On Underspecified Processing of Dynamic Semantics

Alexander Koller, Joachim Niehren
University of the Saarland, Saarbrücken, Germany
`{koller@coli|niehren@ps}.uni-sb.de`

Abstract

We propose a new inference system which operates on underspecified semantic representations of scope and anaphora. This system exploits anaphoric accessibility conditions from dynamic semantics to disambiguate scope ambiguities if possible. The main feature of the system is that it deals with underspecified descriptions directly, i.e. without enumerating readings.

1 Introduction

A particularly appealing aspect of underspecification (van Deemter and Peters, 1996; Reyle, 1993; Muskens, 1995; Pinkal, 1996; Bos, 1996) is that it can in principle deal very efficiently with *local ambiguities* – ambiguities which are only due to lack of information at an intermediate stage of processing and go away by the end of the analysis. An example for this effect is (1): The scope ambiguity that is perceived after processing the first sentence is no longer present after the second one. This effect can be explained in a framework of dynamic semantics (Groenendijk and Stokhof, 1991; Kamp and Reyle, 1993) by the fact that a wide-scope universal quantifier would make the indefinite inaccessible for anaphoric reference from the second sentence.

(1) Every man loves a woman.
Her name is Mary.

In this paper, we show how this particular type of local ambiguity can be processed efficiently. The approach we propose employs deterministic inference rules that can exclude the readings which violate anaphoric accessibility conditions without enumerating them. These rules operate directly on underspecified descriptions and fully maintain underspecifiedness. We also show how this behaviour can be captured by constraint propagation in an existing implementation of tree descriptions using finite set constraints (Duchier and Niehren, 2000; Koller and Niehren, 2000; Duchier and Gardent, 1999).

More specifically, we introduce *DPL structures*, extended tree structures that encode formulas of dynamic predicate logic (DPL) in much the same way as Egg et al.'s (1998) lambda structures encode λ-terms. Then we define a constraint language for the description of DPL structures, called CL(DPL), in analogy to Egg et al.'s constraint langague for lambda structures (CLLS). We characterize those DPL structures in which all restrictions on anaphoric accessibility are obeyed by talking directly about the syntactic structure of a DPL formula. This is in contrast to the standard procedure in dynamic semantics, where the dynamic behaviour is produced by the semantics of the logic; we do not need to (and do not) talk about interpretation of DPL structures and model accessibility by purely "static" means.

The paper is structured as follows. In Section 2, we introduce DPL structures and tree descriptions in the language CL(DPL). In Section 3, we add syntactic restrictions on admissible variable bindings to DPL structures and present axioms that characterize these restrictions. In Section 4, we turn these axioms into deterministic inference rules and combine them with deterministic inference rules known from an existing inference algorithm for dominance constraints. We obtain a procedure that can do the kind of underspecified reasoning described above without enumerating readings. In Section 5, we sketch an implementation of our inference system based on finite set constraint programming. This implementation can be obtained by adapting an existing implementation of a solver for dominance constraints. Finally, we conclude and point to further work in Section 6.

2 Tree Descriptions

In this section, we define the Constraint Language for DPL structures, CL(DPL), a language of tree descriptions which conservatively extends dominance constraints (Marcus et al., 1983; Rambow et al., 1995; Koller et al., 2000) by variable binding constraints. CL(DPL) is a close relative of the Constraint Language for

Lambda Structures (CLLS), presented in (Egg et al., 1998). It is interpreted over *DPL structures* – trees extended by a variable binding function which can be used to encode formulas of dynamic (or static) predicate logic. We will define DPL structures in two steps and then the language to talk about them.

2.1 Tree Structures

For the definitions below, we assume a *signature* $\Sigma = \{@_{|2}, \mathsf{var}_{|0}, \underline{\forall}_{|1}, \underline{\exists}_{|1}, \underline{\wedge}_{|2}, \mathsf{man}_{|1}, \mathsf{like}_{|2}, \ldots\}$ of *node labels*, each of which is equipped with a fixed *arity* $n \geq 0$. The labels $\underline{\wedge}, \underline{\neg}, \underline{\forall}, \ldots$ are the first-order connectives. 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' (written $\pi \leq \pi'$) 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.

Definition 2.1 *A* tree structure *is a tuple* (Δ, σ) *consisting of a finite tree domain* Δ *and a total* labeling function $\sigma : \Delta \to \Sigma$ *such that for all* $\pi \in \Delta$ *and* $i \in \mathbb{N}$*:*

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

We say that the nodes $\pi, \pi_1, \ldots, \pi_n$ are in the *labeling relationship* $\pi{:}f(\pi_1, \ldots, \pi_n)$ iff $\sigma(\pi) = f$ and for each $1 \leq i \leq n$, $\pi_i = \pi i$. Similarly, we say that a node π *properly dominates* a node π' and write $\pi \lhd^+ \pi'$ iff π is a proper prefix of π'. We take π and π' to be *disjoint* ($\pi \perp \pi'$) if they are different and neither node dominates the other. So any two nodes in a tree structure are in one of the four relations $=$ (equality), $\lhd^+$, $\rhd^+$ (the inverse of $\lhd^+$), or $\perp$. We shall also be interested the combinations of these relations by set operators: intersection, complementation, union, and inversion. For instance, the *dominance relation* $\lhd^*$ is defined as the union of node equality and proper dominance $= \cup \lhd^+$. Finally, we define the ternary *non-intervention relation* $\neg(\pi \lhd^* \pi' \lhd^* \pi'')$ to hold iff it is not the case that both $\pi \leq \pi'$ and $\pi' \leq \pi''$.

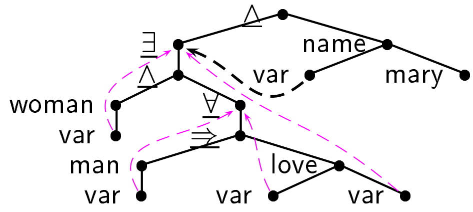

Figure 1: DPL structure for the meaning of (1).

2.2 DPL structures

Now we extend tree structures by variable binding and obtain *DPL structures*. To this end, we partition Σ into three sets: *connectives* $\Sigma_{\mathrm{con}} = \{\underline{\forall}, \underline{\wedge}, \underline{\neg}, \ldots\}$, *predicate symbols* $\Sigma_{\mathrm{pred}} = \{\mathsf{man}, \mathsf{likes}, \ldots\}$, and *term symbols* $\Sigma_{\mathrm{term}} = \{\mathsf{var}, \mathsf{peter}, \mathsf{mother\_of}, \ldots\}$ which subsume the variable symbol var and function symbols.

Definition 2.2 *A* DPL structure *is a triple* $(\Delta, \sigma, \lambda)$ *consisting of a tree structure* (Δ, σ) *and a partial* variable binding function $\lambda : \Delta \rightsquigarrow \Delta$ *which satisfies for all* $\pi, \pi' \in \Delta$*:*

1. $\sigma(\pi) \in \Sigma_{\mathrm{con}}$ *then* $\sigma(\pi i) \in \Sigma_{\mathrm{con}} \cup \Sigma_{\mathrm{pred}}$ *for all* $\pi i \in \Delta$*;*
2. $\sigma(\pi) \in \Sigma_{\mathrm{pred}} \cup \Sigma_{\mathrm{term}}$ *then* $\sigma(\pi i) \in \Sigma_{\mathrm{term}}$ *for all* $\pi i \in \Delta$*;*
3. $\lambda(\pi) = \pi'$ *then* $\sigma(\pi) = \mathsf{var}$ *and* $\sigma(\pi') \in \{\underline{\forall}, \underline{\exists}\}$.

DPL structures can be used to represent formulas of first-order predicate logic. For instance, the DPL structure in Fig. 1 represents the (unique) meaning of (1). So far, however, variables bound by a quantifier do not need to be in any special position in a DPL structure; in particular, not in its scope. To enforce scoping as in static predicate logic, we could simpy add the condition $\pi' \lhd^* \pi$ in condition 3 of Definition 2.2. We will define an appropriate counterpart for DPL in Section 3 (properness).

Modeling variable binding with an explicit binding function instead of variable names was first proposed in (Egg et al., 1998). There, binding functions help to avoid a capturing problem in the context of scope underspecification which becomes most apparent in the presence of ellipsis. Here the binding function mainly gives us a different perspective on variable binding which

$$
\begin{array}{rcll}
R & ::= & \lhd^{+} \mid \rhd^{+} \mid \; = \; \mid \; \perp & \\
 & \mid & R \cup R \mid R \cap R \mid \neg R \mid R^{-1} & \\
\varphi & ::= & X{:}f(X_1, \ldots, X_n) & (f_{|n} \in \Sigma) \\
 & \mid & XRY & \\
 & \mid & \neg(X \lhd^* Y \lhd^* Z) & \\
 & \mid & \lambda(X){=}Y & \\
 & \mid & \varphi \wedge \varphi'. &
\end{array}
$$

Figure 2: Syntax of CL(DPL)

is useful for defining properness of DPL structures.

2.3 The Constraint Language CL(DPL)

The syntax of CL(DPL) is defined in Figure 2. It provides constraints for all the relations discussed above. There are *labeling constraints* $X{:}f(X_1, \ldots, X_n)$, expressive combinations XRY of *dominance constraints with set operators* (Duchier and Niehren, 2000; Cornell, 1994), *non-intervention constraints* $\neg(X \lhd^* Y \lhd^* Z)$, and *binding constraints* $\lambda(X){=}Y$.

CL(DPL) is interpreted over DPL structures. A *variable assignment* into a DPL structure $\mathcal{M}$ is a total function from the set of variables of a constraint to the domain of $\mathcal{M}$. A pair $(\mathcal{M}, \alpha)$ of a DPL structure $\mathcal{M}$ and a variable assignment α into $\mathcal{M}$ *satisfies* a constraint φ iff it satisfies all of its atomic constraints; that is, if the relation with the same symbol holds of the nodes assigned to their arguments. We also call the pair $(\mathcal{M}, \alpha)$ a *solution* and $\mathcal{M}$ a *model* of φ.

Only some of the atomic constraints in CL(DPL) are used in underspecified descriptions – in particular, labeling, dominance, and binding constraints; the other constraints are helpful in processing the others. These three types of constraints can be transparently displayed in *constraint graphs*. For instance, the constraint graph in Fig. 3 represents a constraint describing the readings of example (1) including the scope ambiguity. The nodes of the graph stand for variables in the constraint; labels and solid edges represent labeling constraints, dotted edges, dominance constraints, and dashed arrows, binding constraints. In addition, the constraint graph represents an inequality constraint $X \neg{=} Y$ between each two variables whose nodes carry a label. A constraint with the latter property is called *overlap-free*. The intuition is that the solid-edge tree fragments in the constraint graph must never overlap properly in a solution.

3 Dynamic Semantics in CL(DPL)

The semantics of DPL is built in a way that allows quantifiers to bind only variables in certain positions: inside their scopes and, if it is an existential quantifier, from the left-hand sides of conjunctions and implications into the right-hand sides. In CL(DPL), we model this as a purely syntactic restriction on the accessibility of binders which we define as a structural property of DPL structures. DPL structures which have this property will be called *proper*.

A useful auxiliary concept for the definition is that of an *infimum* of two nodes with respect to the dominance relation $\lhd^*$, which constitutes a lower semilattice because of the underlying treeness of DPL structures. Furthermore, we will use the standard DPL notions of *internally dynamic* connectives $\Sigma_{\mathrm{con}}^{\mathrm{dyn}} = \{\underline{\wedge}, \underline{\Rightarrow}\}$ and *externally static* connectives $\Sigma_{\mathrm{con}}^{\mathrm{stat}} = \{\underline{\neg}, \underline{\forall}, \underline{\Rightarrow}, \underline{\vee}\}$. The semantics definition of DPL gives these two groups special relevance for variable binding.

Now we can define *proper* DPL structures as follows.

Definition 3.1 *A DPL structure $\mathcal{M}$ is called* proper *if for each node π of $\mathcal{M}$ on which λ is defined, one of the following cases holds true where μ is the infimum of π and $\lambda(\pi)$.*

1. $\mu = \lambda(\pi)$, *or*
2. $\lambda(\pi)$ *is labeled with* $\underline{\exists}$, $\mu 1 \lhd^* \lambda(\pi)$, $\mu 2 \lhd^* \pi$, μ *is labeled with an internally dynamic connective, and no node between $\mu 1$ and $\lambda(\pi)$, inclusively, is labeled with an externally static connective.*

Intuitively, the first branch of the definition corresponds to usual binding of variables inside the scope of a quantifier. In the second branch, the positions of the variable and the (existential) quantifier in the DPL structure are disjoint, and the quantifier is dominated by the left child of the infimum. Then the infimum must be labeled with an internally dynamic connective, and there must be no externally static connective between this node and the quantifier. This restriction is what we are going to exploit

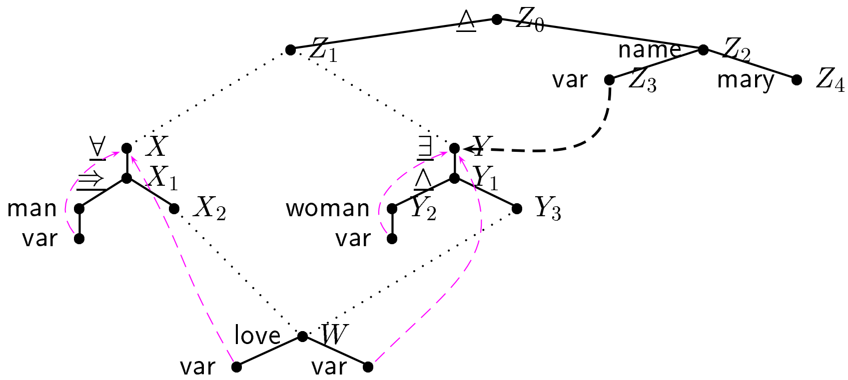

Figure 3: Constraint graph for (1).

to capture the influence on scope. There is no such restriction for the path between the infimum and the variable.

Solutions of a constraint that violate the dynamic accessibility conditions are now excluded simply by restricting the class of admissible solutions to proper ones. As expected from the linguistic intuition, only one solution of the running example (1) is proper: the one where "a woman" is assigned wide scope (Fig. 1). The other solution is not proper because the path from the infimum (denoted by Z_0 in Fig. 3) to the antecedent contains a universal quantifier.

Properness of a DPL structure can be axiomatized syntactically: A DPL structure is proper iff the CL(DPL) axioms (Dyn1) to (Dyn4) in Fig. 4 are valid over it. The rule (Dyn1) forces universal quantifiers to bind only variables in their scopes, and the rules (Dyn2) to (Dyn4) enforce properness of binding when a variable is not in the scope of its binder.

4 Underspecified Reasoning

We next present a procedure for underspecified reasoning with dynamic semantics. The goal is to narrow an underspecified description such that improper DPL-structures are removed from the solution set. Narrowing should apply as soon as possible, so underspecifiedness can be maintained and readings need not be enumerated. We present an inference procedure that can do this and go through two examples.

4.1 Inference Procedure

This inference procedure *saturates* a constraint according to the rules in Figures 4 and 5; that is, whenever a constraint contains the left-hand side of a rule, it adds its right-hand side, until no new conjuncts can be added. Fig. 4 contains simply the properness axioms from the previous sections, turned into deterministic proof rules. The rules in Fig. 5 are propagation rules from Algorithm *DO* in (Duchier and Niehren, 2000), plus new rules for non-intervention constraints. Algorithm *DO* contains some additional rules, in particular *distribution* rules that perform case distinctions, because *DO* is a complete solver for dominance constraints with set operators, which improves on (Duchier and Gardent, 1999; Koller et al., 1998). We have omitted the distribution rules here because we do *not* want to perform case distinctions; by adding them again, we could enumerate all proper solutions, as Schiehlen (1997) does for UDRT.

The new rules (NonI1) and (NonI2) allow to derive dominance information from non-intervention constraints. As we will see, the most interesting rule in Fig. 4 is (Dyn2), which derives explicit non-intervention information from the structural properties of dynamic binding. Note that while the rules in Fig. 5 are sound over any DPL structure, those in Fig. 4 are only sound over proper DPL structures. This is intended: Application of a properness rule is *supposed* to exclude (improper) solutions.

4.2 Examples

The inference rules go a long way towards making the effect of dynamic semantics on scope explicit. Let us consider the running example in Figure 3 to see how this works; we show how to derive $Y_3 \triangleleft^* X$, which specifies the relative quantifier scope.

First of all, we need to make the information

$$
\begin{array}{ll}
\text{(Dyn1)} & \lambda(X){=}Y \wedge Y{:}\underline{\forall}(Y') \quad\rightarrow\quad Y \lhd^* X \\
\text{(Dyn2)} & \lambda(X){=}Y \wedge Z{:}f(Z_1,Z_2) \wedge Z_1 \lhd^* Y \wedge Z_2 \lhd^* X \wedge W{:}g(W_1,\ldots,W_n) \quad\rightarrow\quad \neg(Z_1 \lhd^* W \lhd^* Y) \\
 & \qquad\qquad (f \in \Sigma_{\text{con}}^{\text{dyn}}, g_{|n} \in \Sigma_{\text{con}}^{\text{stat}}) \\
\text{(Dyn3)} & \lambda(X){=}Y \wedge Z{:}f(Z_1,\ldots,Z_n) \wedge Z_i \lhd^* X \wedge Z_j \lhd^* Y \quad\rightarrow\quad \mathsf{false} \quad (f_{|n} \in \Sigma_{\text{con}} - \Sigma_{\text{con}}^{\text{dyn}}, i \neq j) \\
\text{(Dyn4)} & \lambda(X){=}Y \wedge Z{:}f(Z_1,\ldots,Z_n) \wedge Z_i \lhd^* X \wedge Z_j \lhd^* Y \quad\rightarrow\quad \mathsf{false} \quad (f_{|n} \in \Sigma, i < j)
\end{array}
$$

Figure 4: Properness axioms.

$$
\begin{array}{ll}
\text{(Trans)} & X \lhd^* Y \wedge Y \lhd^* Z \quad\rightarrow\quad X \lhd^* Z \\
\text{(Lab.Dom)} & X{:}f(\ldots,Y,\ldots) \quad\rightarrow\quad X \lhd^+ Y \\
\text{(NegDisj)} & X \lhd^* Z \wedge Y \lhd^* Z \quad\rightarrow\quad X \neg\perp Y \\
\text{(Lab.Disj)} & X{:}f(\ldots,X_i,\ldots,X_j,\ldots) \quad\rightarrow\quad X_i \perp X_j \qquad \text{where } i < j \\
\text{(Inter)} & X R_1 Y \wedge X R_2 Y \quad\rightarrow\quad X R Y \qquad \text{if } R_1 \cap R_2 \subseteq R \\
\text{(Inv)} & X R Y \quad\rightarrow\quad Y R^{-1} X \\
\text{(Child.down)} & X \lhd^+ Y \wedge X{:}f(X_1,\ldots,X_n) \wedge \bigwedge_{i=1,i\neq j}^{n} X_i \neg\lhd^* Y \quad\rightarrow\quad X_i \lhd^* Y \\
\text{(NegDom)} & X \neg\perp Y \wedge X \perp Z \quad\rightarrow\quad Z \neg\lhd^* Y \\
\text{(NonI1)} & \neg(X \lhd^* Y \lhd^* Z) \wedge X \lhd^* Y \quad\rightarrow\quad Y \neg\lhd^* Z \\
\text{(NonI2)} & \neg(X \lhd^* Y \lhd^* Z) \wedge Y \lhd^* Z \quad\rightarrow\quad X \neg\lhd^* Y
\end{array}
$$

Figure 5: Propagation rules for dominance and non-intervention constraints.

$Z_2 \lhd^* Z_3$ explicit by application of (Lab.Dom) and (Inter). In this instance, (Inter) is used as a rule of weakening.

$$
\begin{array}{ll}
\text{(Lab.Dom)} & Z_2{:}\underline{\triangle}(Z_3,Z_4) \rightarrow Z_2 \lhd^+ Z_3 \\
\text{(Inter)} & Z_2 \lhd^+ Z_3 \rightarrow Z_2 \lhd^* Z_3
\end{array}
$$

Now we can apply the rule (Dyn2) to the variable binding constraint $\lambda(Z_3) = Y$ (drawn in boldface in the graph) and the $\underline{\forall}$ labeling constraint to derive a non-intervention constraint.

$$
\begin{array}{ll}
\text{(Dyn2)} & Z_0{:}\underline{\triangle}(Z_1,Z_2) \wedge Z_1 \lhd^* X_1 \wedge X{:}\underline{\forall}(X_1) \\
 & \wedge\ Z_2 \lhd^* Z_3 \wedge \lambda(Z_3) = Y \\
 & \rightarrow\quad \neg(Z_0 \lhd^* X \lhd^* Y)
\end{array}
$$

All that is left to do is to make the positive dominance information contained in the new non-intervention constraint explicit. As the constraint also contains $Z_0 \lhd^* X$, we can apply (NonI1) on the new non-intervention constraint and derive $X \neg\lhd^* Y$.

$$
\text{(NonI1)} \quad \neg(Z_0 \lhd^* X \lhd^* Y) \wedge Z_0 \lhd^* X \rightarrow X \neg\lhd^* Y
$$

On the other hand, we can derive non-disjointness of X and Y because (Trans), (Lab.Dom), and (Inter) allow the derivation of $X \lhd^* W$ and $Y \lhd^* W$:

$$
\text{(NegDisj)} \quad X \lhd^* W \wedge Y \lhd^* W \rightarrow X \neg\perp Y
$$

We can now combine all of our constraints for X and Y with the intersection rule and obtain $Y \lhd^* X$, which basically determines the order of the two quantifiers:

$$
\text{(Inter)} \quad X \neg\lhd^* Y \wedge X \neg\perp Y \rightarrow Y \lhd^* X
$$

By exploiting the fact that the constraint is overlap-free (i.e. contains an inequality constraint for each two labeled variables), we can even derive $Y_3 \lhd^* X$ by repeated application of the rules (Child.down), (Lab.Disj), (NegDisj), and (NegDom). This means that we have fully disambiguated the scope ambiguity by saturation with deterministic inference rules.

Now let us consider a more complicated example. Fig. 6 is the underspecified description of the semantics of

(2) Every visitor of a company saw one of its departments.

The constraint graph has five solutions, three of which are proper. Unfortunately, the constraint language is not expressive enough to describe these three solutions in a single constraint: Both X and Z can be either above or below Y, even in a proper solution, but if X is below Y, Z must be too, and if X is above Y, Z must be anywhere below X (but may be above

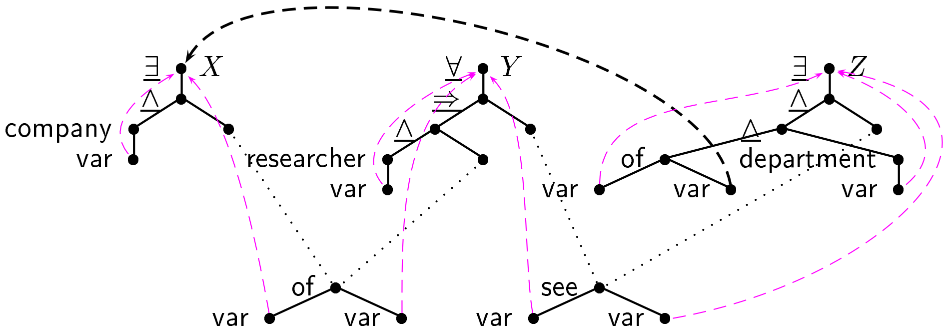

Figure 6: Constraint graph for (2).

Y!). In other words, this constraint is an example where the inference procedure is not strong enough to narrow the description. In this case, we must still resort to performing nondeterministic case distinctions; at worst, the rules will apply on solved forms of CL(DPL) constraints.

5 Processing with Finite Set Constraints

This inference procedure fits nicely with an implementation of dominance constraints based on constraint programming (Marriott and Stuckey, 1998; Koller and Niehren, 2000) with finite set constraints (Müller, 1999). Constraint programming is a technology for solving combinatoric puzzles efficiently. The main idea is to replace "generate and test" by "propagate and distribute". Constraint propagation performs deterministic inferences which prune the search space, whereas distribution performs (nondeterministic) case distinctions.

Duchier and Niehren (2000) show how to implement a dominance constraint solver by encoding dominance constraints as finite set constraints and disjunctive propagators. This solver does not handle non-intervention constraints, but we show here that they can be added very naturally. The (Dyn) rules still have to be implemented as saturation rules.

The idea of this implementation is to encode a solution $(\mathcal{M}, \alpha)$ of a dominance constraints by introducing for each variable X in the constraint and each relation symbol $R \in \{\lhd^+, \rhd^+, =, \perp\}$ a finite set variable $R(X)$. This variable is supposed to denote the set of all variables denoting nodes that are in the relation R to $\alpha(X)$:

$$R(X) = \{Y \in V(\varphi) \mid (\mathcal{M}, \alpha) \models Y R X\}$$

Dominance constraints can now be stated as constraints over these set variables; examples for set constraints are $V \subseteq V'$ and $V = V_1 \cup V_2$. The new non-intervention constraint $\neg(X \lhd^* Y \lhd^* Z)$ can be encoded as

$$Y \in \lhd^+(X) \cup \perp(Z) \cup \rhd^+(Z).$$

The builtin propagation for set constraints automatically implements the rules (NonI1) and (NonI2). For instance, assume that $X \lhd^* Y$ belongs to φ; then there will be a set constraint $Y \notin \lhd^+(X)$, so set constraint propagation will derive $Y \in \perp(Z) \cup \rhd^+(Z)$. This is the immediate encoding of $Y \perp \cup \rhd^+ Z$, which is equivalent to $Y \neg\lhd^* Z$.

6 Conclusion

In this paper, we have shown how a specific type of local ambiguity, which is produced by the interaction of intersentential anaphora and scope ambiguities, can be processed efficiently in the framework of underspecification. We have defined DPL structures, which can be used to model formulas of DPL, and *proper* DPL structures, in which variable binding must obey the accessibility conditions of DPL. Finally, we have shown how an underspecified description can be narrowed to a description of its proper solutions, sometimes without even partial enumeration of readings, and integrated this operation into an implementation of dominance constraints which is based on finite set constraints.

Seen from the perspective of DPL, our definition of properness is purely syntactic and technically has nothing to do with dynamic semantics. We could state such a definition because the explicit variable binding functions gave us a structure-independent handle on variable binding that excluded all forms of capturing. This deviates from the standard perspective of indefinites changing the context, but has the advantage of being extremely modular in that the accessibility conditions are factorized out explicitly. For instance, it is simple to represent the meaning of “Bach-Peters sentences” by relaxing these conditions; it should also be easy to adapt our formalism to other frameworks of dynamic semantics. Of course, the question of how to interpret a DPL structure remains open.

Another open question is how the approach presented here can be extended to higher-order systems of dynamic semantics (e.g. Dynamic Lambda Calculus (Kuschert, 1999)). In this context, it could be worthwhile to restore the distinction of variable binding and anaphoric linking from CLLS.

Finally, it should be interesting to find other classes of local ambiguity that lend themselves to a treatment as presented here. So far, there are not many related examples; one is lexical ambiguity in parsing of dependency grammar, as presented in (Duchier, 1999). However, we believe that the work presented here provides further illustration that underspecified processing can go a long way towards efficient processing of local ambiguities.

Acknowledgments. This work was supported by the Deutsche Forschungsgemeinschaft in the SFB 378. As always, we thank all members of the SFB 378 project CHORUS at the University of the Saarland. We are also grateful to the participants at the Dagstuhl workshop on Dynamic Semantics in February 1999 for comments and discussions on an earlier version of this paper.

References

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

Thomas Cornell. 1994. On determining the consistency of partial descriptions of trees. In *Proceedings of ACL*.

Denys Duchier and Claire Gardent. 1999. A constraint-based treatment of descriptions. In *3rd Int. Workshop on Comp. Semantics*, pages 71–85.

Denys Duchier and Joachim Niehren. 2000. Dominance constraints with set operators. In *1st Int. Conf. on Computational Logic*, LNCS, July.

Denys Duchier. 1999. Axiomatizing dependency parsing using set constraints. In *Proc. of the 6th M. on Mathematics of Language*, pages 115–126.

Markus Egg, Joachim Niehren, Peter Ruhrberg, and Feiyu Xu. 1998. Constraints over lambda-structures in semantic underspecification. In *joint 17th Int. Conf. on Comp. Ling. and 36th Ann. Meet. of the ACL.*, pages 353–359.

Jeroen Groenendijk and Martin Stokhof. 1991. Dynamic predicate logic. *Linguistics & Philosophy*, 14:39–100.

Hans Kamp and Uwe Reyle. 1993. *From Discourse to Logic*. Kluwer, Dordrecht.

Alexander Koller and Joachim Niehren. 2000. Constraint programming in computational linguistics. In *Proc. of the 8th CSLI Workshop on Logic, Language, and Computation*. CSLI Press. To appear.

Alexander Koller, Joachim Niehren, and Ralf Treinen. 1998. Dominance constraints: Algorithms and complexity. In *3rd Conf. on Logical Asp. of Comp. Ling.* To appear as LNCS in 2000.

Alexander Koller, Kurt Mehlhorn, and Joachim Niehren. 2000. A polynomial-time fragment of dominance constraints. In *Proceedings of the 38th ACL*. To appear.

Susanna Kuschert. 1999. *Dynamic Meaning and Accomodation*. Ph.D. thesis, Dept. of Computer Science, University of the Saarland.

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

Kim Marriott and Peter J. Stuckey. 1998. *Programming with Constraints: An Introduction*. MIT Press.

Tobias Müller. 1999. Problem solving with finite set constraints in Oz. A Tutorial. Documentation of the Mozart system of Oz. `www.mozart-oz.org`.

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.

Manfred Pinkal. 1996. Radical underspecification. In *Proceedings of the 10th Amsterdam Colloquium*, pages 587–606.

Owen Rambow, K. Vijay-Shanker, and David Weir. 1995. D-Tree Grammars. In *Proceedings of ACL'95*.

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

Michael Schiehlen. 1997. Disambiguation of underspecified discourse repesentation structures under anaphoric constraints. In *2nd Int. Workshop. on Computational Semantics*, Tilburg.

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