Relaxing Underspecified Semantic Representations for Reinterpretation

Alexander Koller
Dept. of Computational Linguistics
Universität des Saarlandes
Saarbrücken, Germany
koller@coli.uni-sb.de

Joachim Niehren
Programming Systems Lab
Universität des Saarlandes
Saarbrücken, Germany
niehren@ps.uni-sb.de

Kristina Striegnitz
Dept. of Computational Linguistics
Universität des Saarlandes
Saarbrücken, Germany
kris@coli.uni-sb.de

Abstract

Type and sort conflicts in semantics are usually resolved by a process of reinterpretation. Recently, Egg (1999) has proposed an alternative account in which conflicts are avoided by underspecification. The main idea is to derive sufficiently relaxed underspecified semantic representations; addition of reinterpretation operators then simply is further specialization. But in principle, relaxing underspecified representations bears the danger of overgeneration. In this paper, we investigate this problem in the framework of CLLS, where underspecified representations are expressed by tree descriptions subsuming dominance constraints. We introduce some novel properties of dominance constraints and present a safety criterion that ensures that an underspecified description can be relaxed without adding unwanted readings. We then apply this criterion systematically to Egg's analysis and show why its relaxation operation does not lead to overgeneration.

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

1 Introduction

Type and sort conflicts between functors and arguments in semantics are usually resolved by a process of reinterpretation (Bierwisch, 1983; Hobbs et al., 1993; Dölling, 1994; Copestake and Briscoe, 1995; Pustejovsky, 1995; Nunberg, 1995). Two classical examples are:

(1) Peter began a book.

(2) I am parked out back.

In sentence (2), it can be argued that persons can't be parked; it is really the speaker's car which is said to be parked out back. In sentence (1), the problem is that Peter can only begin an activity; in understanding it, we must fill in what, exactly, Peter begins to do with the book – for example, reading it, writing it, etc. However, traditional semantic construction would derive something like (3) as the semantics of (1):

(3) $\exists x.book(x) \wedge begin(peter, x)$,

Formula (3) is not well-typed because *begin* expects a proposition as its second argument, but x denotes an individual. The reinterpreted reading, which is what a human understands, is something like (4).

(4) $\exists x.book(x) \wedge begin(peter, read(peter, x))$.

What reinterpretion does here, intuitively, is to fill in semantic material that wasn't present on the surface, namely $read(peter, \bullet)$, at the location of the type conflict. We call this location the *reinterpretation site*, and the additional semantic material, the *reinterpretation operator*.

Recently, Egg (1999) 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 are left open at all possible reinterpretation sites. The actual reinterpretation step simply is further specialization, i.e instantiation of 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 looks, oversimplifying a bit, as in (5).

(5) $\exists x.book(x) \wedge begin(peter, \ \ldots \ x \ \ldots)$

The expression (5) can be seen as a description of both formulas (3) and (4), since the gap in (5) can be filled with the identity or the reinterpretation operator $read(peter, \bullet)$.

A key advantage of Egg's approach to reinterpretation is that it is compatible with an underspecified treatment of semantics.[1] Underspecification is a general approach to coping with ambiguity that has recently found wide attention in formal semantics (van Deemter and Peters, 1996). Its main idea is to represent all readings of an ambiguous sentence by one compact description. The readings can be extracted from the description, but this step is delayed as long as possible.

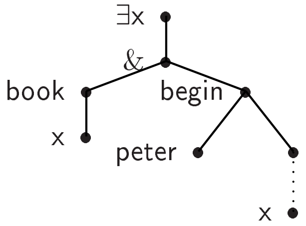


Figure 1: A tree description for (5).

Tree descriptions based on dominance constraints are powerful underspecified representations (Egg et al., 1998; Muskens, 1995). The idea is to consider a formula of some object language – e.g. predicate logic – as a tree which in turn can be described in some metalanguage. For example, the relaxed semantic representation (5) corresponds to the tree description in Figure 1. This graph describes all trees that can be formed from it by inserting arbitrary subtrees (of solid edges) into the gap left open by the dotted edge. This dotted edge is a *dominance constraint*; it expresses that the lower node must be somewhere below the upper one in the tree.

Scope underspecification and sort conflicts can be present in the same sentence, as shown by Example (6).

(6) Every student began a book.

According to Egg's analysis, an underspecified representation of the semantics of (6) would

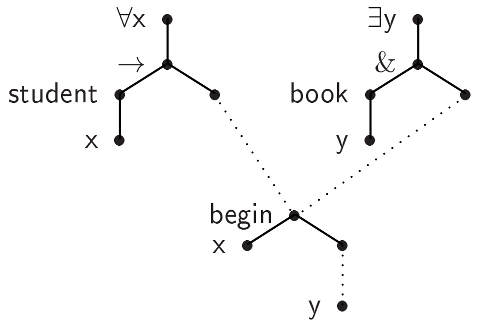


Figure 2: Underspecified representation for (6)

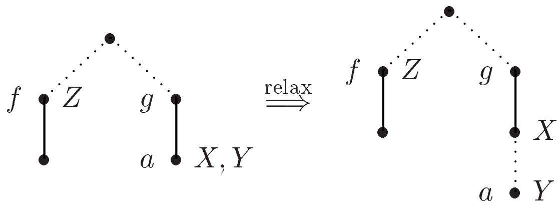


Figure 3: An unsafe constraint and its relaxation.

be as in Figure 2. The graph accounts for the scope ambiguity by leaving the relative position of the two quantifiers open, in this way describing a set of trees in each of which the scoping is fixed in one way or the other. The representation in Figure 2 is *relaxed* at the reinterpretation site (at the verb) such that the reinterpretation operator can be added even before resolving the scope relation.

There is, however, a potential problem with relaxing underspecified representations. Relaxation may lead to overgeneration, as illustrated in the abstract example in Figure 3. The graph on the left-hand side describes those trees that contain some node labeled with f and a subtree of the form $g(a)$. There is a unique minimal tree satisfying this description, which can be written as $f(g(a))$. The graph on the right-hand side of Figure 3 is a relaxation which is obtained by opening a dominance gap between X and Y. The relaxed graph describes more trees than the original graph, as is the purpose of relaxation. However, some of the trees described are not intended, e.g. $g(f(a))$. Here, the newly opened gap is filled by material already present in the original graph, but opening the gap was meant to open space just for the reinterpretation op-

[1] Another advantage of Egg's analysis is that it can account for so-called *landing site coercions* in a straightforward way.

erator.

In this paper, we investigate the overgeneration problem within the framework of CLLS (Egg et al., 1998; Niehren and Koller, 1998), which provides tree descriptions subsuming dominance constraints (Marcus et al., 1983; Vijay-Shanker, 1992). We present a safety criterion which, if satisfied, ensures that an underspecified description can be relaxed without adding unwanted readings. We then apply this criterion systematically to Egg's analysis and show that his approach avoids overgeneration. To this end, we present a toy grammar and a syntax/semantics interface which computes relaxed underspecified representations, and prove that all relaxations are safe by applying our safety criterion.

As a convenient tool for verifying that the Safety Criterion applies, we will introduce the notions of *fragments* and *chains of fragments* in a dominance constraint. Chains are subconstraints with very pleasant structural properties. As an additional example of their expressive power, we also use them to show that every constraint produced by the grammar is satisfiable.

Plan of the paper. Section 2 introduces tree descriptions in CLLS; we apply the formalism to some examples in Section 3. In Section 4, we formalize the notions of relaxation and safety, and state the Safety Criterion. In Section 5, we define chains and fragments and use them to derive some techniques for proving the Safety Criterion, which we will illustrate by means of an example in Section 6. Section 7 presents a grammar fragment with a syntax-semantics interface for deriving relaxed underspecified representations, and an outline of the proof of their safety. For omitted proofs, we globally refer to the long version of this paper (Koller et al., 1999).

2 Tree descriptions in CLLS

We now give a formal introduction to CLLS, the underspecification formalism we employ in this paper. As object language, we now use higher-order rather than first-order predicate logic.

CLLS, the Constraint Language over Lambda Structures (Egg et al., 1998; Niehren and Koller, 1998; Koller et al., 1998; Koller, 1999), is a language of tree descriptions. The trees described encode λ-terms of higher-order logic. We call these trees λ-*structures* and consider them to be standard first-order model structures. At the heart of CLLS is the language of dominance constraints, which has been used for various purposes throughout linguistics (Marcus et al., 1983; Vijay-Shanker, 1992; Gardent and Webber, 1998). With a varying degree of explicity, it has been used especially for representing scope ambiguities (Reyle, 1993; Bos, 1996; Muskens, 1995). For the purposes of this paper, we confine ourselves to the sublanguage of CLLS consisting only of dominance and λ-binding constraints, but note in passing that the full language also provides *parallelism* and *anaphora* constraints; for details, see (Egg et al., 1998).

Figure 4 illustrates the levels of abstraction that CLLS distinguishes by means of an example: On its semantical side, the basic object is the λ-structure, which uniquely represents a λ-term modulo renaming of variables. On the syntactic side, there are *constraints* – the formulae of CLLS – which can be written in a more perspicuous way as *constraint graphs*.

2.1 Lambda Structures

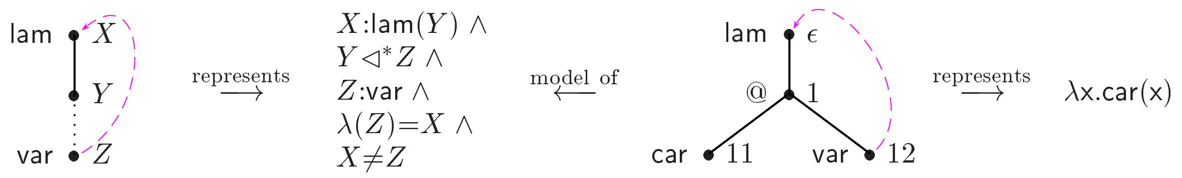
A λ-structure is a tree structure extended with a partial λ-binding function. It can be drawn as a tree-like graph with dashed arrows representing λ-binding (see e.g. Figure 4c).

For the definition of λ-structures, we assume a *signature* $\Sigma = \{@_{|2}, \mathsf{lam}_{|1}, \mathsf{var}_{|0}, \mathsf{car}_{|0}, \dots\}$ of *node labels*, each of which is equipped with a fixed *arity* $n \geq 0$. The labels lam, var, and @ (application) are used to model λ-terms. Node labels are ranged over by f, g, a, b, and the arity of a label f is denoted by $\mathsf{ar}(f)$; i.e. if $f_{|n} \in \Sigma$ then $\mathsf{ar}(f) = n$.

Let $\mathbb{N}$ be the set of natural numbers $n \geq 1$. As usual, we write $\mathbb{N}^*$ for the set of words over $\mathbb{N}$, ϵ for the empty word, and $\pi\pi'$ for the concatenation of two words $\pi, \pi' \in \mathbb{N}^*$. A word π is a *prefix* of π' if there is a word π'' such that $\pi\pi'' = \pi'$.

A *node* of a tree is the word $\pi \in \mathbb{N}^*$ which addresses the node. The empty word $\epsilon \in \mathbb{N}^*$ is called the *root* node. A *tree domain* Δ is a nonempty, prefixed-closed subset of $\mathbb{N}^*$, which is closed under the left-sibling relation. We say that a node π *dominates* a node π' and write $\pi \triangleleft^* \pi'$ iff π is a prefix of π'.

Definition 1. A *tree structure* is a tuple (Δ, σ)



(a) constraint graph (b) constraint (c) λ-structure (d) λ-term

Figure 4: Levels of representation in CLLS.

consisting of a finite tree domain Δ and a total *labeling function* $\sigma : \Delta \to \Sigma$ such that for all $\pi \in \Delta$ and $i \in \mathbb{N}$:

$$\pi i \in \Delta \;\Leftrightarrow\; 1 \leq i \leq \mathsf{ar}(\sigma(\pi)).$$

Definition 2. A *λ-structure* is a triple $(\Delta, \sigma, \lambda)$ consisting of a tree structure (Δ, σ) and a partial *λ-binding function* $\lambda : \Delta \rightsquigarrow \Delta$ which satisfies for all $\pi, \pi' \in \Delta$:

$$\lambda(\pi) = \pi' \text{ then } \begin{cases} \sigma(\pi) = \mathsf{var},\; \sigma(\pi') = \mathsf{lam}, \\ \text{and } \pi' \lhd^* \pi \end{cases}$$

As an example, consider the λ-structure depicted in Figure 4c which modulo renaming of bound variables corresponds uniquely to the λ-term in Figure 4d. The tree domain of this λ-structure is $\{\epsilon, 1, 11, 12\}$, its λ-binding function is given by $\lambda(12) = \epsilon$, and its labeling function is defined by $\sigma(\epsilon) = \mathsf{lam}$, $\sigma(1) = @$, $\sigma(11) = \mathsf{car}$, and $\sigma(12) = \mathsf{var}$.

2.2 A Fragment of CLLS

Now we define the syntax and semantics of that fragment of CLLS which contains labeling, dominance, and λ-binding constraints.

CLLS is a language which talks about relations between nodes of a λ-structure. Here we consider dominance $\pi \lhd^* \pi'$, λ-binding $\lambda(\pi) = \pi'$, and inequality $\pi \neq \pi'$. Finally, for every label f in the signature Σ with $\mathsf{ar}(f) = n$, there is a labeling relation $\pi{:}f(\pi_1, \ldots, \pi_n)$ which means that π is labeled with f and has the immediate successors $\pi_1, \ldots, \pi_n$. More formally, $\pi{:}f(\pi_1, \ldots, \pi_n)$ holds in a tree structure (Δ, σ) iff $\sigma(\pi) = f$ and $\pi_i = \pi i$ for all $1 \leq i \leq n$.

We assume an infinite set of (node) variables ranged over by X, Y, Z, U, V, W. Node variables should not be confused with variables x, y, z of λ-terms or predicate logic formulas. The syntax of the fragment of CLLS we consider is defined in Figure 5. 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 say that φ' *in* φ if all atomic constraints of φ' are also contained in φ. The set of (free) node variables of a constraint φ is denoted by $\mathcal{V}(\varphi)$. The semantics of a constraint in CLLS is fixed by interpretation over the class of λ-structures; these provide relations for the interpretation of all relation symbols in constraints. Note that we use the same symbol for a relation and the corresponding relation symbol; they can be distinguished by context (being applied to either paths π or variables X).

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

Figure 5: Syntax of a fragment of CLLS.

A *variable assignment into a λ-structure $\mathcal{M}$* is a total function from the set of variables to the domain of $\mathcal{M}$. A pair $(\mathcal{M}, \alpha)$ of a λ-structure $\mathcal{M}$ and a variable assignment α into $\mathcal{M}$ *satisfies* a constraint φ iff it satisfies all of its atomic constraints (in the obvious way). We also call the pair $(\mathcal{M}, \alpha)$ a *solution* and $\mathcal{M}$ a *model* of φ. In Figure 4, the λ-structure (c) together with a variable assignment α that maps X to ϵ, Y to 1, and Z to 12 satisfies the constraint (b).

We write $\varphi \models \varphi'$ and say that φ *entails* φ' if every solution of φ is a solution of φ'. The notions of solutions and entailment can be lifted to first-order formulae built from constraints as usual.

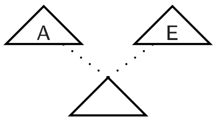


Figure 6: Schematic representation of a scope ambiguity.

2.3 Constraint Graphs

For underspecified descriptions, we will only use *normal* constraints φ, which are distinguished by requiring that any two labeled variables must denote distinct nodes.

Definition 3. A constraint φ is *normal* if for each two distinct occurrences of labeling constraints $X{:}f(X_1, \ldots, X_n)$ and $Y{:}g(Y_1, \ldots, Y_m)$ in φ it holds that $X{\neq}Y$ *in* φ (even if the same labeling constraint occurs twice in φ).

We will usually draw normal constraints as *constraint graphs*. The nodes of such a graph stand for variables in a normal constraint and, despite the similarity, should not be confused with the nodes in a λ-structure. An example for a normal constraint and its graph is given in Figure 4 (a) and (b). In the graph metaphor, normality means that any two labeled nodes in a graph must denote different nodes; "rigid fragments" of constraint graphs, i.e. parts which are connected by solid edges, must not overlap.

3 Underspecified Semantic Description in CLLS

We now discuss examples which illustrate the usage of CLLS in semantic underspecification.

3.1 Scope Underspecification

First, we consider the analysis of scope ambiguities, introducing some terminology along the way.

As in the introduction, a scope ambiguity is characterized by containing two or more quantifiers whose relative scope is not fixed. A constraint graph accounting for this fact typically has a structure as shown in the schematic representation in Figure 6. The triangles in this picture denote tree *fragments*, a fragment being a set of nodes which are connected by solid edges. Each fragment has a unique *root* which must dominate all other nodes in the fragment. Leaves, which are not labeled, are called *holes*. Since graphs represent *normal* constraints, two fragments in a graph can only overlap by identifying a root with a hole. A formal definition of fragments is crucial to this paper and it will be given in Section 5 for now the intuitive idea is sufficient.

In Figure 6 the two triangles annotated by A and E denote the fragments for the two quantifiers. The scope ambiguitiy is accounted for by imposing the constraint that they must dominate a common node. 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.

For the linguistic application, the reading of a sentence is usually represented by a solution which is constructed from the "material" mentioned in the underspecified semantic representation. As a matter of fact, most underspecification formalisms except CLLS assume this in general (Reyle, 1993; Muskens, 1995; Bos, 1996; Dalrymple et al., 1997). CLLS does not make this assumption because it is not strictly correct for ellipses and reinterpretation, but we can incorporate the concept by restricting ourselves to so-called constructive solutions: A solution $(\mathcal{M}, \alpha)$ of a constraint φ is *constructive* if for every node π in the domain of $\mathcal{M}$ there exists a variable $X \in \mathcal{V}(\varphi)$ such that $\alpha(X) = \pi$ and $X{:}f(\ldots)$ *in* φ for some label f in Σ.

3.2 Scope and Reinterpretation

We next investigate a more complex example, which contains both a scope ambiguity and a type conflict, as in (6) in the introduction. Consider sentence (7), whose semantics is described by the constraint graph in Figure 7.

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

In reading the constraint graph in Figure 7, 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". Next, note that the graph exhibits the same schematic structure as in Figure 6. So as above, the scope ambiguity is correctly modeled: We have expressed that one quantifier has to go on top, but we haven't said which one. Finally, note that there is no unintended interaction between scope ambiguity and reinterpre-

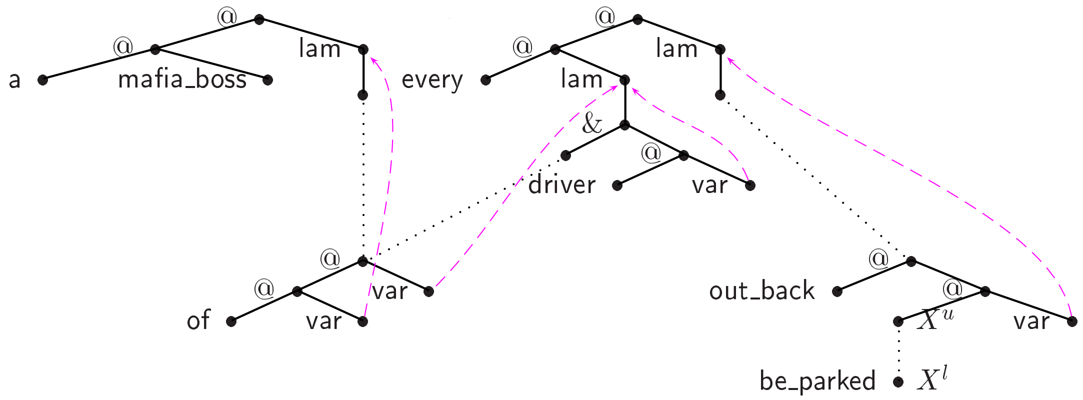


Figure 7: Relaxed underspecified representation of Example (7).

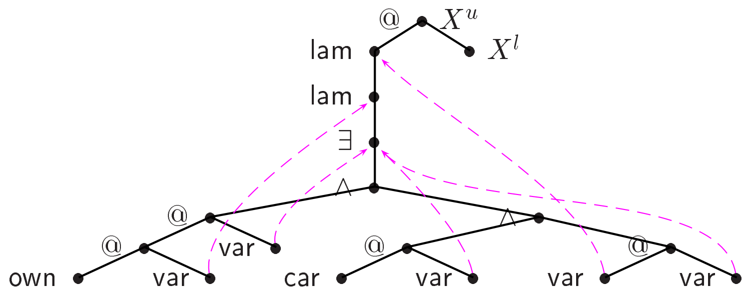


Figure 8: The reinterpretation operator for Figure 7.

tation: none of the fragments for the quantifiers may be moved into the gap $X^u \triangleleft^* X^l$.

Reinterpretation has to coerce "every driver of a mafia boss" into their vehicles. We can do this by filling the appropriate reinterpretation operator (given in Figure 8) into the gap left open by the dominance edge $X^u \triangleleft^* X^l$ in the description of the verb semantics.

4 The Safety Criterion

Now that we have a definition of our underspecification formalism, we can turn to making the notions of "relaxation", "intended solution", and "safety" precise. After introducing some more terminology (disjointness), we then formulate the *Safety Criterion* in the last part of the section, which can be used to verify safety and is stated in purely logical terms.

4.1 Constraint Relaxation

First of all, the operation of constraint relaxation is based on the idea of splitting a node in the graph in two and adding a dominance edge between the two new nodes. An example is drawn in Figure 9, where we split the node corresponding to the variable X into two new nodes for the variables X^u and X^l.

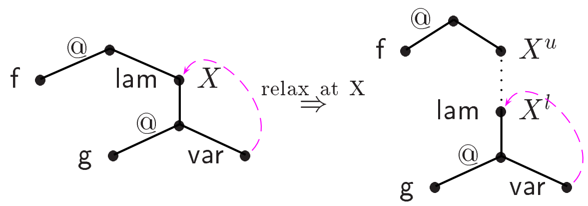


Figure 9: An example of relaxation.

We now define relaxation formally. Let φ be a normal constraint, $X \in \mathcal{V}(\varphi)$ a variable, and $\mathsf{eq}_\varphi(X) = \{Y \in \mathcal{V}(\varphi) \mid \varphi \models X \triangleleft^* Y \wedge Y \triangleleft^* X\}$ the set of variables which must be equal to X. In a satisfiable normal constraint, we can compute $\mathsf{eq}_\varphi(X)$ syntactically by reflexivity and transitivity inferences about dominance constraints.

We partition the set of atomic constraints in φ into three parts, corresponding to edges in the graph of φ involving a variable in $\mathsf{eq}_\varphi(X)$ as the upper node, the lower node, or not at all. The set $U_X(\varphi)$ contains all atomic constraints in φ that relate a $Z \in \mathsf{eq}_\varphi(X)$ to a variable above, i.e. that are of one of the following forms (for some f, Y):

$$Y{:}f(\ldots, Z, \ldots),\ Y \lhd^* Z,$$
$$\lambda(Z){=}Y, Z{\neq}Y, \text{ or } Y{\neq}Z$$

The set $L_X(\varphi)$ contains all atomic constraints in φ that relate a $Z \in \mathsf{eq}_\varphi(X)$ to some variable below, which are of the forms:

$$Z{:}f(\ldots, Y, \ldots),\ Z \lhd^* Y,$$
$$\lambda(Y){=}Z, Z{\neq}Y, \text{ or } Y{\neq}Z$$

The set $O_X(\varphi)$ contains those atomic constraints of φ in which no variable of $\mathsf{eq}_\varphi(X)$ occurs.

In the following definition, we write $[Y/\mathcal{V}]$ for the substitution that maps all variables in the variable set $\mathcal{V}$ to the variable Y.

Definition 4. Let φ be a normal constraint, X a variable in $\mathcal{V}(\varphi)$ such that a labeling constraint $X{:}f(\ldots)$ *in* φ, and X^u, X^l variables fresh for φ. Then the *relaxation* $\mathcal{R}_X(\varphi)$ of a constraint φ at the variable X is defined as the conjunction

$$\begin{aligned} X^u \lhd^* X^l \quad & \wedge \bigwedge\{\mu[X^u/\mathsf{eq}_\varphi(X)] \mid \mu \in U_X(\varphi)\} \\ & \wedge \bigwedge\{\mu[X^l/\mathsf{eq}_\varphi(X)] \mid \mu \in L_X(\varphi)\} \\ & \wedge \bigwedge\{\mu \mid \mu \in O_X(\varphi)\}. \end{aligned}$$

Lemma 5. *The relaxation $\mathcal{R}_X(\varphi)$ of a normal constraint φ at X is normal.*

4.2 Intended Solutions and Safety

The point we made in the introduction (Figure 3) was that sometimes, relaxation can produce unintended solutions, and that the overgeneration thus caused is undesirable. What we mean by an *intended solution* is a solution of a relaxed graph whose overall structure is the same as a solution for the original graph. New solutions of the relaxed graphs should only arise from introducing new material, i.e. labeling constraints not present in the original graph, into the gap opened by relaxation. So the only place where an intended solution of a relaxed constraint should differ from a solution of the original constraint is at the relaxation site. By cutting away the part of a solution that is located between the upper and lower end of the gap, we should thus come back to a solution

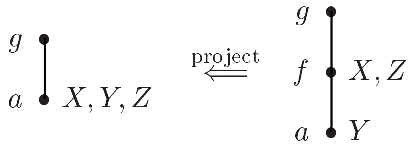


Figure 10: A solution of the relaxed constraint in Figure 3 and its projection

of the original constraint. We call this cutting-away operation *projection* and will define it below. It will be the foundation of our formal definition of intended solutions. *Safety* will then mean that all new solutions are intended.

For illustration, let's apply the new terminology to the constraint of Figure 3. As argued above, the tree $g(f(a))$ is a solution of the relaxed constraint (see Figure 10). This tree is an unintended solution because its projection at the relaxation site is $g(a)$, which is not a solution of the original constraint. Intuitively, the f-labeled node required by the original constraint has slipped into the new gap and disappeared in the projection. So the original constraint is not safe for relaxation at X.

Definition 6 (Projection). Let $\mathcal{M}$ be a λ-structure $(\Delta, \sigma, \lambda)$, and let $\pi_u \lhd^* \pi_l$ be nodes in Δ. Let $\Delta^{\pi_u}_{\pi_l} \subseteq \mathbb{N}^*$ be the subset of Δ without the tree fragment between π_u and π_l:

$$\Delta^{\pi_u}_{\pi_l} = \{\pi \in \Delta \mid \text{ if } \pi_u \lhd^* \pi \text{ then } \pi_l \lhd^* \pi\}$$

The projection $p : \Delta^{\pi_u}_{\pi_l} \to \mathbb{N}^*$ is the function which satisfies for all paths $\pi \in \Delta^{\pi_u}_{\pi_l}$:

$$p(\pi) = \begin{cases} \pi & \text{if not } \pi_u \lhd^* \pi \\ \pi_u \pi' & \text{if } \pi = \pi_l \pi' \end{cases}$$

Note that p is one-to-one and that $p(\Delta^{\pi_u}_{\pi_l})$ is a tree domain. The projection $\mathcal{M}^{\pi_u}_{\pi_l}$ of the λ-structure $\mathcal{M}$ at nodes π_u, π_l is the λ-structure $(p(\Delta^{\pi_u}_{\pi_l}), \sigma', \lambda')$ such that for all $\pi \in \Delta^{\pi_u}_{\pi_l}$:

$$\begin{aligned} \sigma'(p(\pi)) &= \sigma(\pi) \\ \lambda'(p(\pi)) &= \begin{cases} p(\lambda(\pi)) & \text{if } \lambda(\pi) \in \Delta^{\pi_u}_{\pi_l} \\ \text{undefined} & \text{otherwise.} \end{cases} \end{aligned}$$

The projection α' of a variable assignment into $\mathcal{M}$ at nodes π_u, π_l is defined iff

$$\alpha(Y) \in \Delta^{\pi_u}_{\pi_l} \text{ for all } Y \in \mathcal{V}(\varphi).$$

In this case, it is defined by $\alpha'(Y) = p(\alpha(Y))$ for all $Y \in \mathcal{V}(\varphi)$.

Definition 7 (Intended Solutions). Let φ be a constraint, X a variable and $\mathcal{R}_X(\varphi)$ the relaxation of φ at X. A pair $(\mathcal{M}', \alpha')$ is called an *intended solution* of $\mathcal{R}_X(\varphi)$ iff

1. the projection α of α' at $\alpha'(X^u), \alpha'(X^l)$ is defined;
2. $(\mathcal{M}^{\alpha'(X^u)}_{\alpha'(X^l)}, \alpha)$ satisfies φ.

Lemma 8. *An intended solution of $\mathcal{R}_X(\varphi)$ is indeed a solution of $\mathcal{R}_X(\varphi)$.*

Definition 9 (Safety). A constraint φ is called *safe at X* iff all solutions of its relaxation $\mathcal{R}_X(\varphi)$ at X are intended.

4.3 Disjointness

To state the Safety Criterion, we will need to be able to speak about another relation between nodes of a tree, namely *disjointness*. First, we define that nodes π_1, π_2, π_0 are in the *disjointness relation*

$$\pi_1 \perp \pi_2 \text{ at } \pi_0$$

iff there are paths π_1', π_2' and different positive integers i, k such that:

$$\begin{array}{rcl} \pi_1 & = & \pi_0 i \pi_1' \\ \pi_2 & = & \pi_0 k \pi_2' \ . \end{array}$$

Informally speaking, π_1 and π_2 should not dominate each other, and π_0 (called *branching point*) should be the lowest node that dominates them both. The *disjointness formula*

$$X \perp Y \text{ at } \{Z_1, \ldots, Z_n\}$$

is satisfied by a pair $(\mathcal{M}, \alpha)$ iff there is an $1 \leq i \leq n$ such that $\alpha(X) \perp \alpha(Y)$ at $\alpha(Z_i)$. We abbreviate this to $X \perp Y$ if we do not care about the branching point. The following entailment holds of disjointness:

$$\begin{array}{l} Z{:}f(\ldots, X', \ldots, Y', \ldots) \ \wedge \ X' \lhd^* X \ \wedge \ Y' \lhd^* Y \\ \models X \perp Y \text{ at } \{Z\}. \end{array}$$

If we restrict ourselves to a finite signature and allow disjunctions, this formula can easily be extended to an axiomatization of disjointness.

Note that for any two paths π_1, π_2, either $\pi_1 \lhd^* \pi_2$, $\pi_2 \lhd^* \pi_1$, or $\pi_1 \perp \pi_2$.

4.4 The Safety Criterion

Finally, we are ready to express the Safety Criterion, a purely logical characterization of safety.

Proposition 10 (Safety Criterion). *Let φ be a constraint and X a variable in φ. Then φ is safe at X if the following entailment is true:*

$$\mathcal{R}_X(\varphi) \models \bigwedge_{Y \in \mathcal{V}(\varphi) \setminus \{X\}} (Y \lhd^* X^u \vee Y \perp X^u \vee X^l \lhd^* Y).$$

For instance, the Safety Criterion is satisfied by the (safe) constraint in Figure 7; it is not satisfied by the (unsafe) constraint in Figure 3 because Z can be mapped into the gap between X and Y.

Note that this criterion does not state an "if-and-only-if" relation. However, it is still strong enough to cover all practically relevant cases. We will derive techniques for proving its satisfiedness in the next section and then apply them to show that the Safety Criterion holds for all constraints generated by a simple grammar and all possible reinterpretation sites in these constraints in Section 7.

5 Proving Safety via Chains of Fragments

This section discusses some techniques for proving satisfiedness of the Safety Criterion. We define the notion of a fragment and introduce *chains* of fragments as a key concept. Chains allow powerful inferences about dominance and disjointness.

First, we note the following "quasi-transitivity" result about disjointness, which is useful in the proofs of this section.

Proposition 11. *If $\mathcal{V}$ and $\mathcal{W}$ are sets of variables, the following entailment is true:*

$$\begin{array}{l} X \perp Y \text{ at } \mathcal{V} \ \wedge \ Y \perp Z \text{ at } \mathcal{W} \ \wedge \ \bigwedge_{V \in \mathcal{V}, W \in \mathcal{W}} V \neq W \\ \models \ X \perp Z \text{ at } \mathcal{V} \cup \mathcal{W} \end{array}$$

Definition 12 (Fragments). Let φ be a normal CLLS constraint.

1. *Connectedness* in φ is the smallest binary equivalence relation between variables in φ which contains all pairs (X, Y) such that $X{:}f(\ldots Y \ldots)$ *in* φ.

2. A *fragment* of φ is a subset $F \subseteq \mathcal{V}(\varphi)$ of variables that are pairwise connected in φ. A node $X \in F$ is called a *leaf* of a fragment F if F contains no variable Y such that there is a labeling constraint $X{:}f(\dots Y \dots)$ in φ. A *h*ole is a leaf which is not labeled at all.

It is easily shown that every fragment F has a unique *root*, i.e. contains a variable that must dominate all other members of F in any solution, and that all of its leaves must always denote pairwise disjoint nodes.

We are usually interested in maximal fragments; however, maximality does not matter for the proofs below.

Fragments can be composed into *chains*. A chain is intuitively a construction as in Figure 11, where fragments (drawn as triangles) are connected via dominance constraints between leaves of the upper fragments and the roots of the lower fragments, which are called *connection points* of the chain. For example, Figure 7 contains a chain of length 2, i.e. the chain contains two upper fragments (corresponding to *a mafia boss* and *every driver*) and one lower fragment (corresponding to the preposition). Below, we define chains; afterwards, we state a theorem that will help us verify the Safety Criterion for our grammar, along with a sketch of its proof.

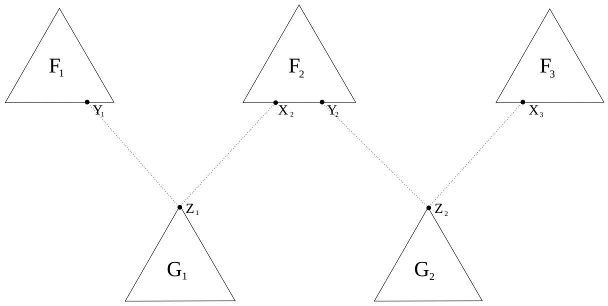


Figure 11: A chain of fragments.

Definition 13 (Chains). Let φ be a normal 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 such that no variable appears in two different fragments. For all i, let X_i, Y_i be different 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 iff for all $1 \leq i \leq n-1$,

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

The variables $X_1, Z_1, \dots, Z_{n-1}, Y_n$ are called the *connection points* of $\mathcal{C}$ and n its *length.*

Now we can formulate certain propositions about the respective positions of two variables belonging to different fragments of the same chain.

Proposition 14. *Let $\mathcal{C} = (\mathcal{F}, \mathcal{G})$ be a chain of length n in φ, and let Z_i, Z_k $(0 \leq i < k \leq n)$ be connection points of $\mathcal{C}$. Then*

$$\varphi \models Z_i \perp Z_k \text{ at } \{V_{i+1}, \dots, V_k\},$$

where the $V_j \in F_j$.

Theorem 15. *Let $\mathcal{C} = (\mathcal{F}, \mathcal{G})$ be a chain in φ, let $F \in \mathcal{F}$ and $G \in \mathcal{G}$, and let Z be the root of G. Then for all $X \in F$,*

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

Proof. Let i be the index of F in $\mathcal{F}$, and let k be the index of G in $\mathcal{G}$. The claim follows immediately for $k \in \{i-1, i\}$.

For the other cases, it can be shown by contradiction: We assume that a pair $(\mathcal{M}, \alpha)$ satisfies both φ and $Z \triangleleft^* X$. Under this assumption, we distinguish cases as to whether $\alpha(Z) \neq \alpha(X)$ or $\alpha(Z) = \alpha(X)$.

The first case is easy. As fragments can only overlap at leaves, $\alpha(Z)$ must be a prefix of the root of F; hence, it follows that $\alpha(Z) \triangleleft^* \alpha(Z')$ for one of the connection points Z' below F; Z' is not Z, as $k \notin \{i-1, i\}$. Now we can show (although we will not do so here) that all distinct connection points of a chain must denote disjoint nodes. Hence $\alpha(Z) \perp \alpha(Z')$, in contradication to $\alpha(Z) \triangleleft^* \alpha(Z')$.

In the second case, we can further distinguish whether i is larger or smaller than k. The two cases can be handled analogously, so we only consider the case $i > k$ here. In this case, we pick $F' \in \mathcal{F}$ to be the fragment whose Y daughter dominates Z. Because fragments can only overlap at leaves, this implies that the root of F' dominates the root of F in $(\mathcal{M}, \alpha)$. But we can also show that in such a situation, the X daughter of F' must dominate the root of F, which contradicts the disjointness of leaves of F'. $\square$

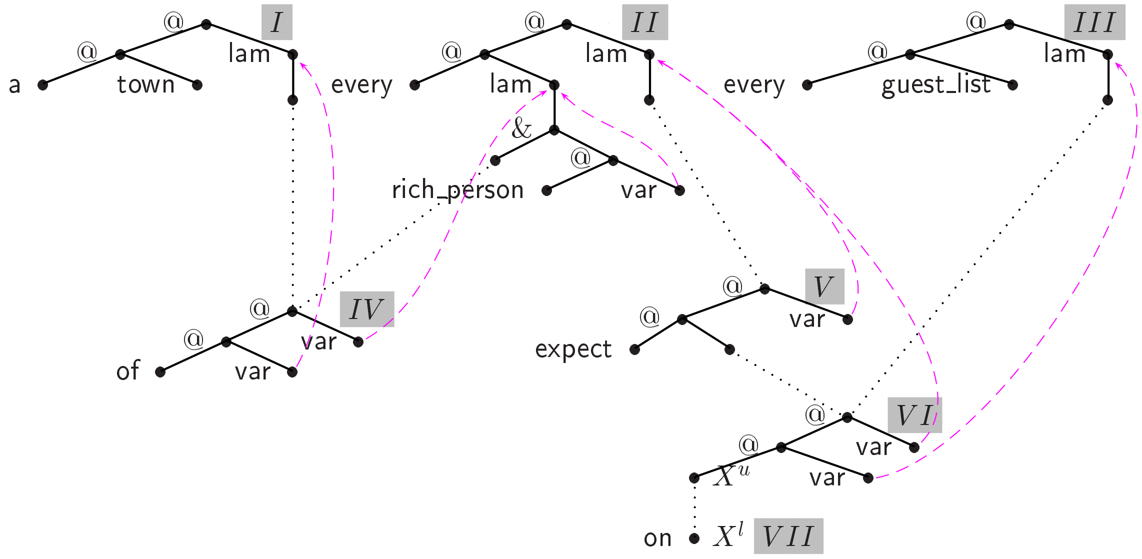


Figure 12: *Every rich person of a town expects to be on every guest list.*

6 An Example of Using Chains to Prove Safety

Now, we will demonstrate by means of an example how the techniques of the previous section can be applied to prove that the relaxation of a constraint at a specific relaxation point is safe. To this end, consider the following (pretty complex) example, whose semantic representation is as in Figure 12.

(8) Every rich person of a town expects to be on every guest list.

We can distinguish seven fragments in the representation, which we have labeled with Roman numbers for convenience. The three nominal phrases *every rich person*, *a town*, and *every guest* are represented by the three fragments in the top row, namely, I, II, and III. The scope ambiguity between them is accounted for by leaving their relative ordering open; but it is restricted by specifying that fragment IV must be dominated by fragments I and II, and fragment VI by fragments II and III, respectively. This yields exactly the five scope orderings which are predicted for this sentence.

Reinterpretation is needed in this example because it is not the person himself who is appearing on the guest list, but the person's name. To account for this, we have relaxed the original semantic description to the constraint in Figure 12; the new gap introduced by relaxation is between the variables X^u and X^l. As a consequence, X^l has become a seventh fragment by itself.

Finally, the control verb *expects* is represented as fragment V. A schematic view of the structure of this constraint is given in Figure 13.

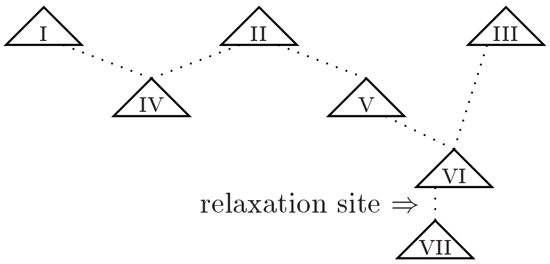


Figure 13: Schematic view of the Example.

We now want to show (informally) that it was safe to do this relaxation. What we need to prove, according to the Safety Criterion, is that every variable in the constraint either dominates or is disjoint to X^u or is dominated by X^l. In our case, the only variable dominated by X^l is X^l itself, so we will show that all other variables either dominate X^u or are disjoint to X^u. We do this by covering the entire constraint with chains in which fragment VI takes part as a lower fragment. Then we can apply results from the previous section: By Proposition 14, variables belonging to different lower fragments of a chain are pairwise disjoint, and Theorem 15 means that all variables belonging to upper fragments are either disjoint from or dominate all variables belonging to lower fragments.

Now there are two different chains in Figure 12 in which fragment VI takes part. The first of those has the upper fragments I, II, and III and the lower fragments IV and VI. The second one has the upper fragments III and V and the lower fragment VI.

As X^u is dominated by the root of fragment VI, this implies that all variables in fragments other than VI are disjoint to X^u or dominate it (by one of the two results from the previous section mentioned above); the same is trivially true of the variables in fragment VI. So we have shown that all variables in the semantic representation except for X^l must be either disjoint to X^u or dominate it. Hence, the Safety Criterion is satisfied here.

Note how we had to employ two chains, each of them covering only part of the representation, to get the desired result. Taken together, however, they covered the whole constraint (with the exception of fragment VII, i.e. node X^l) in such a way that we were able to conclude something about every variable in the representation. This technique of intelligently distinguishing chains in an underspecified semantic representation is the basis for our proofs in the next section.

7 Correctness of Underspecified Reinterpretation

In this section, we finally present a grammar with a syntax/semantics interface which can produce constraints that can (but needn't necessarily) be relaxed at any node that carries the semantics of a verb. We state the result that in any constraint generated by the grammar, the Safety Criterion holds for relaxation at every single unrelaxed verb node. By multiple application of this theorem, it follows that even the completely relaxed version of the constraint has only intended solutions with respect to the completely unrelaxed version. In the last part of the section, we also sketch a proof for the *satisfiability* of all generated constraints. This result is the bare minimum for linguistic adequacy of our formalism and is presented here as an additional example for the range of applications of the notions from Section 5.

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

Figure 14: The grammar

The grammar fragment we consider is displayed in Figure 14 (where IV = intransitive verb; TV = transitive verb; CV = control verb). Lex is a relation between words W and lexical categories $\alpha \in \{\mathtt{Det}, \mathtt{N}, \mathtt{IV}, \mathtt{TV}, \mathtt{CV}, \mathtt{Adv}\}$ 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; as a matter of fact, we have implemented an HPSG grammar producing the same constraints that does care about these things.

The syntax/semantics interface of the grammar associates subconstraints with each node ν of the parse tree. The contributions of these are then conjoined. The rules by which subconstraints are introduced are presented in Figure 15. We take $[_{\nu:P}\ Q\ R]$ to mean that node ν in the syntax tree is labeled with P, and its two daughter nodes $\nu 1$ and $\nu 2$ are labeled with Q and R, respectively. The constraint introduced by such a rule then imposes a CLLS constraint on the variables $X_\nu, X_{\nu 1}, X_{\nu 2}$, which are the roots of the fragments contributed by nodes ν, $\nu 1$, and $\nu 2$ respectively.

Some nodes are given a special name, e.g. X_ν^{scope} in the fragment introduced by rule (b1). This is to make it easier to refer to them later on. Furthermore, it clarifies their function in the final constraint; in the example, X_ν^{scope} is intuitively the scope of the quantifier represented by the NP.

Figure 15 doesn't completely specify all the

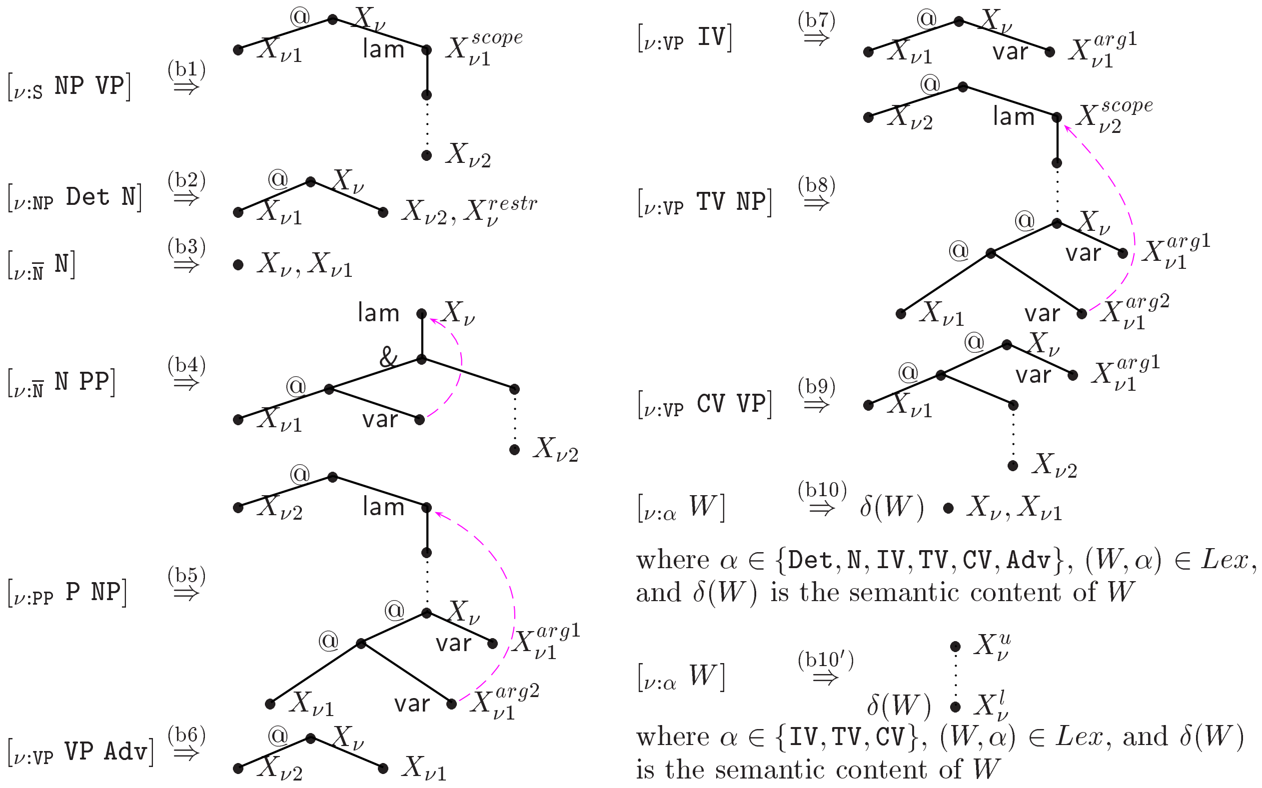


Figure 15: The syntax/semantics interface

necessary λ-binding constraints. This is because variable binding sometimes requires contextual information which could be easily maintained in a unification-based grammar formalism. For the rules (b7), (b8), and (b9), we therefore assume the availability of additional information about the subjects of the verbs considered. 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:

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

Next, 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 all 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. This choice can be made nondeterministically. However, in "real" semantic construction, we will always apply (b10') to verbs; that is, we produce maximally relaxed constraints. We keep (b10) for verbs so we can formulate Theorem 7 below. If the semantics for a syntactic verb node ν is constructed with (b10), we call X_ν a (potential) *unrelaxed reinterpretation site*; if it is constructed with (b10'), we call X_ν^u a (potential) *relaxed interpretation site*.

To give an impression of how semantic construction works, we return briefly to Example (7), whose (relaxed) underspecified semantic representation we showed in Figure 7. The parse tree that the grammar assigns to this sentence is shown in Figure 16. Consider, by way of example, the semantic construction for the lower `NP` node ("a mafia boss") in the parse tree, which will give rise to the top left fragment in Figure 7. First of all, we apply Rule (b2), which represents the application of the determiner "a" to the restriction "mafia boss". Technically, this is done by introducing a single

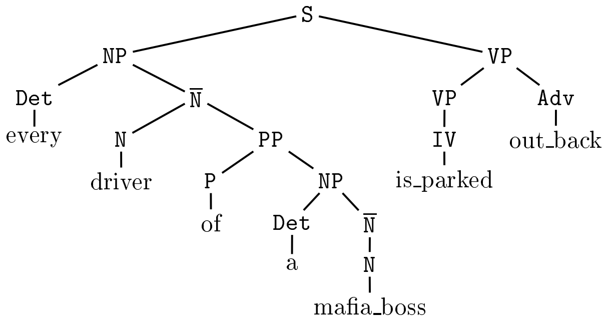


Figure 16: Every driver of a mafia boss is parked out back.

node into the constraint graph, labeled with @. By Rule (b10), we add the determiner semantics $\delta(\text{“a”}) = \mathsf{a}$ as a label at the node $X_{\nu 1}$. Then we apply Rule (b3), which only identifies the nodes corresponding to the $\overline{\texttt{N}}$ and the `N`. Finally, we can again apply Rule (b10) to add the noun semantics $\delta(\text{“mafia boss”}) = \mathsf{mafia\_boss}$ at the correct node. The additional application and abstraction nodes that belong to the fragment in Figure 7 are introduced earlier by Rule (b5).

Now we are ready to state the main result for this grammar and its syntax/semantics interface:

Theorem 16. *Let φ be a constraint which was constructed by the syntax/semantics interface of the example grammar, and let X_ν be an unrelaxed potential interpretation site in φ. Then the Safety Criterion is satisfied for the relaxation of φ at X_ν.*

Assume that φ_0 is a constraint that was constructed from a parse tree without using rule (b10'), but possibly (b10). Using this theorem, we can replace applications of (b10) to verbs by applications of (b10') one by one; in each step, the relaxation is safe. After a finite number of steps, we have obtained a constraint for whose construction (b10) wasn't used, and we know that all of its solutions are intended with respect to φ_0. That is, we never need to worry whether we can safely reinterpret if we want to.

The main proof idea is to distinguish, given a potential reinterpretation site, chains of fragments that cover the constraint and that have the fragment containing the reinterpretation site as a lower fragment (as in the previous section). Then we can employ the results of Section 5 to prove satisfiedness of the Safety Criterion.

Here we only sketch one part of the proof to illustrate the flavour: namely, that the constraint corresponding to a nested `NP` is a chain. But before we do so, we will outline the general structure of the constraints our syntax/semantics interface produces, which should give an intuition about why the proof works out just fine for the rest of the syntax/semantic interface as well.

Look at the outline of a constraint displayed in Figure 17. The picture suggests that for each reinterpretation site in the displayed constraint we can cover the constraint with chains that have the fragment with the reinterpretation site as a lower fragment.

For our grammar, we can show that the structure of any constraint produced by the syntax/semantics interface is as in Figure 17. Noun phrases (in particular, subjects and objects) can either consist of just one determiner and a common noun, or they can be complex, i.e. the noun is modified by a prepositional phrase. In any case, their semantic contributions form a chain (possibly of length 1) with the contributions of the determiners and nouns as upper fragments and the contributions of the prepositions as lower fragments. A verb phrase in our example grammar can consist of zero or more `VP`-nesting control verbs and either a transitive verb or an intransitive verb at the bottom. The nesting structure of control verbs in syntax is mirrored by a nested structure in the semantic representation (as in the verb part of Figure 17). Finally, `VP`s can be modified by adverbs; but this makes no difference, as the processing of an adverb simply attaches additional material to the `VP` fragment. This overall structure of the constraints generated by our syntax/semantics interface suggests that Theorem should actually hold.

Now we will actually prove that the semantic contribution of an `NP` is a chain.

Proposition 17. *Let T be a parse tree of our grammar, and let φ be the constraint that our syntax/semantics interface assigns to it. 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 chain $\mathcal{C} = ((F_1, \ldots, F_n), (G_1, \ldots, \mathcal{G}_{n-1}))$ of length n cov-*

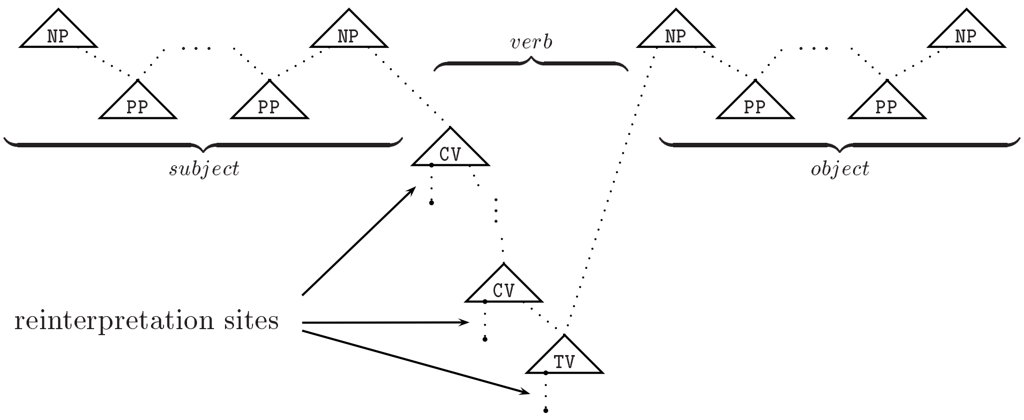


Figure 17: Output of the syntax/semantics interface: schematic view

ering φ:

$$\mathcal{V}(\varphi_t) = \bigcup_{F \in \mathcal{F}} F \cup \bigcup_{G \in \mathcal{G}} G.$$

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, so $\mathcal{C} = ((\mathcal{V}(\varphi_t)), ())$ is 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; then $t = \mathtt{NP}(\mathtt{N}(W_1)\ \mathtt{PP}(\mathtt{P}(W_2)\ t'))$, where W_1 and W_2 are again words, and t' contains $n - 1$ NP nodes.

By induction hypothesis, the semantics $\varphi_{t'}$ of t' is a chain $\mathcal{C}' = (\mathcal{F}', \mathcal{G}')$ of length $n - 1$ such that the fragments in $\mathcal{F}'$ and $\mathcal{G}'$ form a partition of $\mathcal{V}(\varphi_{t'})$; say, $\mathcal{F}' = (F'_1, \dots, F'_{n-1})$ and $\mathcal{G}' = (G'_1, \dots, G'_{n-2})$.

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 $(F_0, F_1, F'_2, \dots, F'_n)$ and $(G_0, G'_1, \dots, G'_{n-1})$ form a chain in φ. Its length is n, and as it contains all variables that the three rule applications introduced, its fragments contain all variables in φ_t. $\square$

As a final application, chains can also be used to prove the following result.

Proposition 18. *Every constraint generated by the grammar is satisfiable.*

The intuition is simple. First of all, it is not difficult to show that every chain is satisfiable. So we can replace the chains for the subject and (if it exists) object in a constraint φ that was generated by the grammar by fragments that fully describe the trees satisfying these chains. Furthermore, the "verb" section of the constraint is satisfiable as well; we replace it by a third fragment that fully describes a satisfying tree. Now φ', the result of these replacements, clearly entails φ; but it is also a chain and hence, satisfiable. So φ is satisfiable as well.

8 Conclusion

In this paper, we developed a general criterion for underspecified semantic representations which, if satisfied, ensures that relaxation at a certain position is safe. We utilised this notion of safe relaxation for the underspecified representation of reinterpretation phenomena, and showed that it is possible to build grammars in such a way that only semantic descriptions which are safe for reinterpretation are generated. To prove satisfiedness of the criterion for a given constraint, we defined *chains* of fragments, a very pleasant type of substructure of constraints which has applications independent of the one presented here.

Although in this paper, we applied the Safety Criterion only to a very specific example, namely reinterpretation within the semantics of verb phrases as treated in CLLS, we believe that it is not restricted to this application. CLLS could be replaced by other representation formalisms for semantic underspecification. Other types of reinterpretation could also be treated, e.g. reinterpretation within the semantics of noun phrases.

Finally, we would like to emphasis a different view point. In principle, Egg shows how to replace tree adjunction in traditional approaches to reinterpretation by instantiation of dominance constraints. The presented method based on *chains of fragments* allows to show in this case – and possibly others – that it is correct to substitute tree adjunction by using dominance constraints.

References

Manfred Bierwisch. 1983. Semantische und konzeptionelle Repräsentation lexikalischer Einheiten. In R. Ruzicka and W. Motsch, editors, *Untersuchungen zur Semantik*, pages 61–99. Akademie-Verlag, Berlin.

Johan Bos. 1996. Predicate logic unplugged. In *Proceedings of the 10th Amsterdam Colloquium*, pages 133–143.

A. Copestake and T. Briscoe. 1995. Semi-productive polysemy and sense extension. *Journal of Semantics*, 12:15–67.

Mary Dalrymple, John Lamping, Fernando Pereira, and Vijay Saraswat. 1997. Quantifiers, anaphora, and intensionality. *Journal of Logic, Language, and Information*, 6:219–273.

J. Dölling. 1994. Sortale Selektionsbeschränkungen und systematische Bedeutungsvariation. In M. Schwarz, editor, *Kognitive Semantik/Cognitive Semantics*, pages 41–59. Narr, Tübingen.

M. Egg, J. Niehren, P. Ruhrberg, and F. Xu. 1998. Constraints over Lambda-Structures in Semantic Underspecification. In *Proceedings COLING/ACL'98*, Montreal.

Markus Egg. 1999. Reinterpretation by underspecification. Submitted.

Claire Gardent and Bonnie Webber. 1998. Describing discourse semantics. In *Proceedings of the 4th TAG+ Workshop*, Philadelphia. University of Pennsylvania.

J. Hobbs, M. Stickel, D. Appelt, and P. Martin. 1993. Interpretation as abduction. *Artificial Intelligence*, 63:69–142.

Alexander Koller, Joachim Niehren, and Ralf Treinen. 1998. Dominance constraints: Algorithms and complexity. In *Proceedings of the 3st Conference on Logical Aspects of Computational Linguistics*.

Alexander Koller, Joachim Niehren, and Kristina Striegnitz. 1999. Relaxing underspecified semantic representations for reinterpretation. Long version. `http://www.coli.uni-sb.de/~koller/papers/reint.html`.

Alexander Koller. 1999. Constraint languages for semantic underspecification. Diplom thesis, Universität des Saarlandes, Saarbrücken, Germany. `http://www.coli.uni-sb.de/~koller/papers/da.html`.

Mitchell P. Marcus, Donald Hindle, and Margaret M. Fleck. 1983. D-theory: Talking about talking about trees. In *Proceedings of the 21st ACL*, pages 129–136.

R.A. Muskens. 1995. Order-Independence and Underspecification. In J. Groenendijk, editor, *Ellipsis, Underspecification, Events and More in Dynamic Semantics*. DYANA Deliverable R.2.2.C.

Joachim Niehren and Alexander Koller. 1998. Dominance Constraints in Context Unification. In *Proceedings of the 3st Conference on Logical Aspects of Computational Linguistics*.

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

James Pustejovsky. 1995. *The generative lexicon*. MIT Press, Cambridge.

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

Kees van Deemter and Stanley Peters. 1996. *Semantic Ambiguity and Underspecification*. CSLI, Stanford.

K. Vijay-Shanker. 1992. Using descriptions of trees in a tree adjoining grammar. *Computational Linguistics*, 18:481–518.