxed constraint must not be mapped into a region of a tree model that fills a relaxation gap. This idea has the disadvantage that employing a more expressive description language (first-order instead of conjunctive constraints) would increase the complexity of computation involving underspecified representations. Note, however, that a similar idea has already been proposed in (Rambow et al., 1995).

Another idea might be that unintended solutions typically violate type and sort restrictions, and they could be filtered out by excluding ill-typed solutions. However, this will not always work; and more im-

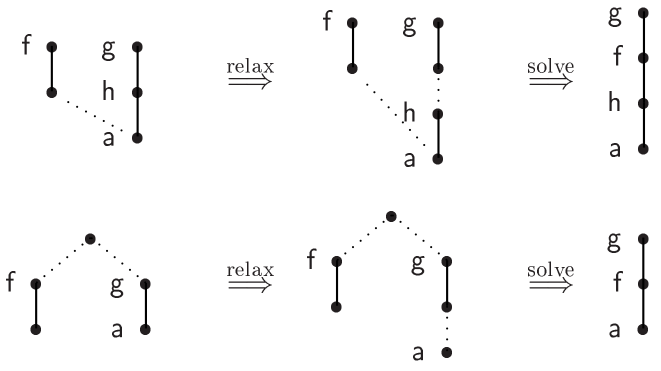


Figure 8. Unsafe constraints and their relaxations.

portantly, as the previous discussion has shown, type and sort conflicts do occur in natural language and do not necessarily lead to exclusion, but rather to a process of reinterpretation repairing the conflict.

The solution proposed in this article relies on the observation that those CLLS constraints that actually arise as underspecified semantic representations have particular properties which ensure that their relaxation is safe anyway. That is, although in general relaxation may give rise to unintended solutions, it never does when used for underspecified reinterpretation.

4. Relaxation

Now that we have a definition of our underspecification formalism and an intuition of what "relaxation" is supposed to mean, we make this notion formally precise. We will define the concepts of *safe* relaxation and of *open* constraints. These concepts are both connected to the notion of an "intended solution", which we used informally in the previous section. We will prove in Section 7 that the relaxations we use in underspecified reinterpretation are all safe and open.

4.1. Constraint Relaxation

Intuitively, relaxation of a constraint means to split a single node of the corresponding constraint graph in two, connecting the two new nodes with a dominance edge. In capturing this idea formally, it turns out that it is cumbersome to define a general relaxation operation that works on every possible constraint. Instead, we define what it means for a constraint to be the relaxation of another constraint. In Section 7, we then need to show that the "relaxed" constraints that the syntax/semantics interface produces are really relaxations of the "unrelaxed" versions.

Definition 4.1 (Relaxation). *Let φ, φ' be constraints and X a variable in φ. φ' is called a* relaxation of φ at X with X^l *iff $\varphi' \models X \triangleleft^* X^l$, $Var(\varphi') = Var(\varphi) \uplus \{X^l\}$ and*

$$\exists X^l(\varphi' \wedge X^l{=}X) \models\!\!\!\!\!\dashv \varphi.$$

This definition captures the idea of a relaxation because it says that there is some variability for what is in between X and X^l in a solution of φ'; but assuming that they are equal, every solution of φ' must also be a solution of φ.

However, relaxations in this sense can misbehave in various pathological ways. We have seen in the previous section how an *unsafe* relaxation can allow for material to disappear into the newly opened gap. Another problem is that by the above definition, φ is its own relaxation – which is certainly counterintuitive. Below, we define two properties of relaxations – safety and openness – which exclude these two types of problems.

4.2. Safety and Openness

As we have seen in the previous section, *safety* is the property which excludes material specified by a constraint from being mapped into the relaxation gap. We can define safety as follows:

Definition 4.2 (Safety). *Let φ' be a relaxation of φ at X with X^l. φ' is a* safe *relaxation of φ at X iff*

$$\varphi' \models \bigwedge_{Y \in Var(\varphi)} (Y \triangleleft^* X \vee Y \perp X \vee X^l \triangleleft^* Y).$$

Another interesting property of a constraint is to be *open* between two variables: This means that the tree structure between the denotations of the two variables X and X^l is not constrained. Thus, one can freely fill in arbitrary material at the relaxations site while still satisfying the constraint. In order to formalize this idea, we need the notion of a projection.

Definition 4.3 (Projection). *Let $\mathcal{M} = (V, E, L, \lambda)$ be a λ-structure with nodes $u \triangleleft^* u^l$ in V. Let $V^u_{u^l}$ be the set of nodes of $\mathcal{M}$ situated strictly*

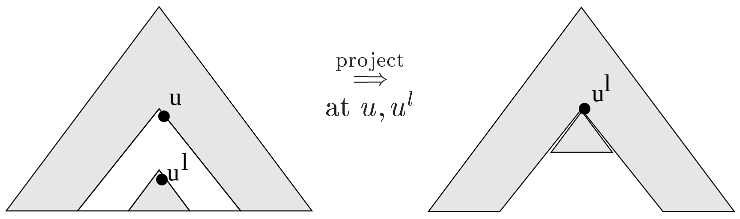


Figure 9. Projection at u, u^l.

between u and u^l:

$$V^u_{u^l} \;=\; \{v \in V \mid u \triangleleft^* v \textit{ in } \mathcal{M} \textit{ and not } u^l \triangleleft^* v \textit{ in } \mathcal{M}\}$$

We define the projection $p^u_{u^l}(\mathcal{M})$ *of* $\mathcal{M}$ *at* u, u^l *illustrated in Figure 9 by applying the following consecutive steps to* $\mathcal{M}$*:*

1. *Remove all edges whose source is in* $V^u_{u^l}$.
2. *Remove all* λ*-binding pairs that involve a node in* $V^u_{u^l}$.
3. *Replace the edge* (r, u)*, if it exists, by* (r, u^l).
4. *Remove all nodes in* $V^u_{u^l}$.

Note that $p^u_{u^l}(\mathcal{M})$ *is indeed a* λ*-structure since* $u \triangleleft^* u^l$*. Given a variable assignment* α *into* $\mathcal{M}$ *and variables* X, X^l *with* $\alpha(X) \triangleleft^* \alpha(X^l)$ *we define its projection* $p^X_{X^l}(\alpha)$ *to be the variable assignment into* $p^{\alpha(X)}_{\alpha(X^l)}(\mathcal{M})$ *which maps, for all* $Y \in \mathsf{Vars}$,

$$p^X_{X^l}(\alpha)(Y) = \begin{cases} \textit{undefined} & \textit{if } \alpha(Y) \in V^{\alpha(X)}_{\alpha(X^l)} \backslash \{\alpha(X)\} \\ \alpha(X^l) & \textit{if } \alpha(Y) = \alpha(X) \\ \alpha(Y) & \textit{otherwise} \end{cases}$$

The projection $p^X_{X^l}(\mathcal{M}, \alpha)$ *of a pair* $(\mathcal{M}, \alpha)$ *satisfying* $\alpha(X) \triangleleft^* \alpha(X^l)$ *is the pair* $(p^{\alpha(X)}_{\alpha(X^l)}(\mathcal{M}), p^X_{X^l}(\alpha))$.

Definition 4.4 (Openness). *A constraint* φ *is* open between X and X^l *iff* $\varphi \models X \triangleleft^* X^l$ *and for each pair* $(\mathcal{M}, \alpha)$ *where* α *assigns into* $\mathcal{M}$*:*

$$\textit{if } p^X_{X^l}(\mathcal{M}, \alpha) \models \varphi \textit{ then } \mathcal{M}, \alpha \models \varphi.$$

Clearly, it is important for a relaxation to be open, but not all relaxations are. $X{:}a$, for instance, is its own relaxation at X with X^l and is not open between X and X^l. There is however a certain class of safe relaxations, which all are open, as expressed by the following proposition.

Proposition 4.5 (Openness of Safety). *Let φ be a safe relaxation at X with X^l such that no constraints in φ mention X and X^l, except for $X \triangleleft^* X^l$ and $X^l{:}f(\ldots)$. Then φ is open between X and X^l.*

Proof. Let $(\mathcal{M}, \alpha)$ be a tuple consisting of a λ-structure and a variable assignment, and let $p^X_{X^l}(\mathcal{M}, \alpha)$ be a solution of φ. We have to show that $(\mathcal{M}, \alpha)$ also is a solution of φ. Because φ is safe, we know that α and $p^X_{X^l}(\alpha)$ coincide on $Var(\varphi)$. By the definition of projection all atomic constraints of φ except for $X \triangleleft^* X^l$ are satisfied by $\mathcal{M}$ in the same way they are satisfied by $p^X_{X^l}(\mathcal{M})$. And $X \triangleleft^* X^l$ is of course also satisfied by $(\mathcal{M}, \alpha)$. □

4.3. Intended solutions

An equivalent characterization of safety and openness can be obtained by specifying *intended solutions* of relaxed constraints. The fact that such an equivalent definition exists reinforces our confidence into the notion of open and safe relaxations, even though it is not essential for the remainder of this article.

Definition 4.6 (Intended Solutions). *Let φ' be a relaxation of φ at X with X^l. A pair $(\mathcal{M}, \alpha)$ is called an* intended solution *of φ' iff $\alpha(X) \triangleleft^* \alpha(X^l)$ and the projection $p^X_{X^l}(\mathcal{M}, \alpha)$ is a solution of φ.*

Lemma 4.7. *Let φ' be a safe relaxation of φ at X with X^l. If $(\mathcal{M}, \alpha)$ is a solution of φ' then its projection $p^X_{X^l}(\mathcal{M}, \alpha)$ solves φ' as well.*

Proof. Let $\mathcal{M} = (V, E, L, \lambda)$ and $\mathcal{M}, \alpha \models \varphi'$. Since $Var(\varphi') = Var(\varphi) \uplus \{X^l\}$, safety yields that $p^{X^l}_X(\alpha)$ is defined for all variables in $\mathsf{Vars}(\varphi')$. Now we can verify that the projection satisfies each atomic constraint in φ'. For instance, let $Y \triangleleft^* Z$ be in φ' then $\alpha(Z)$ can be reached form $\alpha(Y)$ via edges in $\mathcal{M}$. Thus, $p^{X^l}_X(\alpha)(Z)$ can be reached from $p^{X^l}_X(\alpha)(Y)$ in $p^{\alpha(X^l)}_{\alpha(X)}(\mathcal{M})$, i.e. $p^X_{X^l}(\mathcal{M}, \alpha) \models Y \triangleleft^* Z$. The arguments for labeling and λ-binding constraints are similar. □

Proposition 4.8. *All solutions of a safe relaxation are intended.*

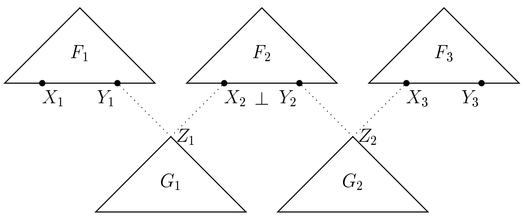


Figure 10. A chain of fragments.

Proof. Let φ' be a safe relaxation of φ at X with X^l and $(\mathcal{M}, \alpha)$ a solution of φ'. By Lemma 4.7, the projection $p^X_{X^l}(\mathcal{M}, \alpha)$ solves φ' as well. Trivially, it also solves $X = X_l$ and thus φ by the equivalence $\exists X^l(\varphi' \wedge X^l{=}X) \models\!\!\!\!\!\;\dashv \varphi$, i.e $(\mathcal{M}, \alpha)$ is an intended solution. □

Proposition 4.9. *Every intended solution of an open relaxation is indeed a solution.*

Proof. Let φ' be an open relaxation of φ at X with X^l. If $(\mathcal{M}, \alpha)$ is an intended solution of φ' then its projection $p^X_{X_l}(\mathcal{M}, \alpha)$ solves φ, and thus $\exists X^l(X{=}X^l \wedge \varphi')$. Since $p^X_{X_l}(\alpha)$ maps X and X_l to the same value, the projection $p^X_{X_l}(\mathcal{M}, \alpha)$ also solves φ'. The openness of φ' yields that $(\mathcal{M}, \alpha)$ solves φ' as well. □

5. Chains of Fragments

In this section, we introduce *chains* of rigid *fragments*. Chains are prototypical substructures of underspecified semantic representations in CLLS. It will turn out that these representations may be more complex than a single chain, but that they can be *covered* by several chains.

A chain is intuitively a construction as in Figure 10, where rigid fragments (drawn as triangles) are connected via dominance constraints between unlabeled leaves of the upper fragments and the roots of the lower fragments. By way of example, reconsider Figure 6 which contains a chain of length two, whose upper fragments are those corresponding to *a mafia boss* and *every driver* and whose lower fragment corresponds to the preposition.

Now we make the informal notions of fragments and chains, which we have used in explanations of constraint graphs all along, precise.

Fragments in an overlap-free constraint are sets of variables that are *connected* by labeling constraints.

Definition 5.1 (Connected Variables). Connectedness *in φ is the smallest binary equivalence relation over $Var(\varphi)$ which contains all pairs (X, Y) such that $X{:}f(\dots Y \dots)$* in *φ.*

Definition 5.2 (Fragments). *Let φ be an overlap-free constraint. A* fragment *of φ is a subset $F \subseteq Var(\varphi)$ of variables that are pairwise connected in φ. A variable $X \in F$ is called a* labeled leaf *of a fragment F if φ contains $X{:}a$ for some constant $a \in \Sigma$ and an* unlabeled leaf *if no labeling constraint $X{:}f(\dots)$ occurs in φ at all. A* leaf *of F is either a labeled or an unlabeled leaf of F.*

We are usually interested in maximal fragments, but will allow non-maximal ones as well. In the constraint graph, fragments look like parts of trees. So intuitively, they should have a unique root, and their leaves should be pairwise disjoint. This is indeed the case:

Lemma 5.3 (Treeness of Fragments). *Let φ be an overlap-free constraint, and let F be a fragment of φ. Then F has the following properties:*

1. *There is a unique variable $Y \in F$ (which we call its* root*) such that $\varphi \models Y \triangleleft^* X$ for all $X \in F$.*
2. *For any two different leaves X, Y of F it holds $\varphi \models X \perp Y$* at *$F$.*

We omit the proof, which is straightforward.

Definition 5.4 (Chains). *Let φ be an overlap-free constraint, and let $\mathcal{F} = (F_1, \dots, F_n)$ and $\mathcal{G} = (G_1, \dots, G_{n-1})$ be sequences of fragments in φ. We assume that no variable appears in two different fragments; so these fragments cannot overlap properly. For all i, let X_i, Y_i be different unlabeled leaves of F_i, and let Z_i be the root of G_i. Then the pair $\mathcal{C} = (\mathcal{F}, \mathcal{G})$ is called a* chain *in φ of length n and with* end points *X_1 and Y_n iff for all $1 \leq i \leq n-1$:*

$$\varphi \models Y_i \triangleleft^* Z_i \wedge X_{i+1} \triangleleft^* Z_i$$

If a constraint contains a chain then it entails some very useful relationships between its variables.

Proposition 5.5 (Relations in Chains). *If $\mathcal{C}$ is a chain in a constraint φ with end point X then all other variables Y of $\mathcal{C}$ satisfy:*

$$\varphi \models X \perp Y \vee Y \triangleleft^* X.$$

Proof. The proof is by induction on the length n of the chain $\mathcal{C}$. Let fragments and variables of $\mathcal{C}$ be named as in Definition 5.4. The case $n = 1$ follows from the treeness of upper fragments (Lemma 5.3). Assume $n \geq 2$ and let $X = X_1$ w.l.o.g. Since $Y_1 \lhd^* Z_1 \wedge X_2 \lhd^* Z_1 \in \varphi$ it follows that $\varphi \models Y_1 \lhd^* X_2 \vee X_2 \lhd^* Y_1$. Let $U_1 \in F_1$ and $U_2 \in F_2$ be the roots of their fragments. As fragments of chains cannot overlap properly, it follows that $\varphi \models Y_1 \lhd^* U_2 \vee X_2 \lhd^* U_1$. For the first case, we consider $\varphi \wedge Y_1 \lhd^* U_2$: If $Y \in F_2 \cup G_1$ then $\varphi \models Y_1 \lhd^* Y \wedge X_1 \perp Y_2$ and we are done. Otherwise, we cancel out G_1 and F_2 and apply the induction hypothesis. For the second case, we consider $\varphi \models X_2 \lhd^* U_1$. Again, the case $Y \in F_2 \cup G_1$ are obvious. Otherwise, we cancel out F_1 and G_1. The induction hypothesis yields $\varphi \models X_2 \perp Y \vee Y \lhd^* X_2$. Thus, $\varphi \models X \perp Y \vee Y \lhd^* X$. □

Proposition 5.6 (Disjointness of End Points). *If X, Y are the end points of a chain in φ, then $\varphi \models X \perp Y$.*

Proof. Proposition 5.5 applied twice (once for each end point) proves:

$$\varphi \models (X \perp Y \vee Y \lhd^* X) \wedge (Y \perp X \vee X \lhd^* Y)$$

The disjointness of end points holds obviously in all case except for $\varphi \wedge Y \lhd^* X \wedge X \lhd^* Y$ which is impossible as $\varphi \models X \neq Y$. □

6. Semantic Construction

Now, we define a toy grammar and a syntax/semantics interface that produces underspecified semantic representations in CLLS. The coverage of the grammar is obviously limited. However, the structure of the semantic representations derived by it are prototypical for the kind of structures one will see in representations of natural language semantics. As a matter of fact, we have implemented an HPSG grammar with a much wider coverage which produces the same type of constraints. The syntax/semantics interface produces constraints that may be relaxed at every variable whose label carries the semantics of a verb. We will prove in the next section that these relaxations are always open and safe.

We leave it as a (nontrivial) open problem to generalize our results further, to classes of grammars and syntax/semantics interfaces. The main problem in doing so is to find the right abstract properties of the syntax/semantics interface which ensure safety of relaxations.

(a1) $\mathtt{S} \rightarrow \mathtt{NP}\ \mathtt{VP}$	**(a6)** $\mathtt{VP} \rightarrow \mathtt{VP}\ \mathtt{Adv}$
(a2) $\mathtt{NP} \rightarrow \mathtt{Det}\ \overline{\mathtt{N}}$	**(a7)** $\mathtt{VP} \rightarrow \mathtt{IV}$
(a3) $\overline{\mathtt{N}} \rightarrow \mathtt{N}$	**(a8)** $\mathtt{VP} \rightarrow \mathtt{TV}\ \mathtt{NP}$
(a4) $\overline{\mathtt{N}} \rightarrow \mathtt{N}\ \mathtt{PP}$	**(a9)** $\mathtt{VP} \rightarrow \mathtt{CV}$ to $\mathtt{VP}$
(a5) $\mathtt{PP} \rightarrow \mathtt{P}\ \mathtt{NP}$	**(a10)** $\gamma \ \rightarrow \ W \quad \text{if } (W, \gamma) \in \mathit{Lex}$

Figure 11. The grammar

6.1. The Grammar

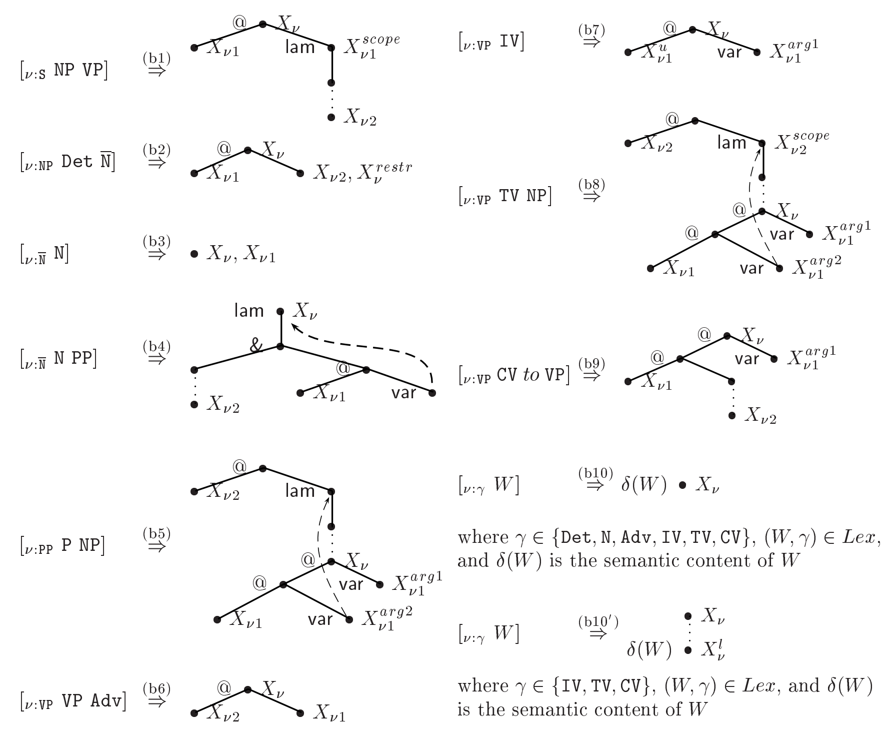
The grammar fragment we consider is displayed in Figure 11 (where IV = intransitive verb; TV = transitive verb; CV = (subject) control verb). *Lex* is a relation between words W and lexical categories $\gamma \in \{\mathtt{Det}, \mathtt{N}, \mathtt{IV}, \mathtt{TV}, \mathtt{CV}, \mathtt{Adv}\}$ which represents the lexicon. The coverage of this grammar is limited, but it should be a simple matter to extend our results to a larger grammar that covers constructions like relative clauses, sentential complement verbs, and ditransitive verbs. Of course, any serious NLP system would employ some unification grammar formalism, which would then also allow to take care of aspects such as agreement which we have ignored completely.

6.2. The Syntax/Semantics Interface

The syntax/semantics interface of the grammar associates with each nonterminal node ν of the parse tree a variable X_ν and a subconstraint that talks about this variable. The contributions of all these nodes are then conjoined, and a sufficient number of inequality constraints is added to make the result overlap-free, i.e. to prevent identification of labeled nodes in the solution.

The rules by which the subconstraints are introduced are presented in Figure 12. We take $[_{\nu:\gamma}\ \gamma_1\ \gamma_2]$ to mean that the node ν in the syntax tree is labeled with category γ, and its two daughter nodes $\nu1$ and $\nu2$ are labeled with γ_1 and γ_2, respectively. The constraint introduced by such a rule then imposes a CLLS constraint on the variables $X_\nu, X_{\nu1}, X_{\nu2}$ which are distinguished variables in the subconstraints associated with $\nu, \nu1$, and $\nu2$ respectively.

Some other variables in the constraints are mentioned explicitly as well, e.g. X_ν^{scope} in rule (b1). This is to make their specific functions



$[_{\nu:\mathtt{S}}\ \mathtt{NP\ VP}] \overset{(b1)}{\Rightarrow}$
$[_{\nu:\mathtt{NP}}\ \mathtt{Det}\ \overline{\mathtt{N}}] \overset{(b2)}{\Rightarrow}$
$[_{\nu:\overline{\mathtt{N}}}\ \mathtt{N}] \overset{(b3)}{\Rightarrow} \bullet X_\nu, X_{\nu 1}$
$[_{\nu:\overline{\mathtt{N}}}\ \mathtt{N\ PP}] \overset{(b4)}{\Rightarrow}$
$[_{\nu:\mathtt{PP}}\ \mathtt{P\ NP}] \overset{(b5)}{\Rightarrow}$
$[_{\nu:\mathtt{VP}}\ \mathtt{VP\ Adv}] \overset{(b6)}{\Rightarrow}$
$[_{\nu:\mathtt{VP}}\ \mathtt{IV}] \overset{(b7)}{\Rightarrow}$
$[_{\nu:\mathtt{VP}}\ \mathtt{TV\ NP}] \overset{(b8)}{\Rightarrow}$
$[_{\nu:\mathtt{VP}}\ \mathtt{CV}\ \mathit{to}\ \mathtt{VP}] \overset{(b9)}{\Rightarrow}$
$[_{\nu:\gamma}\ W] \overset{(b10)}{\Rightarrow} \delta(W) \bullet X_\nu$
$[_{\nu:\gamma}\ W] \overset{(b10')}{\Rightarrow} \delta(W)$
X_ν^l
@
lam
var
&
X_ν
$X_{\nu 1}$
$X_{\nu 2}$
$X_{\nu 1}^u$
$X_{\nu 1}^{scope}$
$X_{\nu 2}^{scope}$
$X_{\nu 2}, X_\nu^{restr}$
$X_{\nu 1}^{arg1}$
$X_{\nu 1}^{arg2}$

where $\gamma \in \{\mathtt{Det}, \mathtt{N}, \mathtt{Adv}, \mathtt{IV}, \mathtt{TV}, \mathtt{CV}\}$, $(W, \gamma) \in Lex$, and $\delta(W)$ is the semantic content of W

where $\gamma \in \{\mathtt{IV}, \mathtt{TV}, \mathtt{CV}\}$, $(W, \gamma) \in Lex$, and $\delta(W)$ is the semantic content of W

Figure 12. The syntax/semantics interface

explicit and facilitate later reference. The variable X_ν^{scope} introduced by rule (b1) is intuitively the scope of the quantifier represented by the NP.

We exploit this to add appropriate λ-binding constraints that cannot be specified locally in Figure 12 without cluttering up the notation. Instead, we assume information about subjects and objects of verbs, which will be available in any more serious grammar formalism. For the object variables introduced in the rules (b7) to (b9), suppose that ν is a VP node in the parse tree and ν' is the NP node that represents the subject; then we add the following λ-binding constraint:

$$\lambda(X_\nu^{arg1}) = X_{\nu'}^{scope}$$

Similarly for rule (b5), if ν' is the NP node modified by the PP at ν then we add the following λ-binding constraint:

$$\lambda(X_\nu^{arg1}) = X_{\nu'}^{restr}$$

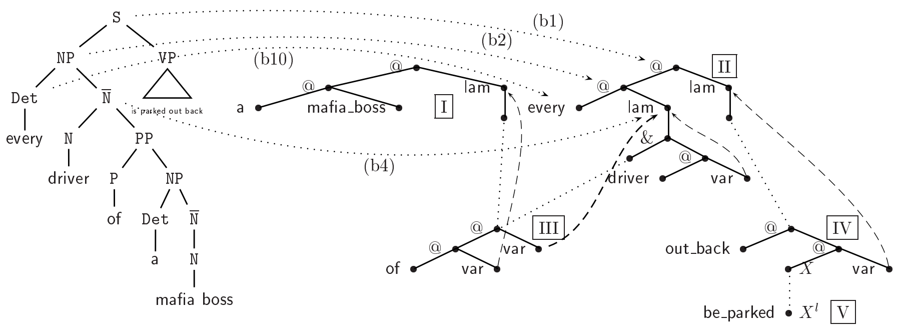


Figure 13. Every driver of a mafia boss is parked out back.

Finally, note how relaxation is compiled into the syntax/semantics interface. Rules (b10) and (b10') are the semantic construction rules for the syntactic rules subsumed under (a10). For categories other than verbs, we can only apply the rule (b10), which just contributes a variable with the appropriate label. For verbs, however, we have a choice between application of (b10) and (b10'); (b10') introduces a dominance constraint for potential reinterpretation. It is these relaxations that we must prove open and safe. This choice can be made nondeterministically. In "real" semantic construction, we will always apply (b10'), that is, we produce maximally relaxed constraints; but we still need (b10) so we can formulate the results in Section 7.

Before we come to the proofs that this specific type of reinterpretation has all the pleasant properties discussed above, we go through an example to illustrate how semantic construction operates.

To this end, we briefly return to Example (8), whose (relaxed) underspecified semantic representation we showed in Figure 6. Figure 13 relates the semantic representation to the parse tree our grammar assigns to this sentence. The dotted arrows go from node ν in the parse tree to the corresponding node X_ν in the semantic representation, indicating which part of the constraint is the semantic contribution of node ν. The dotted arrows are furthermore labeled with the rules of the syntax/semantics interface that were used. For the top most `NP` node e.g., rule (b2) introduces only a labeling constraint. Its first daughter labeled with `Det` only adds the node label every (rule (b10)). Its second daughter, labeled with $\overline{\texttt{N}}$, then introduces a bigger fragment, the root of which is labeled by lam (rule (b4)).

7. Correctness of Underspecified Reinterpretation

In this section, we put all the pieces we have assembled throughout the paper together. We identify a class of constraints with particular structural properties that efficiently supports proving safety of a certain relaxation. This structure depends heavily on the notion of chains, which we presented in Section 5.

To illustrate the use of coverings, we apply them first to two particular relaxed semantic representations, showing that these relaxations are safe. Then we generalize these results to any output of the syntax/semantics interface and demonstrate a strategy for proving that a syntax/semantics interface never produces unsafe relaxations.

7.1. Covering Constraints

We start by defining what it means for a constraint to be covered.

Definition 7.1 (Covering). *Let φ be an overlap-free constraint, let $\mathcal{O}$ be a set of fragments in φ. We call $\mathcal{O}$ a* covering of φ for X, Y *iff $Var(\varphi) = \bigcup\{F | F \in \mathcal{O}\}$ and for each fragment $F \in \mathcal{O}$ one of the following holds:*

1. *either Y dominates the root U of F, i.e. $\varphi \models Y \triangleleft^* U$,*
2. *or there is a chain $\mathcal{C}$ with fragments from $\mathcal{O}$ including F such that some end point Z of $\mathcal{C}$ dominates X, i.e. $\varphi \models Z \triangleleft^* X$,*
3. *or there is a chain $\mathcal{C}$ in φ with fragments of $\mathcal{O}$ then one of the end points dominates X and the other one the root of F.*

Now, we prove a property of coverings, which make them a very interesting and convenient structure for our purposes.

Proposition 7.2 (Safety of Coverings). *Let φ be an overlap-free constraint with covering $\mathcal{O}$ for X, Y. Then, for all $Z \in Var(\varphi)$:*

$$\varphi \models Z \triangleleft^* X \vee Z \perp X \vee Y \triangleleft^* Z.$$

Proof. Every variable $Z \in Var(\varphi)$ belongs to a fragment $F \in \mathcal{O}$, such that one of the conditions of Definition 7.1 is satisfied. Let U be the root of F.

1. $\varphi \models Y \triangleleft^* U$ and therefore holds $Y \triangleleft^* Z$ as well.
2. Z belongs to a chain $\mathcal{C}$ and an end point of $\mathcal{C}$ dominates X. So we know from Proposition 5.5, that $\varphi \models X \perp Z \vee Z \triangleleft^* X$.

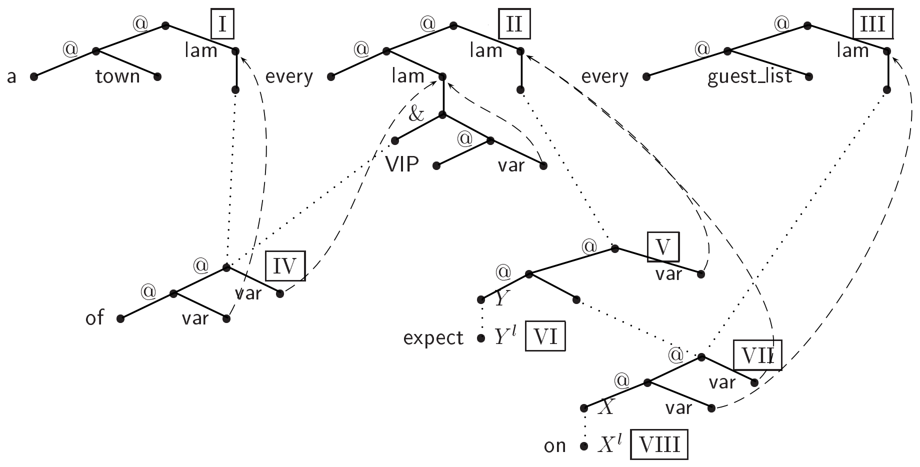


Figure 14. Every VIP of a town expects to be on every guest list.

3. Z is dominated by one end point of a chain and X by the other. From Proposition 5.6 we know that these two end point must be disjoint and therefore $X \perp Z$ also holds.

□

We now illustrate by means of two examples, how coverings can be used to show the safety of a relaxation. We start with the example of the previous section (Figure 13). To show that this constraint is actually safe, we have to prove that every variable in the constraint must either dominate X, be disjoint from X, or be dominated by X^l. To this end, we have to find a covering of the constraint for X, X^l, as defined in Definition 7.1. This will then give us the desired result by Proposition 7.2. Fragment V, containing only X^l is easy – it is if course dominated by itself. Fragments I, II, and III form a chain of length two and an end point of this chain dominates X. This leaves only fragment IV which can be seen as a chains of length one, and since X is a leaf of this fragment, X is of course dominated by an end point.

Admittedly, the structure of this example is very simple: it is more or less one chain with an extra fragment dangling from one of the outer connection points. As an example for a more complex structure the grammar fragment we presented can handle control verbs (rules (a9), (b9)). Consider the following example:

(9) Every VIP of a town expects to be on every guest list.

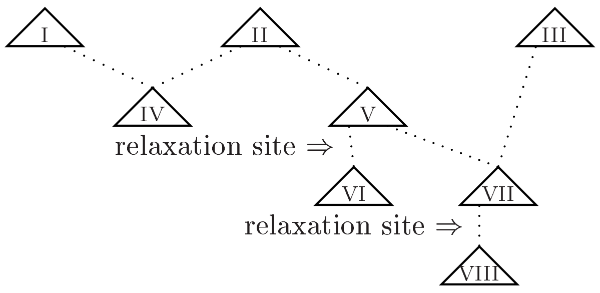


Figure 15. Schematic view of the Example.

We want to be able to reinterpret this sentence because only the *names* of the VIPs will be on the guest lists. The (maximally relaxed) underspecified semantic representation derived by the syntax/semantics interface is shown in Figure 14. To portray the structure more clearly, we have given a schematic representation in Figure 15. Note that there are two relaxation gaps in this example. To show that the constraint is safe we have to prove that every variable in the constraint must either dominate X, be disjoint from X, or be dominated by X^l and that the same holds with respect to Y and Y^l. Again we will use appropriate coverings from which we can draw the desired conclusions by Proposition 7.2.

For the relaxation gap between nodes X and X^l, we will employ one chain of length two, with upper fragments I, II and lower fragment IV. Furthermore we need three chains of length one consisting of fragment III, V, and VII respectively. Note that the one consisting of fragment V also does the covering for fragment VI. X^l again dominates itself, so that we do not have to worry about fragment $VIII$.

7.2. Structure of the Constraints

The constraints that the syntax/semantics interface can generate are of a specific form, specified by the following theorem.

Theorem 7.3 (Structure of the Constraints). *Let φ' be a constraint that we have obtained from the syntax/semantics interface by applying the construction rule (b10') for a certain verb node ν, and that φ is the constraint that we obtain from the same sentence by applying*

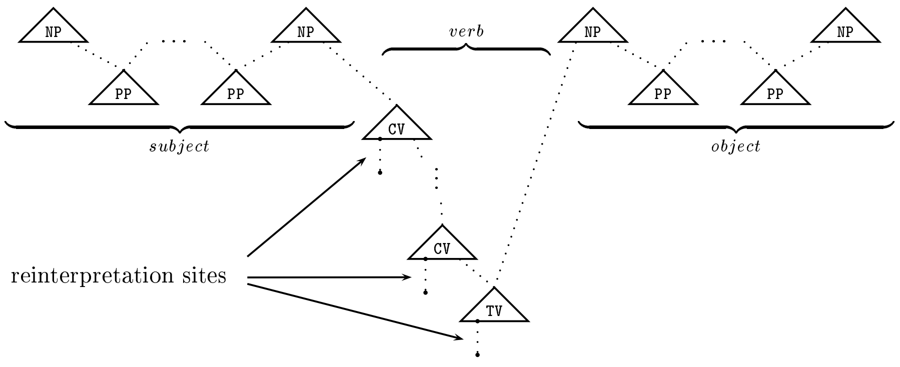


Figure 16. Output of the syntax/semantics interface: schematic view

the same construction rules everywhere, except for the node ν, where we apply (b10). Then, there is a covering $\mathcal{O}$ of φ' for X_ν, X_ν^l.

For the most part, the proof is only a precise formulation of the fact that the structure of a generated constraint generally looks as in Figure 16. We will not go into most parts of this proof, but one of the most interesting parts is the proof that the semantic contribution of a noun phrase is a chain whose length is the number of NP nodes in the parse tree of that noun phrase.

Proposition 7.4. *Let T be the parse tree from which we constructed φ'. Furthermore, let t be a subtree of T whose root is labeled with* NP, *let n be the number of* NP *nodes in t, and let φ_t be the conjunction of the constraints corresponding to the nodes of t. Then there is a singleton covering $\{\mathcal{C}\}$ of φ_t, where $\mathcal{C}$ is a chain of length n.*

Proof. By induction over n.

$\underline{n = 1.}$ In this case, t has the form $\mathtt{NP}(\mathtt{Det}(W_1)\ \overline{\mathtt{N}}(\mathtt{N}(W_2)))$, where W_1 and W_2 are words. As we can easily see from the rules (b2) and (b3), the resulting constraint φ_t is a single fragment and the claim is true with $\mathcal{C}$ being a chain of length 1.

$\underline{n - 1 \to n.}$ Let t' be the largest proper subtree of t whose root is labeled with NP and let $\varphi_{t'}$ be the contribution of t'. Note that $t = \mathtt{NP}(\mathtt{Det}(W_0)\overline{\mathtt{N}}(\mathtt{N}(W_1)\ \mathtt{PP}(\mathtt{P}(W_2)\ t')))$, where W_1 and W_2 are again words, and t' contains $n - 1$ NP nodes.

By the induction hypothesis there is a singleton covering $\{\mathcal{C}'\}$ of $\varphi_{t'}$, such that $\mathcal{C}' = ((F'_1, \dots, F'_{n-1}), (G'_1, \dots, G'_{n-2}))$.

According to the syntax/semantics interface, we obtain φ_t from $\varphi_{t'}$ by applications of the rules (b2), (b4), and (b5). These rules introduce two new fragments; let F_0 be the fragment consisting of the contributions of rules (b2) and (b4), and let G_0 be the fragment introduced by rule (b5). Furthermore, rule (b5) extends fragment F'_1 to a fragment F_1 which contains a new leaf, namely the scope. Finally, there are dominance constraints, demanding that the root of G_0 be dominated by the new leaf of fragment F_1 as well as a leaf of fragment F_0.

This means that $(F_0, F_1, F'_2, \dots, F'_{n-1})$ and $(G_0, G'_1, \dots, G'_{n-2})$ form a chain of length n in φ containing all variables of φ_t. Hence, this chain forms a singleton covering of φ_t. □

7.3. Safe, Open Relaxation

Now, the proof that all relaxed constraints derived by our syntax/semantics interface are safe and open relaxations of their unrelaxed counterparts, becomes quite simple.

Assume that φ and φ' are as defined in Theorem 7.3. First of all, φ' is really a relaxation of φ at X_ν with X^l_ν. The two constraints are equal, except for the constraints for the variables X_ν and X^l_ν. By adding an equality constraint for these two variables to φ', we clearly obtain a constraint which is equivalent to φ.

Safety of the relaxation follows by Proposition 7.2 directly from the fact that φ' has a covering for X_ν, X^l_ν. Openness of φ' between X_ν, X^l_ν follows from the fact that φ' is safe at X_ν with X^l_ν by Proposition 4.5, since we know that the only constraints concerning X_ν and X^l_ν that the syntax/semantics interface adds are the dominance constraint between X_ν and X^l_ν and a labeling constraint for X^l_ν.

8. Conclusion

Type and sort conflicts in semantics have to be resolved by reinterpretation in the presence of underspecification. This article investigates the formal foundations of underspecified reinterpretation in the framework CLLS. Its basic operation is the *relaxation* operation. It makes underspecified semantic representations even less specific, thereby making the introduction of additional material, such as reinterpretation operators, possible.

We identified a structural danger of overgeneration inherent to the relaxation of underspecified semantic representations. We formalized two wellness properties of relaxation, which ensure that structural overgeneration cannot arise: *openness* (arbitrary reinterpretation operators can be filled into the relaxation site) and *safety* (only reinterpretation operators can slip into relaxation gaps). We then introduced *chains* of *fragments* in tree descriptions and applied these structures to prove safety and openness for all CLLS relaxed descriptions produced by the syntax/semantics interface of a prototypical toy grammar. This result manifests our believe that while relaxation in general may cause problems, none of these problems occur in its application to underspecified reinterpretation.

9. Acknowledgements

We thank our reviewers for their detailed and helpful comments. We further thank all members of the CHORUS [1] project of the SFB 378 at the Universität des Saarlandes for the stimulating research environment they managed to create. This work was supported by the SFB 378 of the Deutsche Forschungsgemeinschaft and the DAAD Procope project.

Notes

[1] Papers of the CHORUS project are available at the web pages of the SFB 378: `http://www.coli.uni-sb.de/sfb378`

References

Alshawi, H. and R. Crouch: 1992, 'Monotonic semantic interpretation'. In: *Proceedings of the 30th ACL*. Kyoto, pp. 32–39.

Backofen, R., J. Rogers, and K. Vijay-Shanker: 1995, 'A First-order Axiomatization of the Theory of Finite Trees'. *Journal of Logic, Language, and Information* **4**, 5–39.

Bierwisch, M.: 1983, 'Semantische und konzeptionelle Repräsentation lexikalischer Einheiten'. In: R. Ruzicka and W. Motsch (eds.): *Untersuchungen zur Semantik*. Berlin: Akademie-Verlag, pp. 61–99.

Bos, J.: 1996, 'Predicate Logic Unplugged'. In: *Proceedings of the 10th Amsterdam Colloquium*. pp. 133–143.

Dölling, J.: 1994, 'Sortale Selektionsbeschränkungen und systematische Bedeutungsvariation'. In: M. Schwarz (ed.): *Kognitive Semantik/Cognitive Semantics*. Tübingen: Narr, pp. 41–59.

Duchier, D. and J. Niehren: 2000, 'Dominance Constraints with Set Operators'. In: *First International Conference on Computational Logic*. to appear in Juli.

Egg, M.: 2000, 'Reinterpretation by Underspecification'. Habilitation thesis, Universität des Saarlandes. In preparation.

Egg, M., A. Koller, and J. Niehren: 2000, 'The Constraint Language for Lambda Structures'. Submitted.

Egg, M., J. Niehren, P. Ruhrberg, and F. Xu: 1998, 'Constraints over Lambda-Structures in Semantic Underspecification'. In: *joint COLING/ACL*. pp. 353–359.

Gardent, C. and B. Webber: 1998, 'Describing discourse semantics'. In: *Proceedings of the 4th TAG+ Workshop*. Philadelphia.

Koller, A., K. Mehlhorn, and J. Niehren: 2000, 'A Polynomial-Time Fragment of Dominance Constraints'. Submitted.

Koller, A., J. Niehren, and R. Treinen: 1998, 'Dominance Constraints: Algorithms and Complexity'. In: *Proceedings of the Third Conference on Logical Aspects of Computational Linguistics*. Grenoble. To appear in LNCS.

Lascarides, A. and A. Copestake: 1998, 'Pragmatics and word meaning'. *Journal of Linguistics* **34**, 387–414.

Marcus, M. P., D. Hindle, and M. M. Fleck: 1983, 'D-Theory: Talking about Talking about Trees'. In: *Proceedings of the 21st ACL*. pp. 129–136.

Muskens, R.: 1995, 'Order-Independence and Underspecification'. In: J. Groenendijk (ed.): *Ellipsis, Underspecification, Events and More in Dynamic Semantics*. DYANA Deliverable R.2.2.C.

Nunberg, G.: 1995, 'Transfers of Meaning'. *Journal of Semantics* **12**, 109–132.

Pinkal, M.: 1996, 'Radical Underspecification'. In: *Proceedings of the 10th Amsterdam Colloquium*. pp. 587–606.

Pustejovsky, J.: 1995, *The generative lexicon*. Cambridge: MIT Press.

Rambow, O., K. Vijay-Shanker, and D. Weir: 1995, 'D-Tree Grammars'. In: *33rd Annual meeting of the Association for Computational Linguistics*. pp. 151–158.

Reyle, U.: 1993, 'Dealing with ambiguities by underspecification: construction, representation, and deduction'. *Journal of Semantics* **10**, 123–179.

van Deemter, K. and S. Peters: 1996, *Semantic Ambiguity and Underspecification*. Stanford: CSLI.

Vijay-Shanker, K.: 1992, 'Using Descriptions of Trees in a Tree Adjoining Grammar'. *Computational Linguistics* **18**, 481–518.

Vijay-Shanker, K., D. Weir, and O. Rambow: 1995, 'Parsing D-Tree Grammars'. In: *Proceedings of the International Workshop on Parsing Technologies*.