Chapter 1

Evaluating Context Unification for Semantic Underspecification

Alexander Koller

Abstract. Context unification has been proposed as a powerful formalism for the uniform and underspecified treatment of scope ambiguities and parallelism. We develop an algorithm for context unification that alleviates the enormous computational and overgeneration problems of earlier algorithms. It is incomplete for full context unification, but finds all linguistically relevant solutions. We apply an implementation of the new algorithm to evaluate context unification as a tool for semantic analysis.

1 Introduction

Recently, scope ambiguities and parallelism, e.g. VP ellipses, have been investigated within the framework of underspecification [van Deemter & Peters, 1996]. The fundamental idea of this paradigm is to delay the enumeration of readings of an ambiguous sentence for as long as possible to avoid having to deal with a potentially exponential number of readings; operations are attempted to be executed on the underspecified representation that stands for the entire ambiguous meaning.

There are numerous accounts, underspecified and non-underspecified, that correctly treat one of these classes alone. An approach that covers both classes must consider their interaction, for example in "Hirschbühler sentences" [Hirschbühler, 1982] like that in example (1.1).

(1.1) Every researcher read some books. Every student did, too.

In this example, the first sentence contains a scope ambiguity of two quantifiers. The structure of the second sentence is parallel to that of the first; in particular, it contains a similar scope ambiguity. Although both sentences allow both scope relations, the parallelism also enforces that their scope relations must be the same. Hence, the example has exactly two readings, one for each scope relation in the source sentence.

Proceedings of the Third ESSLLI Student Session
Ivana Kruijff-Korbayová (editor).

Most formalisms that capture this interaction treat these phenomena in different ways. For example, [Shieber et al., 1996] resolve parallelism by higher-order unification and scope ambiguities by a Cooper storage-like mechanism. Only a few [Crouch, 1995, Niehren et al., 1997a] have managed to deal with them in terms of a single mechanism.

The approach in [Niehren et al., 1997a] allows a particularly transparent treatment of these phenomena. Similarly to [Crouch, 1995] and in contrast to [Shieber et al., 1996], it distinguishes clearly between the actual semantic representation of a sentence (the object-level term) and its (meta-level) description; only the description is derived directly from the syntactic analysis. Object-level terms, typically lambda terms, are treated as trees and are described as the minimal solutions of so-called *context constraints*, second-order formulae talking about trees. In the spirit of underspecification, the constraints are kept unresolved for as long as possible. Eventually, the solutions will be enumerated by a unification algorithm; this process is called *context unification*.

In this paper, we will evaluate the context unification approach as a formalism for underspecified semantic analysis. Such an evaluation must have two aspects. For one, it must consider if the formalism yields the linguistically correct results; in particular, we require that the set of minimal solutions of a context constraint is the same as the set of readings of the sentence it represents. On the other hand, it must consider computational adequacy. This means that the unification algorithm we use to enumerate solutions should find exactly the minimal solutions, and find them efficiently. Especially for this second aspect, an implementation of the considered algorithm is necessary. In turn, this makes the examination of the first aspect possible for larger examples.

The investigation of the computational aspects of context unification is an active field of research in theoretical computer science that has spawned several sound and complete algorithms for context unification and various related problems [Lévy, 1996, Schmidt-Schauß & Schulz, 1998, Niehren et al., 1997b]. Unfortunately, each of these algorithms has drawbacks that prevent its use for the linguistic application, and we have to come up with an algorithm of our own that is better suited for this domain before we can evaluate context unification fairly.

In the next section, we present the formal foundations for context unification and how it can be applied in semantics. In Section 3, we describe how a unification algorithm that allows for a reasonably efficient implementation and is linguistically adequate can be obtained. In Section 4, we apply the implementation to a set of examples and evaluate the results. Section 5 summarizes and concludes the paper.

2 Context unification and its application to linguistics

The language of context unification contains first-order variables X, a given set of tree constructors f, and *context variables* C. A *context constraint* is a conjunction of equations between terms t with the following formal syntax:

$$t ::= X \mid f(t_1, \ldots, t_n) \mid C(t).$$

Semantically, terms (and, in particular, first-order variables) are interpreted as ground, i.e. variable-free, terms. We exploit the correspondence between ground terms and finite trees; for example, the ground term $f(a, b)$ is equivalent to a tree whose root is labeled with the binary constructor f and that has two subtrees, each of which consists of a single node labeled with the nullary constructors a and b, respectively (Figure 1.1 (a)).

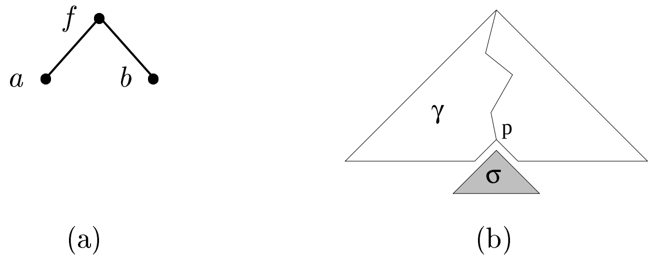

Figure 1.1: (a) The tree corresponding to the ground term $f(a, b)$. (b) A context.

Context variables C denote *context functions*, functions from trees to trees that insert their arguments into fixed *contexts*, trees with exactly one hole (Figure 1.1 (b)). *Context unification* is the problem of solving context constraints. A solution of a context constraint is a mapping of the variables to trees and context functions, respectively. For example, the context constraint

$$f(a, b) = C(a)$$

has exactly one solution: C must be mapped to the context function $\lambda X.f(X, b)$. An important, but not easily definable notion is that of a *minimal* solution. Many context constraints have an infinite number of solutions because a variable can be instantiated with an arbitrary term or context; in a minimal solution, such an "open" variable must be filled with as little additional material as possible.

What makes context unification interesting for natural-language semantics is that we can use it for underspecified descriptions of lambda terms.

More precisely, we represent lambda terms as first-order ground terms with special constructors and impose context constraints to describe them. Application $t_1(t_2)$ is written as $t_1@t_2$; @ is a new binary tree constructor that is pronounced "apply". Lambda abstraction $\lambda x.t$ is represented by the tree $\mathrm{lam}_x(t)$; occurrences of the abstracted variable x are replaced by the nullary constructor var_x. Note the distinction between object-level abstraction $\mathrm{lam}_x(\ldots \mathrm{var}_x \ldots)$ and meta-level abstraction $\lambda X. \ldots X \ldots$ that is used to represent context functions.

By way of example, consider Example (1.1) from above. The semantics X_s and X_t of the source and target sentence of the ellipsis are described by the context constraint (1.3).

(1.2) Every researcher read some books. Every student did, too.

(1.3) $X_s = C_1(\mathit{every}@\mathit{res}@\,\mathrm{lam}_x(C_2(\mathit{read}@\,\mathrm{var}_y\,@\,\mathrm{var}_x)))$
$X_s = C_3(\mathit{some}@\mathit{book}@\,\mathrm{lam}_y(C_4(\mathit{read}@\,\mathrm{var}_y\,@\,\mathrm{var}_x)))$
$X_s = C(\mathit{every}@\mathit{res}) \qquad X_t = C(\mathit{every}@\mathit{student})$

(1.4) $X_s = \mathit{every}@\mathit{res}@\,\mathrm{lam}_x(\mathit{some}@\mathit{book}@\,\mathrm{lam}_y(\mathit{read}@\,\mathrm{var}_y\,@\,\mathrm{var}_x))$
$X_t = \mathit{every}@\mathit{student}@\,\mathrm{lam}_x(\mathit{some}@\mathit{book}@\,\mathrm{lam}_y(\mathit{read}@\,\mathrm{var}_y\,@\,\mathrm{var}_x))$

(1.5) $X_s = \mathit{some}@\mathit{book}@\,\mathrm{lam}_y(\mathit{every}@\mathit{res}@\,\mathrm{lam}_x(\mathit{read}@\,\mathrm{var}_y\,@\,\mathrm{var}_x))$
$X_t = \mathit{some}@\mathit{book}@\,\mathrm{lam}_y(\mathit{every}@\mathit{student}@\,\mathrm{lam}_x(\mathit{read}@\,\mathrm{var}_y\,@\,\mathrm{var}_x))$

The first two equations of (1.3) describe the scope ambiguity in the source sentence. They have the two minimal solutions shown in the first lines of the solutions (1.4) and (1.5). In one of these solutions, $\mathit{every}@\mathit{res}$ has scope over $\mathit{some}@\mathit{book}$; in the other, the relation is the other way round. These are just the two readings of the source sentence.

The lower two equations of (1.3) express that the terms X_s and X_t are almost the same, but X_t has $\mathit{every}@\mathit{student}$ in one location where X_s has $\mathit{every}@\mathit{res}$. In particular, these constraints enforce that no matter how the scope relations between the quantifiers in the source sentence are chosen, those in the target sentence will be the same. The values for X_t can be found in the second lines of the solutions (1.4) and (1.5).

Unfortunately, the constraint (1.3) has additional, linguistically unwanted solutions where X_s is bound to a term like

$$\begin{array}{l} f(\mathit{every}@\mathit{res}@\,\mathrm{lam}_x(\mathit{read}@\,\mathrm{var}_y\,@\,\mathrm{var}_x), \\ \quad \mathit{some}@\mathit{book}@\,\mathrm{lam}_y(\mathit{read}@\,\mathrm{var}_y\,@\,\mathrm{var}_x)), \end{array}$$

where f is an arbitrary binary constructor. The underlying problem is that the constraint expresses a *subtree* relation between the trees starting with the quantifiers and the tree $\mathit{read}@\,\mathrm{var}_y\,@\,\mathrm{var}_x$; what we really want to express is that the root node of this tree is *dominated* by the quantifiers. In [Niehren et al., 1997a], this is done by replacing the first two equations of the constraint by equations between contexts; the modified constraint looks like this:

(1.6) $C_s = \lambda X.C_1(every@res@\,\mathrm{lam}_x(C_2(X)))$
$C_s = \lambda X.C_3(some@book@\,\mathrm{lam}_y(C_4(X)))$
$X_s = C_s(read@\,\mathrm{var}_y\,@\,\mathrm{var}_x)$
$X_s = C(every@res) \qquad X_t = C(every@student)$

In [Niehren & Koller, 1998], it is shown that this can be extended to a systematic way of encoding dominance in context unification. In the same paper, it is shown that every equation between contexts can be expressed as a conjunction of two equations between tree-valued expressions; so allowing them does not affect the expressiveness of the language.

3 An algorithm for context unification

In general, the decidability of context unification is an open problem. Its complexity is between that of string unification [Makanin, 1977], which is decidable but extremely complex, and that of second-order unification, which is undecidable. There are several known decidable fragments; unfortunately, neither of them is sufficient for our application. For example, there is a decision procedure for *stratified* context constraints – context constraints where every occurrence of a variable must have the same second-order prefix, i.e. the context variables visited to reach the variable must be the same [Schmidt-Schauß & Schulz, 1998, Höhl, 1997]. This fragment is too small for the linguistic application; consider the following example:

(1.7) *Every student read a book. Every professor who hadn't read every paper did too.*

(1.8) $C_s = \lambda X.C_1(every@student@\,\mathrm{lam}_x(C_2(X)))$
$C_s = \lambda X.C_3(a@book@\,\mathrm{lam}_y(C_4(X)))$
$X_s = C_s(read@\,\mathrm{var}_x\,@\,\mathrm{var}_y)$
$X_s = C(every@student)$
$X_t = C(every@\,\mathrm{lam}_z(professor@\,\mathrm{var}_z \wedge C'(read@\,\mathrm{var}_z\,@\,\mathrm{var}_w)))$
$C' = \lambda X.C_5(every@paper@\,\mathrm{lam}_w(C_6(X)))$
$C' = \lambda X.C_7(\neg C_8(X))$

The constraint (1.8) is not stratified because the variable C' has the second-order prefix C in the fifth equation and the second-order prefix ϵ in the sixth and seventh one.

Another decidable fragment that has been found by Lévy within the framework of linear second-order unification [Lévy, 1996] imposes the restriction that every variable must occur at most twice in the constraint, which is clearly violated in Example (1.8). Lévy's (not necessarily terminating) algorithm for full linear second-order unification uses second-order variables that can take more than one argument; in the setting of context unification, this would correspond to contexts with multiple holes, which is not what we want.

The algorithm we have based our work on [Niehren et al., 1997b, Appendix B] doesn't necessarily terminate, either; but it is very simple. This algorithm, shown in Figure 1.2, is a nondeterministic system of rewrite rules $A \to B \mid C$, replacing the atomic constraint A by B and binding variables as described in C (where Id means no variables are bound in this step). It terminates when the constraint has been reduced to `true`. The rules are, for the most part, adaptations of the standard rules for higher-order unification algorithms. In Flex/Rigid situations, a choice can be made between applications of the Projection and Imitation rules; the latter demands that the argument of the context variable must be unified with one of the arguments of the rigid term, while the former simply removes the context variable and demands that its argument must be unified with the other side. In Flex/Flex situations, a choice can be made between Projection and two rules that are specific for such a situation; one of them states that C is a subcontext of C', and the other one says that both contexts start with the same (newly introduced) constructor and contain the arguments in compatible positions.

A deterministic implementation has to make these choices explicitly. The different sequences of choices span a search tree whose exploration can be easily coded in the concurrent constraint programming language Oz [Smolka, 1995, Oz, 1997] due to its powerful searching primitives.

Apart from not necessarily terminating, the original algorithm has two essential drawbacks. Even for a unification problem as simple as (1.3), the search space has tens of thousands of nodes and takes very long to explore. Even worse, the algorithm enumerates thousands of solutions, the vast majority of which are linguistically unwanted because they are non-minimal.

To make the algorithm more tractable and take care of the minimality problem, we change it in two essential directions.

1. To reduce the size of the search space, we remove the rules that introduce the most nondeterminism from the algorithm.

2. To exclude the non-minimal solutions, we add simple types to the object language and require all produced terms to be well-typed.

Especially the first change that we made to the algorithm deserves some discussion. The most problematic rules are the two Flex-Flex rules. The removal of the first Flex-Flex rule has no effect on the completeness of the algorithm: Every minimal solution of a constraint that can be derived by using the Flex-Flex 1 rule can be derived by using a series of applications of the Imitation and Projection rules instead.

The removal of the second Flex-Flex rule, however, renders the algorithm incomplete for full context unification; for example, the modified algorithm

(Subst)	$X{=}t \longrightarrow \mathsf{true}$ if X does not appear free in t $\mid X \mapsto t$
(Decomp)	$a(t_1,...,t_n){=}a(t'_1,...,t'_n) \longrightarrow \bigwedge_{i=1..n} t_i{=}t'_i \mid Id$
(Proj)	$t{=}C(t') \longrightarrow t{=}t' \mid C \mapsto \lambda X.X$
(Imit)	$a(t_1,...,t_n){=}C(t') \longrightarrow t_i{=}C'(t') \mid C \mapsto \lambda X.a(t_1,..,t_{i-1},C'(X),t_{i+1},..,t_n)$

(Flex-Flex1)	$C(t){=}C'(t') \longrightarrow t{=}C''(t')$	$C' \mapsto \lambda X.C(C''(X))$
(Flex-Flex2)	$C(t){=}C'(t') \longrightarrow \mathsf{true}$	$C \mapsto \lambda Y.C_1(a(\,\pi(\overline{X},C_2(Y),C_3(t'))\,))$, $C' \mapsto \lambda Z.C_1(a(\,\pi(\overline{X},C_2(t),C_3(Z))\,))$ where π is a permutation

Figure 1.2: The algorithms for context unification. The rules below the line are removed to make the complete algorithm more tractable.

will claim that the following constraint has no solutions (when, in fact, it is solved by $X = f(a,b)$):

(1.9) $X = C_1(a)$
$X = C_2(b)$

Fortunately, this does not seem to matter for our application. The context variables that Flex-Flex 2 could be applied on are there to model a scope ambiguity; they are part of scope-bearing elements that should dominate each other in a solution. Hence, the solutions produced by Flex-Flex 2 (in which the scope-bearing elements are placed next to each other in the tree) are linguistically unwanted. Another hint is that in the linguistic analysis, all the material that appears in a solution is known in the beginning and mentioned in the constraints. Flex-Flex 2 always introduces a new constructor, so every solution that is produced using this rule will contain spurious material. We have no formal proof that the incompleteness doesn't affect the linguistic domain, but as we will show below, next to these more intuitive reasons, we have tested the modified algorithm on a large set of examples and never encountered a problem.

The other change is more straightforward; typing lambda terms has a long tradition in linguistics. The typing system we have to use here is slightly more sophisticated to account for the representation of lambda terms via special constructors: we have two varieties of types, τ-types that describe

(1) *Every man loves a woman.*
(2) *Peter likes Mary. John does too.*

Example	complete algorithm			modified algorithm		
	space	solutions	time	space	solutions	time
(1)	3000	600	40 sec	72	2	1 sec
(2)	13000+	2200+	2+ hrs	112	2	1 sec

Figure 1.3: Performance with the original (complete) and the modified unification algorithm.

the types of terms and κ-types that describe, for each constructor f, how the τ-type of the term $f(t_1, \ldots, t_n)$ is related to those of its subterms $t_1, \ldots, t_n$. These types can be of the following forms:

$$\begin{array}{lcl} \tau & ::= & e \mid t \mid \langle \tau_1, \tau_2 \rangle \\ \kappa & ::= & (\tau_1, \ldots, \tau_n) \rightsquigarrow \tau. \end{array}$$

In our application, the only constructors that appear at internal nodes of the tree are the apply constructor @ and the lam_x constructors. Non-logical constructors such as *speak'* and *peter'* never have subtrees, so their κ-types always have empty argument lists. For example, the constructor *speak'* would be assigned the κ-type $\texttt{nil} \rightsquigarrow \langle e, t \rangle$; the τ-type of the term *speak'* would then simply be $\langle e, t \rangle$. The apply constructor @ can have different types in different positions of the same tree; it has the polymorphic κ-type

$$(\langle \tau_1, \tau_2 \rangle, \tau_1) \rightsquigarrow \tau_2,$$

where τ_1 and τ_2 are τ-type variables. The lam_x constructors abstract over a variable of type e, so their type generally is $t \rightsquigarrow \langle e, t \rangle$.

4 Results

These two changes make the algorithm computationally much more pleasant. The size of the search space is reduced drastically, and almost all unwanted solutions are cut away. This makes a comprehensive evaluation of the linguistic and computational aspects of context unification in semantic analysis possible. We have integrated our implementation with an HPSG front-end that produces context constraints from natural-language input [Debusmann et al., 1998] and run it on a large number of examples. It turns out that there is both good and bad news.

On one hand, context unification correctly captures scope ambiguities, ellipses, and their interaction. On all examples that we tested with the CHOLI system, the algorithm from the previous section enumerates exactly the minimal solutions, taking less than 15 seconds for most examples. Consider our running example (1.10).

(1.10) Every researcher read some books. Every student did, too.

If we feed its formalization, (1.3), to the implemented algorithm, the program explores a search space of 168 nodes and finds exactly the two solutions (1.4) and (1.5). This takes less than two seconds. The implementation also correctly handles more complex scope ambiguities such as the well-known benchmark problem (1.11) from [Hobbs & Shieber, 1987]:

(1.11) Every researcher of a company saw most samples.

This example has three quantifiers, yielding six theoretically possible scope relations. However, only five of these are linguistically attested. Our implementation finds exactly these five readings. It explores a search space of 6.500 nodes, which takes around 15 seconds. For a comparison of the performances of the original (complete) and the modified algorithm, see Figure 1.3. The extra reading of the second example is due to the spurious scope ambiguity of $peter'$ and $mary'$ and disappears after beta reduction.

On the other hand, by making possible the examination of larger problems, our work shows that the complexity of context unification makes even restricted algorithms very sensitive towards the size of the problem. This is not surprising because the fragment of context unification for which the modified algorithm from Figure 1.2 is complete contains (the CU encoding of) string unification [Makanin, 1977], the best known algorithm for which is doubly exponential. In fact, there are unification problems on which it doesn't even terminate; but the constraints arising in linguistic analyses are always satisfiable, so this is not a problem in this domain.

The performance of our algorithm depends heavily on the exact formalization of a sentence. By adding two redundant equations to Example (1.3), for example, the run time is blown up dramatically and the algorithm finds 26 solutions; most of these have an additional (vacuous) quantifier. Generally, it can be observed that the run times and search spaces explode when the numbers of equations and variables in the solution grow. While it is easy to produce small representations of unification problems in a systematic way (which is, in fact, done by the HPSG grammar mentioned above), this certainly isn't a very pleasant property.

Some more fundamental problems remain. If, for example, the context variable C in the term $\mathrm{lam}_x(C(\mathrm{var}_x))$ is instantiated with a term that contains another lam_x, the variable is captured; in other words, it is unclear if lam_x should really be considered to bind var_x in the original constraint. This makes the correct treatment of alpha conversion extremely difficult and suggests the exploration of alternative approaches. A promising one, whose development has been stimulated by our results, is CLLS [Egg et al., 1998], which models lambda binding by explicit binding relations between tree nodes instead of via variable names. CLLS also talks directly about dominance relations between tree nodes, avoiding the somewhat awkward encoding of Example (1.6), and allows specific parallelism constraints that exactly restore the expressiveness of context unification [Niehren & Koller, 1998]. Moreover, mechanisms for modelling anaphoric links (that can't be described with context unification) are provided, and strict/sloppy ambiguities are captured correctly. It is likely that a complete algorithm for CLLS can still be implemented much more efficiently than even our modified algorithm for context unification. For example, dominance constraints (which are decidable) can be solved by efficient constraint propagation techniques and independently from parallelism constraints, which reintroduce the expressiveness (and complexity) of context unification.

5 Conclusion

In this paper, we have evaluated the adequacy of context unification as a formalism for the underspecified treatment of scope ambiguities and parallelism. To this end, we had to introduce a new algorithm for context unification that is better suited for the use in linguistics because it only finds minimal solutions and has acceptable computational properties. It is incomplete for the full problem, but finds all linguistically relevant solutions. With this unification algorithm, context unification is an elegant formalism that offers a broad coverage of semantic phenomena and can be made to run efficiently for small examples, but is very sensitive to the problem size, cannot easily be made to take anaphora into account, and has problems with variable binding. As such, it is an important step towards the development of a new formalism, CLLS, that solves these problems and even improves on the efficiency.

Acknowledgments

The author is indebted to Joachim Niehren and Peter Ruhrberg for their contributions to the work reported in this article and to Markus Egg and Joachim Niehren as well as three anonymous referees for many helpful comments about the article itself.

References

[Crouch, 1995] Richard Crouch. Ellipsis and quantification: A substitutional approach. In *Proceedings 7th European ACL*, pages 229–236, Dublin, 1995.

[Debusmann et al., 1998] R. Debusmann, M. Egg, A. Koller, K. Konrad, J. Niehren, G. Schaefer, S. Thater, V. Winter, and F. Xu. CHOLI: A natural language system for semantic construction and evaluation. CLAUS report, Universität des Saarlandes, Saarbrücken, 1998.

[Egg et al., 1998] Markus Egg, Joachim Niehren, Peter Ruhrberg, and Feiyu Xu. Constraints over lambda-structures in semantic underspecification. In *Proceedings of the 17th International Conference on Computational Linguistics (COLING '98) and 36th Annual Meeting of the Association for Computational Linguistics (ACL '98)*, Montreal, Canada, August 1998. To appear.

[Hirschbühler, 1982] Paul Hirschbühler. VP deletion and across the board quantifier scope. In J. Pustejovsky and P. Sells, editors, *NELS 12*. University of Massachusetts, 1982.

[Hobbs & Shieber, 1987] Jerry R. Hobbs and Stuart Shieber. An algorithm for generating quantifier scoping. *Computational Linguistics*, 13:47–63, 1987.

[Höhl, 1997] Claudia Höhl. Funktionale Implementierung eines Unfikationsalgorithmus für Stratified Second-Order Terme. Master's thesis, Johann Wolfgang Goethe-Universität, Frankfurt, Germany, 1997.

[Lévy, 1996] Jordi Lévy. Linear second order unification. In *Proceedings of the Conference on Rewriting Techniques and Applications*, volume 1103 of *LNCS*, pages 332–346. Springer-Verlag, 1996.

[Makanin, 1977] G. S. Makanin. The problem of solvability of equations in a free semigroup. *Soviet Akad. Nauk SSSR*, 223(2), 1977.

[Niehren et al., 1997a] J. Niehren, M. Pinkal, and P. Ruhrberg. A uniform approach to underspecification and parallelism. In *Proceedings of the 35th Annual Meeting of the Association of Computational Linguistics (ACL)*, pages 410–417, Madrid, 1997.

[Niehren et al., 1997b] J. Niehren, M. Pinkal, and P. Ruhrberg. On equality upto constraints over finite trees, context unification, and one-step rewriting. In *Proceedings of the 14th International Conference on Automated Deduction*, Townsville, 1997. Springer-Verlag.

[Niehren & Koller, 1998] Joachim Niehren and Alexander Koller. Dominance constraints in context unification. Submitted. `http://www.ps.uni-sb.de/Papers/abstracts/Dominance.html`.

[Oz, 1997] The Oz Programming System, 1997. Programming Systems Lab, Universität des Saarlandes: `http://www.ps.uni-sb.de/www/oz2/`.

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

[Schmidt-Schauß & Schulz, 1998] Manfred Schmidt-Schauß and Klaus Schulz. On the exponent of periodicity of minimal solutions of context equations. In Tobias Nipkow, editor, *8th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science, Rutgers University, NJ, USA, 1998. Springer-Verlag. To appear.

[Shieber et al., 1996] Stuart Shieber, Fernando Pereira, and Mary Dalrymple. Interaction of scope and ellipsis. *Linguistics and Philosophy*, 19:527–552, 1996.

[Smolka, 1995] Gert Smolka. The Oz Programming Model. In Jan van Leeuwen, editor, *Computer Science Today*, volume 1000 of *LNCS*, pages 324–343. Springer Verlag, 1995.

[van Deemter & Peters, 1996] Kees van Deemter and Stanley Peters, editors. *Semantic Ambiguity and Underspecification*. CSLI, Stanford, 1996.