

# Evaluating Context Unification for Semantic Underspecification.

## Extended Abstract

**Abstract.** Context unification has been proposed as a powerful formalism for the uniform and underspecified treatment of scope ambiguities and parallelism. The readings of a sentence are represented as the minimal solutions of tree constraints; eventually, all these solutions are enumerated by a unification algorithm. Complete unification algorithms have an immense time complexity and overgenerate strongly, which makes them useless for the linguistic application. We develop an algorithm for context unification that alleviates these computational problems; it is incomplete for full context unification, but finds all linguistically relevant solutions and essentially solves the overgeneration problem. The new algorithm has been implemented and used to evaluate context unification as a tool for semantic analysis.

## 1 Introduction

Among the most important issues of current research in natural-language semantics are the treatment of (quantifier) scope ambiguities and of parallelism, e.g. of VP ellipses. Recently, these phenomena have been investigated within the framework of underspecification [16]. 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.

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” [4] like that in example (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. Of the formalisms that capture this interaction (e.g. [14]), only a few [1, 9] have managed to deal with them in terms of a single mechanism. The approach in [9] allows a particularly transparent treatment of these phenomena. Following [12, 1] and in contrast to [14], it distinguishes between the actual semantic representation of a sentence (the object-level term) and its (meta-level) description; only the description is derived from the syntax. Object-level terms, typically lambda terms, are considered as trees and are described by so-called *context constraints*, second-order formulae talking about trees. The different readings of a sentence are represented by the minimal solutions of such a constraint. 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. This is a stronger condition than mere soundness and completeness of the algorithm, which only relate an algorithm to the set of all solutions. 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 [7, 13, 8]. Unfortunately, each of these algorithms is either incredibly complex, is not complete for the kind of constraints that occur in semantic analysis, has an unacceptable runtime, or enumerates large numbers of non-minimal solutions. This means that we have to come up with an algorithm of our own that is better suited for our application 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 some 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 constructors  $f$ , and *context variables*  $C$ . A *context constraint* is a conjunction of equations between terms  $t$  with the following formal syntax:

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

Semantically, terms are interpreted as ground terms, 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. Context variables  $C$  denote functions from trees to trees that insert their arguments into fixed *contexts*, trees with exactly one hole. *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  $\lambda X.f(X, b)$ .

What makes context unification interesting for natural-language semantics is that we can use it to describe lambda terms. More precisely, we represent lambda terms as first-order terms with special constructors. 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  $\text{lam}_x(t)$ ; occurrences of the abstracted variable  $x$  are replaced by the nullary constructor  $\text{var}_x$ .

By way of example, consider Example (1) from above. The semantics  $X_s$  and  $X_t$  of the source and target sentence are described by the context constraint (3).<sup>1</sup>

---

<sup>1</sup> The constraint (3) has an unwanted third minimal solution. In [9], it is excluded by modifying the constraint (cf. the transition from Example 6 to Example 6’ in that paper). Since the extra solution is

- (2) Every researcher read some books. Every student did, too.
- (3)  $X_s = C_1(\text{every@res@lam}_x(C_2(\text{read@var}_y @ \text{var}_x)))$   
 $X_s = C_3(\text{some@book@lam}_y(C_4(\text{read@var}_y @ \text{var}_x)))$   
 $X_s = C(\text{every@res}) \quad X_t = C(\text{every@student})$
- (4)  $X_s = \text{every@res@lam}_x(\text{some@book@lam}_y(\text{read@var}_y @ \text{var}_x))$   
 $X_t = \text{every@student@lam}_x(\text{some@book@lam}_y(\text{read@var}_y @ \text{var}_x))$
- (5)  $X_s = \text{some@book@lam}_y(\text{every@res@lam}_x(\text{read@var}_y @ \text{var}_x))$   
 $X_t = \text{some@book@lam}_y(\text{every@student@lam}_x(\text{read@var}_y @ \text{var}_x))$

The first three equations of (3) describe the scope ambiguity in the source sentence. They have exactly two minimal solutions, shown in the first lines of the solutions (4) and (5). In one of these solutions, *every@res* has scope over *some@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 (3) express that the terms  $X_s$  and  $X_t$  are almost the same, but  $X_t$  has *every@student* in one location where  $X_s$  has *every@res*. In particular, these constraints enforce that however 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.

### 3 An algorithm for context unification

Several algorithms for context unification and various related problems have been proposed. Unfortunately, each of them has a problem that restricts its usefulness for our purposes. An algorithm that is complete for the entire language [8, Appendix B] has an enormous complexity; generally, the decidability of full context unification is an open problem. Algorithms for sublanguages either do not fully cover our domain of application [7] or are simply too complex [13, 6].

The algorithm we will present here is based on the one from [8, Appendix B] because it is particularly simple. It is a nondeterministic system of rewriting rules, i.e., in some situations, there is a choice between different applicable rules. 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 [15, 11] due to its powerful searching primitives.

As outlined above, the original algorithm has two essential drawbacks. Even for a unification problem as simple as (3), the search space has tens of thousands of nodes and takes very long to explore. Even worse, the algorithm overgenerates strongly: it enumerates thousands of solutions, most of which are linguistically unwanted because they are non-minimal.

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

1. To reduce the size of the search space, we remove the two 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.

The result is shown in Figure 1. We take over the notation of [8]; application of a rule  $A \rightarrow B \mid C$  replaces the atomic constraint  $A$  by  $B$ , binding variables as described in  $C$ . The

---

ruled out by the algorithm we present in the next section and the correct constraint is more difficult to understand, we ignore this problem here. This looks like a wild hack, but is soundly justified and shows how well our algorithm matches the linguistic intuition.

(Subst)	$X=t \longrightarrow \text{true}$ if $X \notin V(t) \mid X \mapsto t$
(Decomp)	$a(t_1, \dots, t_n)=a(t'_1, \dots, t'_n) \longrightarrow \bigwedge_{i=1..n} t_i=t'_i \mid Id$
(Proj)	$t=C(t') \longrightarrow t=t' \mid C \mapsto \lambda X.X$
(Imit)	$a(t_1, \dots, t_n)=C(t') \longrightarrow t_i=C'(t') \mid C \mapsto \lambda X.a(t_1, \dots, t_{i-1}, C'(X), t_{i+1}, \dots, t_n)$

**Fig. 1.** The modified algorithm for context unification.

algorithm terminates if the constraint has been reduced to **true**. The rules are, for the most part, adaptations of the standard rules of the world of higher-order unification; for example, Decomposition breaks down the unification of two terms with the same (rigid) head to those of their subterms.

Especially the first change that we made to the algorithm deserves some discussion. The most problematic rules are the two Flex/Flex rules that are applicable in situations where the heads of the terms on both sides of an equation are context variables. For now, it must suffice to say that the removal of the Flex/Flex 1 rule has no effect on the completeness of the algorithm; applications of this rule can essentially be replaced by sequences of Projection and Imitation rules. The removal of Flex/Flex 2 renders the algorithm incomplete for full context unification. However, [10] argues that this rule is never needed for the class of context constraints occurring in semantic analysis. This means that all linguistically relevant solutions are still found.

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

$$\begin{aligned} \tau &::= e \mid t \mid \tau_1 \rightarrow \tau_2 \\ \kappa &::= (\tau_1, \dots, \tau_n) \rightsquigarrow \tau. \end{aligned}$$

## 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. This is facilitated even further because our implementation has been integrated with an HPSG front-end that produces context constraints from natural-language input [2]. It turns out that there is both good and bad news.

On one hand, context unification correctly captures scope ambiguities, ellipses, and their interaction. The algorithm from the previous section enumerates exactly the minimal solutions for almost all examples that we have tested it on and has an acceptable performance for small examples. Consider our running example (6).

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

If we feed its formalization, (3), to the implemented algorithm, the program explores a search space of 168 nodes and finds exactly the two solutions (4) and (5). This takes less than two

seconds. The implementation also correctly handles more complex scope ambiguities such as the well-known benchmark problem (7) from [5]:

(7) 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 wanted. Our implementation finds exactly these five readings. It explores a search space of 6.500 nodes, which takes less than half a minute.

On the other hand, by making possible the examination of larger problems, the implementation shows that the complexity of context unification makes even restricted algorithms very sensitive towards the size of the problem. Besides, some more fundamental problems remain, such as the correct treatment of alpha conversion. A promising approach, whose development has been stimulated by our results, is CLLS [3], which models lambda binding by explicit binding relations between tree nodes instead of via variable names. CLLS should also allow for more efficient algorithms.

## 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 incomplete for the full problem, but finds all linguistically relevant solutions. It is much better suited for the use in linguistics because it essentially solves the overgeneration problem that previous algorithms had and has acceptable computational properties. 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.

## References

1. Richard Crouch. Ellipsis and quantification: A substitutional approach. In *Proceedings 7th European ACL*, pages 229–236, Dublin, 1995.
2. R. Debusmann, M. Egg, A. Koller, K. Konrad, J. Niehren, G. Schaefer, S. Thater, V. Winter, and F. Xu. A natural language system for semantic construction and evaluation. CLAUS report, Universität des Saarlandes, Saarbrücken, 1998. To appear.
3. Markus Egg, Joachim Niehren, Peter Ruhrberg, and Feiyu Xu. Constraints over lambda structures in semantic underspecification. Submitted. <http://www.ps.uni-sb.de/Papers/abstracts/CLLS-98.html>.
4. Paul Hirschbühler. VP deletion and across the board quantifier scope. In J. Pustejovsky and P. Sells, editors, *NELS 12*. University of Massachusetts, 1982.
5. Jerry R. Hobbs and Stuart Shieber. An algorithm for generating quantifier scoping. *Computational Linguistics*, 13:47–63, 1987.
6. Claudia Höhl. Funktionale Implementierung eines Unifikationsalgorithmus für Stratified Second-Order Terme. Master’s thesis, Johann Wolfgang Goethe-Universität, Frankfurt, Germany, 1997.
7. 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.
8. J. Niehren, M. Pinkal, and P. Ruhrberg. On equality up-to constraints over finite trees, context unification, and one-step rewriting. In *Proceedings of the 14th International Conference on Automated Deduction*, Townsville, 1997. Springer-Verlag.

9. 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.
10. Joachim Niehren and Alexander Koller. Dominance constraints in context unification. Submitted. <http://www.ps.uni-sb.de/Papers/abstracts/Dominance.html>.
11. The Oz Programming System, 1997. Programming Systems Lab, Universität des Saarlandes: <http://www.ps.uni-sb.de/www/oz2/>.
12. Uwe Reyle. Dealing with ambiguities by underspecification: Construction, representation, and deduction. *Journal of Semantics*, 10:123–179, 1993.
13. 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.
14. Stuart Shieber, Fernando Pereira, and Mary Dalrymple. Interaction of scope and ellipsis. *Linguistics and Philosophy*, 19:527–552, 1996.
15. Gert Smolka. The Oz Programming Model. In Jan van Leeuwen, editor, *Computer Science Today*, volume 1000 of *LNCS*, pages 324–343. Springer Verlag, 1995.
16. Kees van Deemter and Stanley Peters, editors. *Semantic Ambiguity and Underspecification*. CSLI, Stanford, 1996.