Underspecified Beta Reduction

Manuel Bodirsky
Katrin Erk
Joachim Niehren
Programming Systems Lab
Saarland University
D-66041 Saarbrücken, Germany
{bodirsky|erk|niehren}@ps.uni-sb.de

Alexander Koller
Department of Computational Linguistics
Saarland University
D-66041 Saarbrücken, Germany
koller@coli.uni-sb.de

Abstract

For ambiguous sentences, traditional semantics construction produces large numbers of higher-order formulas, which must then be β-reduced individually. Underspecified versions can produce compact descriptions of all readings, but it is not known how to perform β-reduction on these descriptions. We show how to do this using β-reduction constraints in the constraint language for λ-structures (CLLS).

1 Introduction

Traditional approaches to semantics construction (Montague, 1974; Cooper, 1983) employ formulas of higher-order logic to derive semantic representations compositionally; then β-reduction is applied to simplify these representations. When the input sentence is ambiguous, these approaches require all readings to be enumerated and β-reduced individually. For large numbers of readings, this is both inefficient and unelegant.

Existing *underspecification* approaches (Reyle, 1993; van Deemter and Peters, 1996; Pinkal, 1996; Bos, 1996) provide a partial solution to this problem. They delay the enumeration of the readings and represent them all at once in a single, compact *description*. An underspecification formalism that is particularly well suited for describing higher-order formulas is the Constraint Language for Lambda Structures, CLLS (Egg et al., 2001; Erk et al., 2001). CLLS descriptions can be derived compositionally and have been used to deal with a rich class of linguistic phenomena (Koller et al., 2000; Koller and Niehren, 2000). They are based on dominance constraints (Marcus et al., 1983; Rambow et al., 1995) and extend them with parallelism (Erk and Niehren, 2000) and binding constraints.

However, lifting β-reduction to an operation on underspecified descriptions is not trivial, and to our knowledge it is not known how this can be done. Such an operation – which we will call *underspecified β-reduction* – would essentially β-reduce all described formulas at once by deriving a description of the reduced formulas. In this paper, we show how underspecified β-reductions can be performed in the framework of CLLS.

Our approach extends the work presented in (Bodirsky et al., 2001), which defines *β-reduction constraints* and shows how to obtain a complete solution procedure by reducing them to parallelism constraints in CLLS. The problem with this previous work is that it is often necessary to perform local disambiguations. Here we add a new mechanism which, for a large class of descriptions, permits us to perform underspecified β-reduction steps without disambiguating, and is still complete for the general problem.

Plan. We start with a few examples to show what underspecified β-reduction should do, and why it is not trivial. We then introduce CLLS and β-reduction constraints. In the core of the paper we present a procedure for underspecified β-reduction and apply it to illustrative examples.

2 Examples

In this section, we show what underspecified β-reduction should do, and why the task is nontrivial. Consider first the ambiguous sentence *Every student didn't pay attention*. In first-order logic, the two readings can be represented as

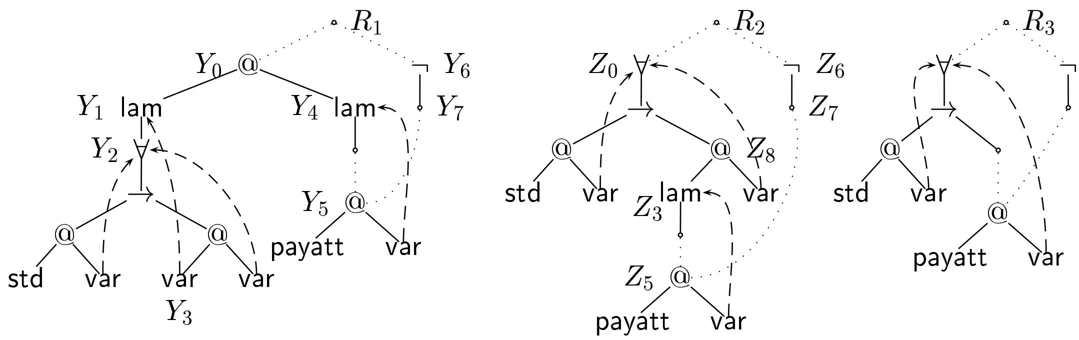


Figure 1: Underspecified β-reduction steps for ‘Every student did not pay attention’

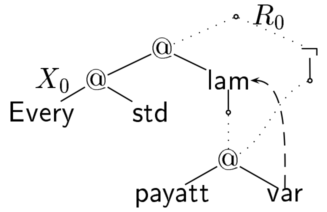


Figure 2: Description of ‘Every student did not pay attention’

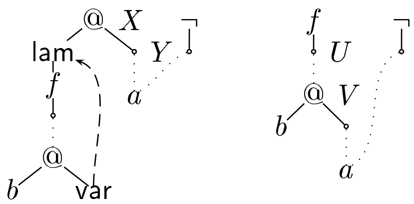


Figure 3: Problems with rewriting of descriptions

$$\forall x(\mathsf{std}\ x \rightarrow \neg(\mathsf{payatt}\ x))$$
$$\neg(\forall x(\mathsf{std}\ x \rightarrow \mathsf{payatt}\ x))$$

A classical compositional semantics construction first derives these two readings in the form of two HOL-formulas:

$$(\mathsf{Every\ std})\ \lambda x\ (\neg \mathsf{payatt}\ x)$$
$$\neg((\mathsf{Every\ std})\ \lambda x\ (\mathsf{payatt}\ x))$$

where Every is an abbreviation for the term

$$\mathsf{Every} = \lambda P \lambda Q(\forall x(P\ x \rightarrow Q\ x))$$

An underspecified description of both readings is shown in Figure 2. For now, notice that the graph has all the symbols of the two HOL formulas as node labels, that variable binding is indicated by dashed arrows, and that there are dotted lines indicating an “outscopes” relation; we will fill in the details in Section 3.

Now we want to reduce the description in Figure 2 as far as possible. The first β-reduction step, with the redex at X_0 is straightforward. Even though the description is underspecified, the reducing part is a completely known λ-term. The result is shown on the left-hand side of Figure 1. Here we have just one redex, starting at Y_0, which binds a single variable. The next reduction step is less obvious: The $\neg$ operator could either belong to the context (the part between R_1 and Y_0) or to the argument (below Y_4). Still, it is not difficult to give a correct description of the result: it is shown in the middle of Fig. 1. For the final step, which takes us to the rightmost description, the redex starts at Z_8. Note that now the $\neg$ might be part of the body or part of the context of this redex. The end result is precisely a description of the two readings as first-order formulas.

So far, the problem does not look too difficult. Twice, we did not know what exactly the parts of the redex were, but it was still easy to derive correct descriptions of the reducts. But this is not always the case. Consider Figure 3, an abstract but simple example. In the left description, there are two possible positions for the $\neg$: above X or below Y. Proceeding naïvely as above, we arrive at the right-hand description in Fig. 3. But this description is also satisfied by the term $f(\neg(b(a)))$, which cannot be obtained by reducing any of the terms described on the left-hand side. More generally, the naïve “graph rewriting” approach is unsound; the resulting descriptions can have too many readings. Similar problems arise in (more complicated) examples from semantics, such as the coordination in Fig. 8.

The underspecified β-reduction operation we propose here does not *rewrite* descriptions. Instead, we *describe* the result of the step using a

"β-reduction constraint" that ensures that the reduced terms are captured correctly. Then we use a saturation calculus to make the description more explicit.

3 Tree descriptions in CLLS

In this section, we briefly recall the definition of the constraint language for λ-structures (CLLS). A more thorough and complete introduction can be found in (Egg et al., 2001).

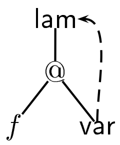
We assume a signature $\Sigma = \{f, g, \dots\}$ of function symbols, each equipped with an arity $\mathsf{ar}(f) \geq 0$. A *tree* θ consists of a finite set of nodes $\pi \in D_\theta$, each of which is labeled by a symbol $L_\theta(\pi) \in \Sigma$. Each node π has a sequence of children $\pi 1, \dots, \pi n \in D_\theta$ where $n = \mathsf{ar}(L_\theta(\pi))$ is the arity of the label of π. A single node ϵ, the *root* of θ, is not the child of any other node.

3.1 Lambda structures

The idea behind λ-structures is that a λ-term can be considered as a pair of a tree which represents the structure of the term and a *binding function* encoding variable binding. We assume Σ contains symbols var (arity 0, for variables), lam (arity 1, for abstraction), @ (arity 2, for application), and analogous labels for the logical connectives.

Definition 1. *A* λ-structure τ *is a pair* (θ, λ) *of a tree* θ *and a binding function* λ *that maps every node* π *with label* var *to a node with label* lam, $\forall$, *or* $\exists$ *dominating* π.

The binding function λ explicitly maps nodes representing variables to the nodes representing their binders. When we draw λ-structures, we represent the binding function using dashed arrows, as in the picture to the right, which represents the λ-term $\lambda x.f(x)$.



A λ-structure corresponds uniquely to a closed λ-term modulo α-renaming. We will freely consider λ-structures as first-order model structures with domain D_θ. This structure defines the following relations. The labeling relation $\pi{:}f(\pi_1, \dots, \pi_n)$ holds in θ if $L_\theta(\pi) = f$ and $\pi_i = \pi i$ for all $1 \leq i \leq n$. The dominance relation $\pi \triangleleft^* \pi'$ holds iff there is a path π'' such that $\pi\pi'' = \pi'$. Inequality $\neq$ is simply inequality of nodes; *disjointness* $\pi \perp \pi'$ holds iff neither $\pi \triangleleft^* \pi'$ nor $\pi' \triangleleft^* \pi$.

3.2 Basic constraints

Now we define the constraint language for λ-structures (CLLS) to talk about these relations. X, Y, Z are variables that will denote nodes of a λ-structure.

$$\begin{array}{lcl} \varphi & ::= & X \triangleleft^* Y \mid X {\neq} Y \mid X \perp Y \mid \varphi \wedge \varphi' \\ & \mid & X{:}f(X_1, \dots, X_n) \quad (\mathsf{ar}(f) = n) \\ & \mid & \lambda(X){=}Y \mid \lambda^{-1}(X_0){=}\{X_1, \dots, X_n\} \end{array}$$

A constraint φ is a conjunction of *literals* (for dominance, labeling, etc). We use the abbreviations $X \triangleleft^+ Y$ for $X \triangleleft^* Y \wedge X \neq Y$ and $X = Y$ for $X \triangleleft^* Y \wedge Y \triangleleft^* X$. The λ-binding literal $\lambda(X){=}Y$ expresses that Y denotes a node which the binding function maps to X. The *inverse* λ-binding literal $\lambda^{-1}(X_0){=}\{X_1, \dots, X_n\}$ states that $X_1, \dots, X_n$ denote the *entire* set of variable nodes bound by X_0. A pair (τ, σ) of a λ-structure τ and a variable assignment σ *satisfies* a λ-structure iff it satisfies each literal, in the obvious way.

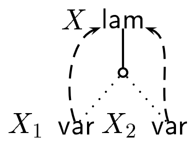


Figure 4: The constraint graph of $\lambda^{-1}(X){=}\{X_1, X_2\} \wedge X \triangleleft^* X_1 \wedge X \triangleleft^* X_2$

We draw constraints as graphs (Fig. 4) in which nodes represent variables. Labels and solid lines indicate labeling literals, while dotted lines represent dominance. Dashed arrows indicate the binding relation; disjointness and inequality literals are not represented. The informal diagrams from Section 2 can thus be read as constraint graphs, which gives them a precise formal meaning.

3.3 Segments and Correspondences

Finally, we define *segments* of λ-structures and *correspondences* between segments. This allows us to define parallelism and β-reduction constraints.

A segment is a contiguous part of a λ-structure that is delineated by several nodes of the structure. Intuitively, it is a tree from which some subtrees have been cut out, leaving behind holes.

Definition 2 (Segments). *A segment* α *of a* λ-*structure* (θ, λ) *is a tuple* $\pi_0/\pi_1 \dots, \pi_n$ *of nodes* *in D_θ such that $\pi_0 \lhd^* \pi_i$ and $\pi_i \perp \pi_j$ hold in θ for $1 \leq i \neq j \leq n$. The* root $r(\alpha)$ *is* π_0*, and* $hs(\alpha) = \pi_1, \ldots, \pi_n$ *is its (possibly empty) sequence of* holes. *The set* $\mathsf{b}(\alpha)$ *of nodes of* α *is*

$$\mathsf{b}(\alpha) = \{\pi \in D_\tau \mid \quad r(\alpha) \lhd^* \pi, \text{ and not } \pi_i \lhd^+ \pi \text{ for all } 1 \leq i \leq n\}$$

To exempt the holes of the segment, we define $\mathsf{b}^-(\alpha) = \mathsf{b}(\alpha) - hs(\alpha)$. *If* $hs(\alpha)$ *is a singleton sequence then we write* $h(\alpha)$ *for the unique hole of* α*, i.e. the unique node with* $h(\alpha) \in hs(\alpha)$.

For instance, $\alpha = \pi_1/\pi_2, \pi_3$ is a segment in Fig. 5; its root is π_1, its holes are π_2 and π_3, and it contains the nodes $\mathsf{b}(\alpha) = \{\pi_1, \pi_5, \pi_2, \pi_3\}$.

Two tree segments α, β *overlap properly* iff $\mathsf{b}^-(\alpha) \cap \mathsf{b}^-(\beta) \neq \emptyset$. The syntactic equivalent of a segment is a *segment term* $X_0/X_1, \ldots X_n$. We use the letters A, B, C, D for them and extend $r(A)$, $hs(A)$, and $h(A)$ correspondingly.

A correspondence function is intuitively an isomorphism between segments, mapping holes to holes and roots to roots and respecting the structures of the trees:

Definition 3. *A* correspondence function *between the segments* α, β *is a bijective mapping* $c : \mathsf{b}(\alpha) \to \mathsf{b}(\beta)$ *such that c maps the* i*-th hole of* α *to the* i*-th hole of* β *for each* i*, and for every* $\pi \in \mathsf{b}^-(\alpha)$ *and every label* f*,*

$$\pi{:}f(\pi 1, \ldots, \pi n) \Leftrightarrow c(\pi){:}f(c(\pi 1), \ldots c(\pi n)).$$

There is at most one correspondence function between any two given segments. The *correspondence literal* $\mathrm{co}(C, D)(X){=}Y$ expresses that a correspondence function c between the segments denoted by C and D exists, that X and Y denote nodes within these segment, and that these nodes are related by c.

Together, these constructs allow us to define *parallelism*, which was originally introduced for the analysis of ellipsis (Egg et al., 2001). The parallelism relation $\alpha \sim \beta$ holds iff there is a correspondence function between α and β that satisfies some natural conditions on λ-binding which we cannot go into here. To model parallelism in the presence of global λ-binders relating multiple parallel segments, Bodirsky et al. (2001) generalize parallelism to *group parallelism*. Group parallelism $(\alpha_1, \ldots, \alpha_n) \sim (\beta_1, \ldots, \beta_n)$ is entailed by the conjunction $\wedge_{i=1}^n \alpha_i \sim \beta_i$ of ordinary parallelisms, but imposes slightly weaker restrictions on λ-binding. By way of example, consider the λ-structure in Fig. 5, where $(\pi_0/\pi_1, \pi_2/\pi_4, \pi_3/) \sim (\pi'_0/\pi'_1, \pi'_1/\pi'_4, \pi'_4/)$ holds.

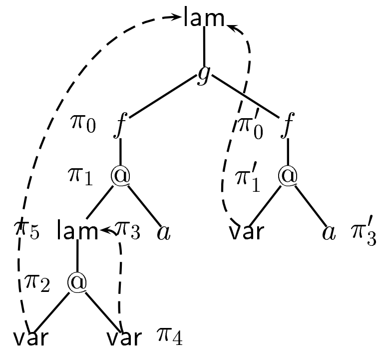


Figure 5: $f\ ((\lambda x.z(x))(a)) \to_\beta f\ (z(a))$

On the syntactic side, CLLS provides *group parallelism literals* $(A_1, \ldots, A_n) \sim (B_1, \ldots, B_n)$ to talk about (group) parallelism.

4 Beta reduction constraints

Correspondences are also used in the definition of *β-reduction constraints* (Bodirsky et al., 2001). A β-reduction constraint describes a single β-reduction step between two λ-terms; it enforces correct reduction even if the two terms are only partially known.

Standard β-reduction has the form

$$C((\lambda x.B)\ A) \to_\beta C(B[x/A]) \quad x \text{ free for } A.$$

The reducing λ-term consists of context C which contains a redex $(\lambda x.B)A$. The redex itself is an occurrence of an application of a λ-abstraction $\lambda x.B$ with body B to argument A. β-reduction then replaces all occurrences of the bound variable x in the body by the argument while preserving the context.

We can partition both redex and reduct into argument, body, and context *segments*. Consider Fig. 5. The λ-structure contains the reducing λ-term $f\ ((\lambda x.z(x))(a))$ starting at π_0. The reduced term can be found at π'_0. Writing γ, γ' for the context, β, β' for the body and α, α' for the argument tree segments of the reducing and the reduced term, respectively, we find

$$\begin{array}{lll} \gamma = \pi_0/\pi_1 & \beta = \pi_2/\pi_4 & \alpha = \pi_3/ \\ \gamma' = \pi'_0/\pi'_1 & \beta' = \pi'_1/\pi'_3 & \alpha' = \pi'_3/ \end{array}$$

Because we have both the reducing term and the reduced term as parts of the same λ-structure, we can express the fact that the structure below π'_0 can be obtained by β-reducing the structure below π_0 by requiring that α corresponds to α', β to β', and γ to γ', again modulo binding. This is indeed true in the given λ-structure, as we have seen above.

More generally, we define the *β-reduction relation*

$$(\gamma, \beta, \alpha) \xrightarrow{\beta} (\gamma', \beta', \alpha'_1, \ldots, \alpha'_n)$$

for a body β with n holes (for the variables bound in the redex). The β-reduction relation holds iff two conditions are met: (γ, β, α) must form a reducing term, and the structural equalities that we have noted above must hold between the tree segments. The latter can be stated by the following group parallelism relation, which also represents the correct binding behaviour:

$$(\gamma, \beta, \alpha, \ldots, \alpha) \sim (\gamma', \beta', \alpha'_1, \ldots, \alpha'_n)$$

Note that any λ-structure satisfying this relation must contain both the reducing and the reduced term as substructures. Incidentally, this allows us to accommodate for global variables in λ-terms; Fig. 5 shows this for the global variable z.

We now extend CLLS with β-reduction constraints

$$(C, B, A) \xrightarrow{\beta} (C', B', A'_1, \ldots, A'_n),$$

which are interpreted by the β-reduction relation.

The reduction steps in Section 2 can all be represented correctly by β-reduction constraints. Consider e.g. the first step in Fig. 1. This is represented by the constraint $(R_1/Y_0, Y_2/Y_3, Y_4/) \xrightarrow{\beta} (R_2/Z_0, Z_0/Z_3, Z_3/)$. The entire middle constraint in Fig. 1 is entailed by the β-reduction literal. If we learn in addition that e.g. $Y_7 \triangleleft^* Y_0$, the β-reduction literal will entail $Z_7 \triangleleft^* Z_0$ because the segments must correspond. This correlation between parallel segments is the exact same effect (quantifier parallelism) that is exploited in the CLLS analysis of "Hirschbühler sentences", where ellipses and scope interact (Egg et al., 2001).

β-reduction constraints also represent the problematic example in Fig. 3 correctly. The spurious solution of the right-hand constraint does not satisfy the β-reduction constraint, as the bodies would not correspond.

```
usb(φ, X) =
  if all syntactic redexes in φ below X
    are reduced then return (φ, X)
  else
     pick a formula redex_Y(C, B, A) in φ
       that is unreduced, with X=r(C) in φ
     add (C, B, A) -β-> (C', B', A'_1, ... , A'_n)
       to φ where C', B', A'_1, ... , A'_n are new
       segment terms with fresh variables
     add X ⊥ r(C') to φ
     for all φ' ∈ solve(φ) do usb(φ', r(C'))
  end
```

Figure 6: Underspecified β-reduction

5 Underspecified Beta Reduction

Having introduced β-reduction constraints, we now show how to process them. In this section, we present the procedure `usb`, which performs a sequence of underspecified β-reduction steps on CLLS descriptions. This procedure is parameterized by another procedure `solve` for solving β-reduction constraints, which we discuss in the following section.

A syntactic redex in a constraint φ is a subformula of the following form:

$$\mathsf{redex}_Y(C, B, A) =_{\mathsf{df}} h(C){:}@(Y, r(A)) \wedge Y{:}\mathsf{lam}(r(B)) \wedge \lambda^{-1}(Y) = hs(B)$$

A context C of a redex must have a unique hole $h(C)$. An *n-ary* redex has n occurrences of the bound variable, i.e. the length of $hs(B)$ is n. We call a redex *linear* if $n = 1$.

The algorithm `usb` is shown in Figure 6. It starts with a constraint φ and a variable X, which denotes the root of the current λ-term to be reduced. (For example, for the redex in Fig. 2, this root would be R_0.) The procedure then selects an unreduced syntactic redex and adds a description of its reduct at a disjoint position. Then the `solve` procedure is applied to resolve the β-reduction constraint, at least partially. If it has to disambiguate, it returns one constraint for each reading it finds. Finally, `usb` is called recursively with the new constraint and the root variable of the new λ-term.

Intuitively, the `solve` procedure adds entailed literals to φ, making the new β-reduction literal more explicit. When presented with the left-hand constraint in Fig. 1 and the root variable R_1, `usb` will add a β-reduction constraint for the redex at Y_1; then `solve` will derive the middle constraint. Finally, `usb` will call itself recursively with the new root variable R_2 and try to resolve the redex at Z_3, etc. The partial solving steps do essentially the same as the naïve graph rewriting approach in this case; but the new algorithm will behave differently on problematic constraints as in Fig. 3.

6 A single reduction step

In this section we present a procedure `solve` for solving β-reduction constraints. We go through several examples to illustrate how it works. We have to omit some details for lack of space; they can be found in (Bodirsky et al., 2001).

The aim of the procedure is to make explicit information that is implicit in β-reduction constraints: it introduces new corresponding variables and copies constraints from the reducing term to the reduced term.

We build upon the solver for β-reduction constraints from (Bodirsky et al., 2001). This solver is complete, i.e. it can enumerate all solutions of a constraint; but it disambiguates a lot, which we want to avoid in underspecified β-reduction. We obtain an alternative procedure `solve` by disabling all rules which disambiguate and adding some new non-disambiguating rules. This allows us to perform a complete underspecified β-reduction for many examples from underspecified semantics without disambiguating at all. In those cases where the new rules alone are not sufficient, we can still fall back on the complete solver.

6.1 Saturation

Our constraint solver is based on *saturation* with a given set of *saturation rules*. Very briefly, this means that a constraint is seen as the set of its literals, to which more and more literals are added according to *saturation rules*. A saturation rule of the form $\varphi_0 \to \bigvee_{i=1}^{n} \varphi_i$ says that we can add one of the φ_i to any constraint that contains at least the literals in φ_0. We only apply rules where each possible choice adds new literals to the set; a constraint is *saturated* under a set S of saturation rules if no rule in S can add anything else. `solve` returns the set of all possible saturations of its input. If the rule system contains nondeterministic *distribution* rules, with $n > 1$, this set can be non-singleton; but the rules we are going to introduce are all deterministic *propagation* rules (with $n = 1$).

6.2 Solving Beta Reduction Constraints

The main problem in doing underspecified β-reduction is that we may not know to which part of a redex a certain node belongs (as in Fig. 1). We address this problem by introducing *underspecified correspondence literals* of the form

$$\mathrm{co}(\{(C_1, D_1), \ldots, (C_n, D_n)\})(X)=Y.$$

Such a literal is satisfied if the tree segments denoted by the C's and by the D's do not overlap properly, and there is an i for which $\mathrm{co}(C_i, D_i)(X) = Y$ is satisfied.

In Fig. 7 we present the rules UB for underspecified β-reduction; the first five rules are the core of the algorithm. To keep the rules short, we use the following abbreviations (with $1 \leq i \leq n$):

$$\begin{array}{rll} \text{beta} & =_{\text{def}} & (C, B, A) \xrightarrow{\beta} (C', B', A'_1, \ldots, A'_n) \\ \text{co}_i & =_{\text{def}} & \text{co}(\{(C, C'), (B, B'), (A, A'_i)\}) \end{array}$$

The procedure `solve` consists of UB together with the propagation rules from (Bodirsky et al., 2001). The rest of this section shows how this procedure operates and what it can and cannot do.

First, we discuss the five core rules. Rule (Beta) states that whenever the β-reduction relation holds, group parallelism holds, too. (This allows us to fall back on a complete solver for group parallelism.) Rule (Var) introduces a new variable as a correspondent of a redex variable, and (Lab) and (Dom) copy labeling and dominance literals from the redex to the reduct. To understand the exceptions they make, consider e.g. Fig. 5. Every node below π_0 has a correspondent in the reduct, except for π_3. Every labeling relation in the redex also holds in the reduct, except for the labelings of the @-node π_1, the lam-node π_3, and the var-node π_4. For the variables that possess a correspondent, all dominance relations in the redex hold in the reduct too. The rule (λ.Inv) copies inverse λ-binding literals, i.e. the information that all variables bound by a λ-binder are known. For now,

(Beta)	$(C,B,A) \xrightarrow{\beta} (C',B',A'_1,\ldots,A'_n) \to (C,B,A,\ldots,A) \sim (C',B',A'_1,\ldots,A'_n)$	
(Var)	$\mathrm{beta} \wedge \mathsf{redex}_Y(C,B,A) \wedge r(C) \triangleleft^* Z \wedge Z \neq Y \to \exists Z'.\mathrm{co}_i(Z)=Z'$	
(Lab)	$\mathrm{beta} \wedge \mathsf{redex}_Y(C,B,A) \wedge Z_0{:}f(Z_1,\ldots,Z_\ell) \wedge \bigwedge_{k=0}^{\ell} \mathrm{co}_i(Z_k)=Z'_k \wedge Z_0 \neq h(C) \wedge Z_0 \notin hs(B) \to Z'_0{:}f(Z'_1,\ldots,Z'_\ell)$	
(Dom)	$\mathrm{beta} \wedge \bigwedge_{k=1}^{2} \mathrm{co}_i(Z_k)=Z'_k \wedge Z_1 \triangleleft^* Z_2 \to Z'_1 \triangleleft^* Z'_2$	
(λ.Inv)	$\mathrm{beta} \wedge \mathsf{redex}_Y(C,B,A) \wedge \lambda^{-1}(Z_0)=\{Z_1,\ldots,Z_m\} \wedge \bigwedge_{k=0}^{m} \mathrm{co}_1(Z_k)=Z'_k \to \lambda^{-1}(Z'_0)=\{Z'_1,\ldots,Z'_m\}$	redex linear
(Par.part)	$\mathrm{beta}_n \wedge \mathrm{co}_i(Z)=Z' \wedge Y \in \mathsf{b}(A) \wedge Z \triangleleft^* Y \to Z' \notin \mathsf{i}(B')$	$1 \leq i \leq n$
(Par.all)	$\mathrm{co}(\{(D,D'),\ldots\})(Z)=Z' \wedge Z \in \mathsf{b}(D) \to Z' \in \mathsf{b}(D') \wedge \mathrm{co}(D,D')(Z)=Z'$	

Figure 7: New saturation rules UB for constraint solving during underspecified β-reduction.

it is restricted to linear redexes; for the nonlinear case, we have to take recourse to disambiguation.

It can be shown that the rules in UB are *sound* in the sense that they are valid implications when interpreted over λ-structures.

6.3 Some Examples

To see what the rules do, we go through the first reduction step in Fig. 1. The β-reduction constraint that belongs to this reduction is

$(C,B,A) \xrightarrow{\beta} (C',B',A'_1)$ with
$C = R_1/Y_0, \quad B = Y_1/Y_3, \quad A = Y_4/,$
$C' = R_2/Z_0, \quad B' = Z_0/Z_3, \quad A'_1 = Z_3/$

Now saturation can add more constraints, for example the following:

(1)	$Y_6 \neq Y_1$		(5)	$Y_6 \neq Y_3$	
(2)	$Y_7 \neq Y_1$		(6)	$Z_6{:}\neg(Z_7)$	(Lab)
(3)	$\exists Z_6.\mathrm{co}_1(Y_6)=Z_6$	(Var)	(7)	$R_2 \triangleleft^* Z_6$	(Dom)
(4)	$\exists Z_7.\mathrm{co}_1(Y_7)=Z_7$	(Var)			

We get (1), (2), (5) by propagation rules from (Bodirsky et al., 2001): variables bearing different labels must be different. Now we can apply (Var) to get (3) and (4), then (Lab) to get (6). Finally, (7) shows one of the dominances added by (Dom). Copies of all other variables and literals can be computed in a completely analogous fashion. In particular, copying gives us another redex starting at Z_8, and we can continue with the algorithm `usb` in Figure 6.

Note what happens in case of a nonlinear redex, as in the left picture of Fig. 8: as the redex is 2-ary, the rules produce two copies of the $\neg$ labeling constraint, one via co_1 and one via co_2. The result is shown on the right-hand side of the figure. We will return to this example in a minute.

6.4 More Complex Examples

The last two rules in Fig. 7 enforce consistency between scoping in the redex and scoping in the reduct. The rules use literals that were introduced in (Bodirsky et al., 2001), of the forms $X \in \mathsf{b}(A)$, $X \notin \mathsf{i}(B)$, etc., where A, B are segment terms. We take $X \in \mathsf{b}(A)$ to mean that X must be inside the tree segment denoted by A, and we take $X \in \mathsf{i}(B)$ (i for 'interior') to mean that $X \in \mathsf{b}(B)$ and X denotes neither the root nor a hole of B.

As an example, reconsider Fig. 3: by rule (Par.part), the reduct (right-hand picture of Fig. 3) cannot represent the term $f(\neg(b(a)))$ because that would require the $\neg$ operator to be in $\mathsf{i}(B')$.

Similarly in Fig. 8, where we have introduced two copies of the $\neg$ label. If the $\neg$ in the redex on the left ends up as part of the context, there should be only one copy in the reduct. This is brought about by the rule (Par.all) and the fact that correspondence is a function (which is enforced by rules from (Erk et al., 2001) which are part of the solver in (Bodirsky et al., 2001)). Together, they can be used to infer that Z_0 can have only one correspondent in the reduct context.

7 Conclusion

In this paper, we have shown how to perform an underspecified β-reduction operation in the CLLS framework. This operation transforms underspecified descriptions of higher-order formulas into descriptions of their β-reducts. It can be used to essentially β-reduce all readings of an ambiguous sentence at once.

It is interesting to observe how our underspecified β-reduction interacts with parallelism constraints that were introduced to model ellipses. Consider the elliptical three-reading example "Peter sees a loophole. Every lawyer does too." Under the standard analysis of ellipsis in CLLS (Egg et al., 2001), "Peter" must be represented as a generalized quantifier to obtain all three readings. This leads to a spurious ambigu-

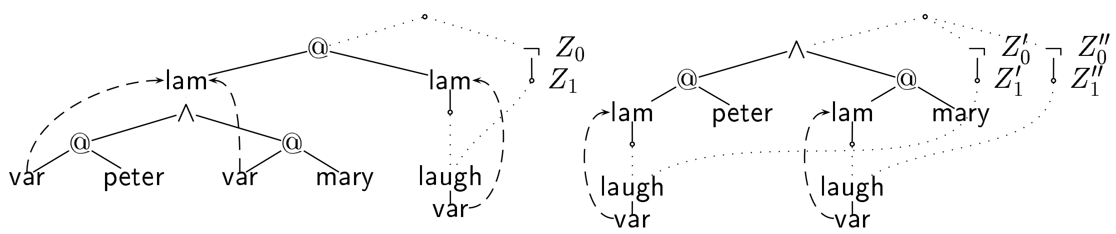


Figure 8: "Peter and Mary do not laugh."

ity in the source sentence, which one would like to get rid of by β-reducing the source sentence. Our approach can achieve this goal: Adding β-reduction constraints for the source sentence leaves the original copy intact, and the target sentence still contains the ambiguity.

Under the simplifying assumption that all redexes are linear, we can show that it takes time $O(kn^3)$ to perform k steps of underspecified β-reduction on a constraint with n variables. This is feasible for large k as long as $n < 50$, which should be sufficient for most reasonable sentences. If there are non-linear redexes, the present algorithm can take exponential time because subterms are duplicated. The same problem is known in ordinary λ-calculus; an interesting question to pursue is whether the sharing techniques developed there (Lamping, 1990) carry over to the underspecification setting.

In Sec. 6, we only employ propagation rules; that is, we never disambiguate. This is conceptually very nice, but on more complex examples (e.g. in many cases with nonlinear redexes) disambiguation is still needed.

This raises both theoretical and practical issues. On the theoretical level, the questions of completeness (elimination of all redexes) and confluence still have to be resolved. To that end, we first have to find suitable notions of completeness and confluence in our setting. Also we would like to handle larger classes of examples without disambiguation. On the practical side, we intend to implement the procedure and disambiguate in a controlled fashion so we can reduce completely and still disambiguate as little as possible.

References

M. Bodirsky, K. Erk, A. Koller, and J. Niehren. 2001. Beta reduction constraints. In *Proc. 12th Rewriting Techniques and Applications*, Utrecht.

J. Bos. 1996. Predicate logic unplugged. In *Proceedings of the 10th Amsterdam Colloquium.*

R. Cooper. 1983. *Quantification and Syntactic Theory*. Reidel, Dordrecht.

M. Egg, A. Koller, and J. Niehren. 2001. The constraint language for lambda structures. *Journal of Logic, Language, and Information*. To appear.

K. Erk and J. Niehren. 2000. Parallelism constraints. In *Proc. 11th RTA*, LNCS 1833.

K. Erk, A. Koller, and J. Niehren. 2001. Processing underspecified semantic representations in the Constraint Language for Lambda Structures. *Journal of Language and Computation*. To appear.

A. Koller and J. Niehren. 2000. On underspecified processing of dynamic semantics. In *Proc. 18th COLING*, Saarbrücken.

A. Koller, J. Niehren, and K. Striegnitz. 2000. Relaxing underspecified semantic representations for reinterpretation. *Grammars*, 3(2/3). Special Issue on MOL'99. To appear.

J. Lamping. 1990. An algorithm for optimal lambda calculus reduction. In *ACM Symp. on Principles of Programming Languages*.

M. P. Marcus, D. Hindle, and M. M. Fleck. 1983. D-theory: Talking about talking about trees. In *Proc. 21st ACL.*

R. Montague. 1974. The proper treatment of quantification in ordinary English. In *Formal Philosophy. Selected Papers of Richard Montague*. Yale UP.

M. Pinkal. 1996. Radical underspecification. In *Proc. 10th Amsterdam Colloquium.*

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

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

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