Beta Reduction Constraints

Manuel Bodirsky Katrin Erk[⋆]
Alexander Koller[♯] [⋆⋆] Joachim Niehren[⋆⋆]

Programming Systems Lab [♯] Dept. of Computational Linguistics
Universität des Saarlandes, Saarbrücken, Germany
www.ps.uni-sb.de/~{bodirsky,erk,koller,niehren}

Abstract. The constraint language for lambda structures (CLLS) can model lambda terms that are known only partially. In this paper, we introduce beta reduction constraints to describe beta reduction steps between partially known lambda terms. We show that beta reduction constraints can be expressed in an extension of CLLS by group parallelism. We then extend a known semi-decision procedure for CLLS to also deal with group parallelism and thus with beta-reduction constraints.

1 Introduction

The constraint language for lambda structures (CLLS) [7, 6, 8] can model λ-terms that are known only partially. The idea is to see a λ-term as a λ-structure: a tree decorated with binding edges. One can then describe a λ-term partially as one would describe a standard tree structure. CLLS provides dominance [13, 2, 5], parallelism [9] and binding constraints for this purpose.

This paper shows how to lift β-reduction to partial descriptions of λ-terms in CLLS. We define *beta reduction constraints*, which allow a declarative description of the result of a single β-reduction step. At first, this description is very implicit; it is made explicit by *solving* the constraints. To this end, we show how beta reduction constraints can be expressed as *group parallelism constraints*. Then we adapt a known semi-decision procedure for CLLS to also deal with group parallelism and thus with beta-reduction constraints.

Beta-reduction constraints lay the foundation for *underspecified beta reduction*, which is needed in the application of CLLS to *semantic underspecification* of natural language [15, 17, 14]. Given a CLLS constraint describing many lambda terms, the aim is to compute a compact description of all corresponding beta normal forms efficiently. In particular, we want to avoid enumerating and individually beta-reducing the described lambda terms. (Enumerating is neither efficient, nor is its result compact.) A recent proposal towards *underspecified beta reduction* is described by the authors in a follow-up paper [4].

⋆ Supported by the DFG through the Graduiertenkolleg Kognition in Saarbrücken.
⋆⋆ Supported by the Collaborative Research Center (SFB) 378 of the DFG and the Procope project of the DAAD.

Solving beta reduction constraints is very much different from higher-order unification [10] in that CLLS constraints express α-equality rather than $\alpha\beta\eta$-equality. CLLS is closely linked to context unification [12, 16], and it can express sharing as in optimal lambda reduction [11] or calculi with explict substitutions [1] but can also describe several lambda terms at once.

Plan. We first recall the definition of CLLS restricted to dominance and λ-binding constraints (Sec. 2); then we go through two examples to give an idea of how one might lift β-reduction to partial descriptions (Sec. 3). We next define β-reduction constraints (Sec. 4). Then we define group parallelism constraints and show how they can express β-reduction constraints (Sec. 5). Finally, we present a sound and complete semi-decision procedure for CLLS with group parallelism (Sec. 6) and illustrate it with an example (Sec. 7).

2 CLLS with dominance and lambda binding constraints

We first introduce λ-structures and then a fragment of CLLS for their description. This fragment contains dominance and λ-binding constraints, but not parallelism and anaphoric binding constraints.

We assume a signature $\Sigma = \{f, g, \dots\}$ of function symbols, each equipped with an arity $\mathsf{ar}(f) \geq 0$. Symbols of arity 0 are constants, written as $a, b, \dots$

A tree θ is a ground term over Σ, e.g. $g(f(a, b))$. A *node* of a tree is identified with a *path* from the root to this node, expressed by a word over the naturals (excluding 0). ϵ is the empty path, and $\pi_1\pi_2$ the concatenation of π_1 and π_2. π is a *prefix* of a path π' if there is a (possibly empty) π'' s.t. $\pi\pi'' = \pi'$. The set of all nodes of a tree θ is defined as

$$D_\theta(f(\theta_1, \dots, \theta_n)) \;=\; \epsilon \cup \{i\pi \mid \pi \in D_\theta(\theta_i),\ 1 \leq i \leq n\}$$

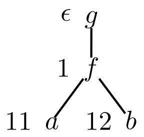

Fig. 1. Tree structure for $g(f(a, b))$.

A tree θ can be characterized uniquely by the set D_θ of its nodes and a labeling function $L_\theta : D_\theta \to \Sigma$.

Now we can consider λ-terms as pairs of a tree and a *binding function* that encodes variable binding. We assume that Σ contains the symbols var (arity 0, for variables), lam (arity 1, for abstraction), and @ (arity 2, for application), and quantifiers $\exists$ and $\forall$ (arity 1). The tree uses these symbols to reflect the structure of the λ-term and of first-order connectives. The binding function λ explicitly maps var-labeled nodes to binders. For example, Fig. 2 shows a representation of the term $\lambda x.f(x)$. Here $\lambda(12) = \epsilon$. Such a pair of a tree and a binding function is called a λ-structure.

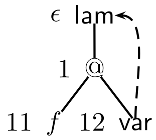

Fig. 2. The λ-structure of $\lambda x.f(x)$

Definition 1. *A* λ-structure τ *is a pair* (θ, λ) *of a tree* θ *and a total binding function* $\lambda : L_\theta^{-1}(\mathsf{var}) \to L_\theta^{-1}(\{\mathsf{lam}, \exists, \forall\})$ *such that* $\lambda(\pi)$ *is always a prefix of* π.

A λ-structure corresponds uniquely to a closed λ-term modulo α-renaming. We freely consider λ-structures as first-order structures with domain D_θ. As such, they define relations of labeling, binding, inverse binding, dominance, disjointness, and inequality of nodes. (Later we will add group parallelism and β-reduction relations.) The labeling relation $\pi{:}f(\pi_1, \ldots, \pi_n)$ holds in a λ-structure $\tau = (\theta, \lambda)$ if $L_\theta(\pi) = f$, $\mathsf{ar}(f) = n$ and $\pi_i = \pi i$ for all $1 \leq i \leq n$. *Dominance* $\triangleleft^*$ is the prefix relation between paths of D_θ; inequality $\neq$ is simply inequality of paths; *disjointness* $\pi \perp \pi'$ holds if neither $\pi \triangleleft^* \pi'$ nor $\pi' \triangleleft^* \pi$. We will also consider intersections, unions, and complements of these relations; for instance, *proper dominance* $\triangleleft^+$ is $\triangleleft^* \cap \neq$, and *equality* $=$ is $\triangleleft^* \cap \triangleright^*$. The relation $\lambda^{-1}(\pi_0){=}\{\pi_1, \ldots, \pi_n\}$ states that $\pi_1, \ldots, \pi_n$, and only those nodes, are λ-bound by π_0. Note that an element of a set can be mentioned multiply, i.e. $\{\pi, \pi\} = \{\pi\}$.

Now we can define *dominance and binding* constraints to talk about λ-structures as follows; X, Y, Z are variables that denote nodes.

$$\begin{array}{rll} \varphi, \psi & ::= XRY \mid X{:}f(X_1, \ldots, X_n) \mid \varphi \wedge \psi \mid \textbf{false} & (\mathsf{ar}(f) = n) \\ & \mid \ \ \lambda(X){=}Y \mid \lambda^{-1}(X_0){=}\{X_1, \ldots, X_n\} & \\ R, R' & ::= \triangleleft^* \mid \triangleright^* \mid \perp \mid \neq \mid R{\cup}R' \mid R{\cap}R' & \end{array}$$

A constraint φ is a conjunction of *literals* (for dominance, labeling, etc). Set operators in relation descriptors R [5] are mainly needed for processing purposes. As above we also use $\triangleleft^+, =$ to abbreviate set operators. The one literal that has not appeared in the literature before is the *inverse binding literal* $\lambda^{-1}(X){=}\{X_1, \ldots, X_n\}$, which matches the inverse binding relation.

We will also use first-order formulas Φ built over constraints. We write $\mathcal{V}(\Phi)$ for the set of variables occurring in Φ. Given a pair (τ, σ) of a λ-structure τ and a variable assignment $\sigma : \mathcal{G} \to D_\tau$, for some set $\mathcal{G} \supseteq \mathcal{V}(\varphi)$, we can associate a truth value to Φ in the usual Tarskian sense. We say that (τ, σ) *satisfies* Φ iff Φ evaluates to true under (τ, σ). In this case, we write $(\tau, \sigma) \models \Phi$ and say that (τ, σ) is a *solution* of Φ. Φ is *satisfiable* iff it has a solution. Entailment $\Phi \models \Phi'$ means that all solutions of Φ are also solutions of Φ', equivalence $\Phi \models\!\!\!\!\dashv \Phi'$ is mutual entailment.

We draw constraints as graphs with the nodes representing variables. E.g. Fig. 3 is the graph of $\lambda^{-1}(X){=}\{X_1, X_2\} \ \wedge \ X \triangleleft^* X_1 \ \wedge \ X \triangleleft^* X_2$. 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.

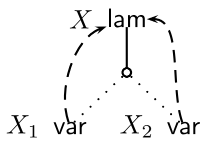

Fig. 3. A constraint graph

3 Examples

Before we begin with the formal investigation of beta reduction constraints, we first go through two examples which illustrate how beta-reduction can be lifted to descriptions of λ-structures in CLLS, and why the problem is nontrivial.

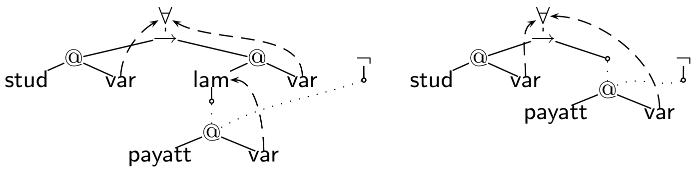

Fig. 4. Underspecified representations of 'Every student did not pay attention' before and after beta reduction.

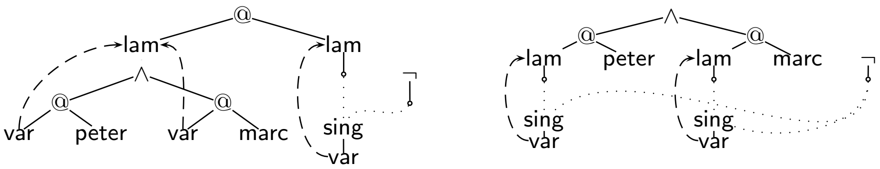

Fig. 5. Representation of 'Peter and Marc do not sing', wrong description of the reduct.

First, consider the left constraint in Fig. 4. The constraint contains just one redex, and it is easy to see how to obtain a description of the reduced formulas. We can essentially replace the bound variables with the argument description; the result is shown on the right-hand side of Fig. 4.

Incidentally, the left constraint in Fig. 4 is an underspecified description of the ambiguous sentence *Every student didn't pay attention*. Its two readings are given by the HOL formulas:

$$\forall x\ (\text{stud}\ x \to (\lambda y \neg(\text{payatt}\ y))\ x), \qquad \neg\forall x\ (\text{stud}\ x \to (\lambda y\ \text{payatt}\ y)\ x).$$

(These are the only models of the constraint that do not contain additional material not mentioned in the constraint. We ignore this aspect of "solution minimality" in this paper and always consider all solutions.)

The naive replacement approach, however, does not work in general. Fig. 5 shows an example (which describes the ambiguous sentence 'Peter and Marc do not sing.') This constraint also describes a β-redex, this time one where the binder binds two variables. Here it is no longer trivial to replace the bound variables by the argument description, as we do not know what belongs to the argument. There is no useful choice for the part of the constraint that should be duplicated; for example, if we decide not to duplicate the negation, we get the description on the right-hand side of Fig. 5, which lacks one solution. Describing the reduct using β-reduction constraints solves this problem; the description is correct even if it is not yet known which variables belong to the body and the argument of the redex.

4 Beta Reduction Constraints

In this section, we add the *β-reduction relation* to lambda structures and extend the constraint language by *β-reduction constraints*. The β-reduction relation on nodes of a lambda structure corresponds exactly to traditional beta reduction on lambda terms.

Stated in the unfolded notation for λ-terms we use to build the λ-structures (with application as an internal label @, etc.), β-reduction looks as follows:

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

We call the left-hand side the *reducing tree*, the right-hand side *the reduct* of the β-reduction. We call C the context, B the body, and A the argument of the reduction step.

Now an important notion throughout the paper are *tree segments*. Intuitively, a tree segment is a subtree which may itself be missing some subtrees (see Fig. 6). The context, body, and argument of a beta reduction step will all be tree segments.

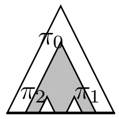

Fig. 6. The tree segment $\pi_0/\pi_1, \pi_2$.

Definition 2. *A* tree segment α *of a λ-structure* τ *is given by a tuple* $\pi_0/\pi_1, \ldots, \pi_n$ *of nodes in* D_τ*, such that* $\tau \models \pi_0 \triangleleft^* \pi_i$ *and* $\tau \models \pi_i(\perp\cup=)\pi_j$ *for* $1 \leq i < j \leq n$*. The node* $r(\alpha) = \pi_0$ *is called the* root*, and* $hs(\alpha) = \pi_1, \ldots, \pi_n$ *is the sequence of* holes *of* α*. If* $n = 0$ *we write* $\alpha = \pi_0/$*. The nodes* between *the root* $r(\alpha)$ *and the holes* $hs(\alpha)$ *are defined as*

$$\mathsf{b}(\alpha) =_{\mathsf{df}} \{\pi \in D_\tau \mid r(\alpha) \triangleleft^* \pi \wedge \bigwedge_{\pi' \in \{hs(\alpha)\}} \pi' \neg \triangleleft^+ \pi\}$$

To exempt the holes of the segment, we define $\mathsf{b}^-(\alpha) =_{\mathsf{df}} \mathsf{b}(\alpha) - \{hs(\alpha)\}$*.*

Definition 3. *A* correspondence function *between tree segments* α, β *in a lambda structure* τ *is a bijective mapping* $c : \mathsf{b}(\alpha) \rightarrow \mathsf{b}(\beta)$ *which satisfies for all nodes* $\pi_1, \ldots, \pi_n$ *of* τ*:*

1. *The roots correspond:* $c(r(\alpha)) = r(\beta)$
2. *The sequences of holes correspond:*

$$hs(\alpha) = \pi_1, \ldots, \pi_n \Leftrightarrow hs(\beta) = c(\pi_1), \ldots, c(\pi_n)$$

3. *Labels and children correspond within the proper segments. For* $\pi \in \mathsf{b}^-(\alpha)$ *and label* f*:*

$$\pi{:}f(\pi_1, \ldots, \pi_n) \Leftrightarrow c(\pi){:}f(c(\pi_1), \ldots c(\pi_n)).$$

We next define the β-reduction relation on λ-structures to be a relation between nodes in the *same* λ-structure. This allows us to see the β-reduction relation as a conservative extension of the existing λ-structures. The representations both of the reducing and reduced term are part of same big λ-structure—in Fig. 7, these are the subtrees rooted by $r(\gamma)$ and $r(\gamma')$ respectively.

A *redex* in a lambda structure is a sequence of segments (γ, β, α) that are connected by nodes π_0, π_1 with the following properties.

$$hs(\gamma) = \pi_0,\ \pi_0{:}@(\pi_1, r(\alpha)),\ \pi_1{:}\mathsf{lam}(r(\beta)), \text{ and } \lambda^{-1}(\pi_1) = \{hs(\beta)\}$$

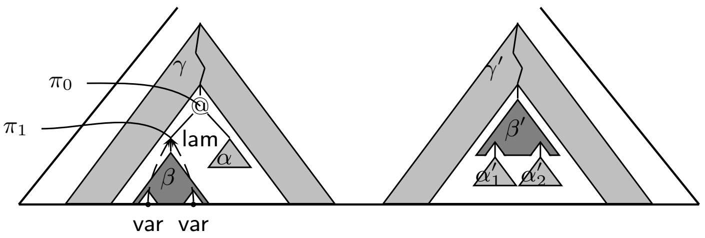

Fig. 7. The beta reduction relation for a binary redex.

We call a group $(\gamma', \beta', \alpha'_1, \ldots, \alpha'_n)$ reductlike, iff $hs(\gamma') = r(\beta')$ and $r(\alpha'_i)$ is the ith hole of β' for all $1 \leq i \leq n$.

Note that not every reductlike group is a potential reduct of a beta reduction, since we cannot enforce that there is no binder from the argument into the body (that would violate the freeness condition).

The lambda structure in Fig. 7 contains a redex (γ, β, α) and also its reduct $(\gamma', \beta', \alpha'_1, \alpha'_2)$. There, corresponding segments (γ to γ', β to β', α to both α'_1 and α'_2) have the same structure.

Definition 4 (Beta Reduction Relation). *Let τ be a λ-structure. Then*

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

holds in τ iff first, (γ, β, α) form a redex and $(\gamma', \beta', \alpha'_1, \ldots, \alpha'_n)$ are reductlike. *Second, there are correspondence functions c_γ between γ, γ', c_β between β, β' and c^i_α between α, α'_i (for $1 \leq i \leq n$), such that for each δ, δ' among these segment pairs with correspondence function c between them and for each $\pi \in \mathsf{b}^-(\delta)$, the following conditions hold:*

1. *for a* var*-labeled node bound in the same segment, the correspondent is bound by the c-corresponding binder node.*

$$\lambda(\pi) \in \mathsf{b}^-(\delta) \Rightarrow \lambda(c(\pi)) = c(\lambda(\pi))$$

2. *for a* var*-labeled node bound in the context γ, the correspondent is bound by the c_γ-corresponding binder node.*

$$\lambda(\pi) \in \mathsf{b}^-(\gamma) \Rightarrow \lambda(c(\pi)) = c_\gamma(\lambda(\pi))$$

3. *for a* var*-labeled node bound above the reducing tree, the corresponding node is bound at the same place:*

$$\lambda(\pi) \notin \mathsf{b}(r(\gamma)/) \Rightarrow \lambda(c(\pi)) = \lambda(\pi)$$

The β-reduction relation on λ-structures models β-reduction on λ-terms faithfully. This even holds for λ-terms with global variables, although λ-structures can only model closed λ-terms. Global variables correspond to var-labeled nodes that are bound in the surrounding tree, i.e. above the root node of the context of the redex. Rule 3 of Def. 4 thus ensures a proper treatment of global variables.

Capturing in β-reduction on λ-terms in classical λ-calculus is usually ´avoided by a freeness condition. For instance, one cannot simply β-reduce $(\lambda x.\lambda y.x)y$ without renaming the bound ocurrence of y beforehand. Otherwise, the global variable y in the argument gets captured by the binder λy. The following proposition states that the analoguous problem can never arise with the β-reduction relation on λ-structures.

Proposition 5 (No Capturing). *Global variables in the argument are never captured by a λ-binder in the body: with the notation of Def. 4, this means that no* var*-labeled node in* $\mathsf{b}(\alpha'_i)$ *is bound by a* lam*-labeled node in* $\mathsf{b}^-(\beta')$*.*

Proof. Assume there exists a node π' in $\mathsf{b}(\alpha'_i)$ such that $\lambda(\pi') \in \mathsf{b}^-(\beta')$. There must be a corresponding var-labeled node π with $c^i_\alpha(\pi) = \pi'$, which is bound either in α, in γ or outside the reducing tree. In the first case property (1) leads to a contradiction, in the second case property (2), and in the third case (3).

The β-reduction relation conservatively extends λ-structures. We extend our constraint syntax similarly, by *β-reduction literals*, which are interpreted by the β-reduction relation. Let a *segment term* A, B, C be given by the following abstract syntax:

$$A, B, C =_{\mathsf{df}} X_0/X_1, \ldots, X_n$$

Then β-reduction literals have the following form:

$$(C, B, A) \rightarrow^\beta (C', B', A'_1, \ldots, A'_n)$$

5 Group Parallelism

In this section, we extend dominance and binding constraints with *group parallelism constraints* (Def. 6), a generalization of the parallelism constraints found in CLLS [6, 9]. Then we show that CLLS with group parallelism can express β-reduction constraints (Thm. 7).

Group parallelism relates two *groups*, i.e. sequences of tree segments. It requires that corresponding entries in the two sequences must have the same tree structures, and binding in the two groups must be parallel. The following definition makes this precise; all conditions but the last are illustrated in Fig. 8.

Definition 6. *The* group parallelism relation $\sim$ *of a λ-structure τ is the greatest symmetric relation between groups of the same size such that*

$$(\alpha_1, \ldots, \alpha_n) \sim (\alpha'_1, \ldots, \alpha'_n)$$

implies there are correspondence functions $c_k : \mathsf{b}(\alpha_k) \rightarrow \mathsf{b}(\alpha'_k)$ *for all* $1 \leq k \leq n$ *that satisfy the following properties for all* $1 \leq i, j \leq n$ *and* $\pi \in \mathsf{b}^-(\alpha_i)$*:*

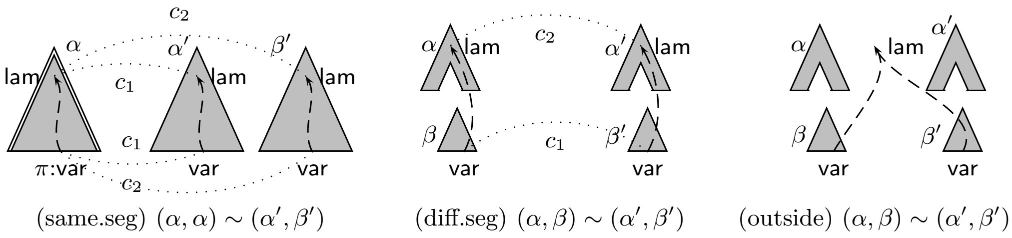

Fig. 8. Possible bindings in a group parallelism.

(same.seg) *for a* var*-labeled node bound in the same segment, the corresponding node is bound correspondingly:*

$$\lambda(\pi) \in \mathsf{b}^-(\alpha_i) \;\Rightarrow\; \lambda(c_i(\pi)) = c_i(\lambda(\pi))$$

(diff.seg) *for a* var*-labeled node bound outside α_i but inside α_j, the correspondent is bound at the corresponding place with respect to c_j:*

$$\lambda(\pi) \in \mathsf{b}^-(\alpha_j) \wedge \lambda(\pi) \notin \mathsf{b}^-(\alpha_i) \quad \Rightarrow\; \lambda(c_i(\pi)) = c_j(\lambda(\pi))$$

(outside) *corresponding* var*-labeled nodes with binders outside the group segments are bound by the same binder node:*

$$\lambda(\pi) \notin \cup_{k=1}^{n} \mathsf{b}^-(\alpha_k) \;\Rightarrow\; \lambda(c_i(\pi)) = \lambda(\pi)$$

(hang) *there are no hanging binders:*

$$\lambda^{-1}(\pi) \subseteq \cup_{k=1}^{n} \mathsf{b}^-(\alpha_k)$$

On the syntactic side, we extend CLLS by *group parallelism literals* that are interpreted by the group parallelism relation. Let $A_1, \ldots A_m, A'_1, \ldots, A'_m$ be segment terms, then group parallelism literals have the form

$$(A_1, \ldots, A_m) \quad \sim \quad (A'_1, \ldots, A'_m)$$

Group parallelism extends ordinary parallelism constraints [6, 9], which are simply the special case for groups of size one. This extension is proper; ordinary parallelism constraints cannot handle the case where a node is bound in a different segment of the group, as illustrated in the (diff.seg) part of Fig. 8. From the perspective of ordinary parallelism, the node is bound outside the parallel segment, and thus the (outside) condition applies, and the corresponding node must be bound by the *same* binder.

Another interesting observation in Fig. 8 is that the conditions (same.seg) and (diff.seg) must be mutually exclusive. If (diff.seg) was applicable in the leftmost case, it would enforce $\lambda(c_1(\pi)) = c_2(\lambda(\pi))$, which is clearly wrong.

Now we show how to encode beta reduction constraints in CLLS. First, we define the following formula to express that the segment term $A = X_0/X_1, \ldots, X_n$ indeed denotes a tree segment:

$$\mathsf{seg}(A) =_{\mathsf{df}} \wedge_{i=1}^{n} X_0 \lhd^* X_i \ \wedge\ \wedge_{1 \leq i < j \leq n} X_i (\bot \cup =) X_j$$

Using this, we can axiomatize a redex in CLLS. For segment terms $C = X'/X_0$, $B = X_3/X_4, \ldots, X_n$ and $A = X_2/$ we set:

$$\begin{aligned}\mathsf{redex}_{X_0,X_1}(C, B, A) \ =_{\mathsf{df}} & \ \mathsf{seg}(A) \ \wedge\ \mathsf{seg}(B) \ \wedge\ \mathsf{seg}(C) \\ & \wedge\ X_0{:}@(X_1, X_2) \wedge X_1{:}\mathsf{lam}(X_3) \\ & \wedge\ \lambda^{-1}(X_1) = \{X_4, \ldots, X_n\}\end{aligned}$$

Next, we define reduct-like groups. Let $C = X/X_0$, $B = X_0'/X_1', \ldots, X_n'$ and $A_i = X_i/$ for $1 \leq i \leq n$, then we define:

$$\begin{aligned}\mathsf{reductlike}(C, B, A_1, \ldots, A_n) \ =_{\mathsf{df}} & \ \mathsf{seg}(A_1) \wedge \cdots \wedge \mathsf{seg}(A_n) \wedge \mathsf{seg}(B) \wedge \mathsf{seg}(C) \\ & \wedge X_0{=}X_0' \wedge X_1{=}X_1' \wedge \cdots \wedge X_n{=}X_n'\end{aligned}$$

Theorem 7. *Beta reduction constraints can be expressed in CLLS with group parallelism via the following equivalence:*

$$\begin{aligned}(C, B, A) \rightarrow^{\beta} (C', B', A_1', \ldots, A_n') \quad \models\!\mid \quad & \exists\, X_0, X_1 : \mathsf{redex}_{X_0,X_1}(C, B, A) \\ & \wedge\ (C, B, A, \ldots, A) \sim (C', B', A', \ldots, A') \\ & \wedge\ \mathsf{reductlike}(C', B', A_1', \ldots, A_n')\end{aligned}$$

Proof. We will check the two-side entailment separately, first from right to left. Let σ be a variable assignment into some λ-structure that solves the right hand side. Properties (same.seg), (diff.seg), and (outside) of group parallelism (Def. 6) then subsume the corresponding properties of β-reduction (Def. 4).

For the other direction, let (τ, σ') solve the beta-reduction literal on the left hand side. According to (Def. 4) there exists a redex (γ, β, α) in τ with nodes π_0, π_1 as in Sec. 4. Let σ be the variable assignment $\sigma'[\pi_0/X_0, \pi_1/X_1, \alpha/A, \beta/B, \gamma/C]$. It remains to check that (τ, σ) solves the group parallelism literal on the right hand side.

We consider the symmetric relation $\approx$ which relates the group $(\sigma(C), \sigma(B), \sigma(A), \ldots, \sigma(A))$ to $(\sigma(C'), \sigma(B'), \sigma(A_1'), \ldots, \sigma(A_n'))$ and conversely. We show that $\approx$ satisfies all conditions in the definition of group parallelism (Def. 6), which means that $\approx$ is subsumed by the group parallelism relation $\sim$.

First of all, both above groups satisfy condition (hang). This is clear for the group $(\sigma(C'), \sigma(B'), \sigma(A_1'), \ldots, \sigma(A_n'))$, which covers the complete subtree below $r(\sigma(C'))$. A similar argument applies to $(\sigma(C), \sigma(B), \sigma(A), \ldots, \sigma(A))$, which covers the whole tree below $r(\sigma(C))$ exept the @-labeled node π_0, the lam-labeled node π_1 and the var-labeled nodes $hs(\sigma(B))$. But these var-labeled nodes are bound by π_1.

By Def. 4 there exist correspondence functions c_γ between segments $\sigma(C)$, $\sigma(C')$, c_β between $\sigma(B)$, $\sigma(B')$ and c_α^i between $\sigma(A), \sigma(A_i')$ for $1 \leq i \leq n$. Since $\approx$ is symmetric, we have to check properties (same.seg), (diff.seg), and (outside) of group parallelism (Def. 6) for the correspondence functions and their inverse functions.

(D.clash.ineq) $X{=}Y \wedge X{\neq}Y \to \textbf{false}$

(D.dom.trans) $X{\triangleleft^*}Y \wedge Y{\triangleleft^*}Z \to X{\triangleleft^*}Z$

(D.lab.ineq) $X{:}f(\ldots) \wedge Y{:}g(\ldots) \to X{\neq}Y \quad$ where $f \neq g$

(D.lab.dom) $X{:}f(\ldots,Y,\ldots) \to X{\triangleleft^+}Y$

(D.distr.notDisj) $X{\triangleleft^*}Z \wedge Y{\triangleleft^*}Z \to X{\triangleleft^*}Y \vee Y{\triangleleft^*}X$

(D.distr.child) $X{\triangleleft^*}Y \wedge X{:}f(X_1,\ldots,X_m) \to Y{=}X \vee \bigvee_{i=1}^{m} X_i{\triangleleft^*}Y$

Fig. 9. Saturation rules for dominance constraints

We only show the particularly interesting property (diff.seg) for a correspondence function $(c^i_\alpha)^{-1}$ with $1 \leq i \leq n$. Let π' be a var-labeled node in $\mathsf{b}^-(\sigma(A'_i))$, and $\lambda(\pi') \notin \mathsf{b}^-(\sigma(A'_i))$. There are three cases: $\lambda(\pi') \in \mathsf{b}^-(\sigma(C'))$, or $\lambda(\pi') \in \mathsf{b}^-(\sigma(B'))$, or $\lambda(\pi') \in \mathsf{b}^-(\sigma(A'_j))$ for some $1 \leq j \leq n$. The second case is impossible by Proposition 5. The third case is impossible as the holes of the segment $\sigma(B')$ are disjoint or equal (Def. 2). We can thus concentrate on the first case. Let π be the corresponding node of π'. i.e. $c^i_\alpha(\pi) = \pi'$. The node π has to be var-labeled by Def 3. Properties (same.seg) and (outside) of Def. 4 yield $\lambda(\pi) \in \sigma(C)$ (some computation is needed here). Thus, Property 2 of Def. 4 can be applied. It implies $\lambda(\pi') = c_\gamma(\lambda(\pi))$, i.e. $c_\gamma^{-1}(\lambda(\pi')) = \lambda(c_\gamma^{-1}(\pi'))$ as required.

6 Solving group parallelism constraints

We now turn to a sound and complete semi-decision procedure for CLLS with group parallelism, which thus solves β-reduction constraints. To keep the presentation readable, we focus on the most relevant rules. The full procedure is given in [3]. An illustrative example follows in Section 7.

The procedure is obtained by extending an existing semi-decision procedure for CLLS [8] that is based on *saturation*. A constraint is freely identified with the *set* of its literals. Starting with a set of literals, more literals are added according to some *saturation rules*. Our saturation rules are implications of the form $\varphi_0 \to \vee_{i=1}^n \varphi_i$ for some $n \geq 1$. To write down rules more compactly, we will also use arbitrary positive existential formulas on the left hand side. These can be eliminated in a preprocessing step: $\exists$-quantified variables can be replaced by arbitrary variables, and disjunction is eliminated by explosion into several rules.

A saturation rule of the above form is applicable to a constraint φ if φ_0 is contained in φ, but none of the φ_i is. A rule $\varphi \to \Phi$ is *sound* if $\varphi \models \Phi$. Apart from that, we have saturation rules of the form $\varphi_0 \to \exists X \varphi_1$, which introduce fresh variables. Such a rule is applicable to φ if φ_0 is in φ, but φ_1, modulo renaming of X, is not. Given a set S of saturation rules, we call a constraint *saturated* (under S) if no further rule of S applies to it. We say that a constraint is in *S-solved form* if it is saturated under S and clash-free (i.e. it does not contain **false**).

Fig. 9 contains an (incomplete) set of saturation rules for dealing with dominance constraints (the constraints of Sec. 2 without binding). A more

(P.symm) $\overline{A} \sim \overline{B} \rightarrow \overline{B} \sim \overline{A}$

(P.init) $\overline{A} \sim \overline{B} \rightarrow \mathsf{seg}(A_i) \wedge \mathrm{co}(A_i, B_i)(X_i^j){=}Y_i^j$ where $1 \le i \le n$, $A_i = X_i^0/X_i^1, \ldots, X_i^{m_i}$, $B_i = X_i^0/Y_i^1, \ldots, Y_i^{m_i}$, and $0 \le j \le m_i$

(P.new) $\overline{A} \sim \overline{B} \wedge U \in \mathsf{b}(A_i) \rightarrow \exists V\ \mathrm{co}(A_i, B_i)(U){=}V$ where V fresh, $1 \le i \le n$

(P.copy.lab) $\bigwedge_{i=0}^{m} \mathrm{co}(A, B)(X_i){=}Y_i \wedge X_0{:}f(X_1, \ldots, X_m) \wedge X_0{\in}\mathsf{b}^-(A) \rightarrow Y_0{:}f(Y_1, \ldots, Y_m)$

(P.copy.dom) $U_1\, R\, U_2 \wedge \bigwedge_{i=1}^{2} \mathrm{co}(A, B)(U_i){=}V_i \rightarrow V_1\, R\, V_2$

(P.distr.eq) $\varphi \rightarrow X{=}Y \vee X{\neq}Y$ for $X, Y \in \mathcal{V}(\varphi)$

Fig. 10. Saturation rules, where $\overline{A} = A_1, \ldots, A_n$ and $\overline{B} = B_1, \ldots, B_n$

complete collection including a treatment of set operators can be found in [5, 8, 3]. To deal with parallelism, we first introduce some formulas that describe membership in (proper) segments and groups.

$$\begin{aligned}
X{\in}\mathsf{b}(A) &=_{\mathsf{df}} X_0{\lhd^*}X \wedge \textstyle\bigwedge_{i=1}^{n} X(\lhd^*{\cup}\bot)X_i \\
X{\in}\mathsf{b}^-(A) &=_{\mathsf{df}} X{\in}\mathsf{b}(A) \wedge \textstyle\bigwedge_{i=1}^{n} X{\neq}X_i \\
X{\notin}\mathsf{b}^-(A) &=_{\mathsf{df}} X(\lhd^+{\cup}\bot)X_0 \vee \textstyle\bigvee_{i=1}^{n} X_i{\lhd^*}X \\
X{\in}\mathsf{b}(A_1, \ldots, A_m) &=_{\mathsf{df}} \textstyle\bigvee_{i=1}^{m} X{\in}\mathsf{b}(A_i) \\
X{\in}\mathsf{b}^-(A_1, \ldots, A_m) &=_{\mathsf{df}} \textstyle\bigvee_{i=1}^{m} X{\in}\mathsf{b}^-(A_i)
\end{aligned}$$

Note that the terms $\mathsf{b}(A)$, $\mathsf{b}^-(A)$, $\mathsf{b}(A_1, \ldots, A_m)$ are not given any formal meaning, even though it would be correct to interpret them as the corresponding sets of nodes.

We also want to be able to speak about correspondence functions. So we extend our constraint language by auxiliary correspondence literals

$$\varphi \quad ::= \quad \ldots \mid \mathrm{co}(A, B)(X){=}Y$$

where A and B are segment terms for segments with the same number of holes. Such a literal states that A and B are parallel within some group parallelism, that $X{\in}\mathsf{b}(A)$ and $Y{\in}\mathsf{b}(B)$, and that X corresponds to Y with respect to the correspondence function for A and B. We introduce two more formulas. Let $\overline{A} = (A_1, \ldots, A_n)$, $\overline{B} = (B_1, \ldots, B_n)$, and $1 \le k \le n$.

$$\begin{aligned}
\mathrm{co}^-(A, B)(X){=}Y &=_{\mathsf{df}} \mathrm{co}(A, B)(X){=}Y \wedge X{\in}\mathsf{b}^-(A) \\
\mathrm{co}_k^-(\overline{A}, \overline{B})(X){=}Y &=_{\mathsf{df}} \overline{A}{\sim}\overline{B} \wedge \mathrm{co}^-(A_k, B_k)(X){=}Y
\end{aligned}$$

The second lets us talk about correspondence functions for a group parallelism, picking out the k-th correspondence function. In that respect, $\mathrm{co}_k^-(\overline{A}, \overline{B})$ matches the c_k of Def. 6 (except that $\mathrm{co}_k^-(\overline{A}, \overline{B})(X){=}Y$ additionally demands $X{\in}\mathsf{b}^-(A_k)$ for convenience).

The main rules for handling parallelism are given in Fig. 10. A complete set can be found in [9, 8, 3]. The rules (P.init) and (P.new) introduce correspondence literals; between them, they state that each node in a parallel segment

(L.same.seg) $\lambda(U_1)=U_2 \wedge \bigwedge_{i=1}^2 \mathrm{co}_k^-(\overline{A},\overline{B})(U_i)=V_i \rightarrow \lambda(V_1)=V_2$

(L.diff.seg) $\lambda(U_1)=U_2 \wedge \bigwedge_{i=1}^2 \mathrm{co}_{k_i}^-(\overline{A},\overline{B})(U_i)=V_i \wedge U_2\notin \mathsf{b}^-(A_{k_1}) \rightarrow \lambda(V_1)=V_2$

(L.outside) $\lambda(U)=Y \wedge \mathrm{co}_k^-(\overline{A},\overline{B})(U)=V \wedge Y\notin \mathsf{b}^-(\overline{A}) \rightarrow \lambda(V)=Y$

(L.hang) $\lambda(U_1)=U_2 \wedge \overline{A}{\sim}\overline{B} \wedge U_2\in \mathsf{b}^-(\overline{A}) \rightarrow U_1\in \mathsf{b}^-(\overline{A})$

(L.distr.1) $\lambda(U_1)=U_2 \wedge \overline{A}{\sim}\overline{B} \wedge U_1\in \mathsf{b}^-(\overline{A}) \rightarrow \mathsf{distr}_{\overline{A}}(U_2)$

(L.distr.2) $\lambda(U_1)=U_2 \wedge \overline{A}{\sim}\overline{B} \wedge U_2 \in \mathsf{b}^-(\overline{A}) \rightarrow \mathsf{distr}_{\overline{A}}(U_1)$

(L.equal) $\lambda(X_1)=X_2 \wedge \bigwedge_{i=1}^2 X_i=Y_i \rightarrow \lambda(Y_1)=Y_2$

(L.inverse) $\lambda^{-1}(X)=S_1 \wedge \mathrm{co}_k^-(\overline{A},\overline{B})(X)=Y \wedge \mathrm{co}^-(\overline{A},\overline{B})(S_1)=S_2 \cup S_3 \wedge$
$\bigwedge_{V\in S_2} \lambda(V)=Y \wedge \bigwedge_{V\in S_3} \lambda(V)\neq Y \rightarrow \lambda^{-1}(Y)=S_2$

Fig. 11. Lambda binding rules for group parallelism

needs to have a correspondent. (P.init) states that in a correspondence function, root corresponds to root, and hole to hole, while (P.new) is responsible for all other nodes. (P.copy.dom) and (P.copy.lab) between them ascertain the structural isomorphism that Def. 3 demands for a correspondence function.

Fig. 11 shows saturation rules for the interaction of group parallelism and lambda binding. The first four rule schemata directly express the conditions of Def. 6. The rules (L.distr.gr.1) and (L.distr.gr.2) decide, loosely speaking, whether variables occurring in a binding literal belong to some segment of a group or not. This is necessary because we need to know which of the schemata (L.same.seg), (L.diff.seg), (L.outside) and (L.hang) is applicable. This is expressed by using the following formula, where $\overline{A} = (A_1, \ldots, A_n)$:

$$\mathsf{distr}_{\overline{A}}(U) =_{\mathsf{df}} \bigwedge_{i=1}^{n} (U\in \mathsf{b}^-(A_i) \vee U\notin \mathsf{b}^-(A_i).$$

Finally, (L.inverse) deals with the copying of λ^{-1} literals. This is necessary if we want to perform a second beta reduction step, where we need the λ^{-1} information again. The schema uses two more formulas. The first one is simple:

$$\lambda(X)\neq Y =_{\mathsf{df}} \exists Z(\lambda(X)=Z \wedge Z\neq Y)$$

The second formula collects, for a finite set S_1 of variables, all correspondents with respect to $\overline{A} \sim \overline{B}$. Let S_1, S_2 stand for finite sets of variables, and let $\overline{A} = A_1 \ldots, A_n$.

$$\begin{aligned}\mathrm{co}^-(\overline{A},\overline{B})(S_1)=S_2 =_{\mathsf{df}} & \textstyle\bigwedge_{i=1}^n \bigwedge_{X\in S_1} (X\notin \mathsf{b}^-(A_i) \vee \bigvee_{Y\in S_2} \mathrm{co}_i^-(\overline{A},\overline{B})(X)=Y) \\ & \wedge \ \textstyle\bigwedge_{Y\in S_2} \bigvee_{X\in S_1} \bigvee_{i=1}^n \mathrm{co}_i^-(\overline{A},\overline{B})(X)=Y\end{aligned}$$

So (L.inverse) collects all correspondents of all variables bound by X; for each of these correspondents it must be known whether it is bound by Y or definitely bound by something else. Then we can determine $\lambda^{-1}(Y)$. The soundness of this rule is not obvious: is it really sufficient to look among the correspondents of $\lambda^{-1}(X)$ to compute $\lambda^{-1}(Y)$? The following proposition shows that it is.

Proposition 8 (Inverse lambda binding). *Suppose* $(\alpha_1, \ldots, \alpha_n) \sim (\alpha'_1, \ldots, \alpha'_n)$ *holds with correspondence functions* $c_1, \ldots, c_n$. *Then for all* $1 \leq k \leq n$ *and all* $\pi \in \mathsf{b}^-(\alpha_k)$,

$$\lambda^{-1}(c_k(\pi)) \subseteq \bigcup_{i=1}^{n} \{c_i(\pi') \mid \pi' \in \lambda^{-1}(\pi) \cap \mathsf{b}^-(\alpha_i)\}$$

Proof. Let $\omega \in \lambda^{-1}(c_k(\pi))$. The "no hanging binders" condition (hang) of Def. 6 is critical here: it enforces $\omega \in \bigcup_{i=1}^{n} \mathsf{b}^-(\alpha'_i)$. If $\omega \in \mathsf{b}^-(\alpha'_k)$, then there exists some $\pi' \in \mathsf{b}^-(\alpha_k)$ with $c_k(\pi') = \omega$. π' is var-labeled by Def. 3 and has a binder since λ is total. So we must have $\lambda(c_k(\pi')) = c_k(\lambda(\pi'))$ by condition (same.seg) of Def. 6. Now $\lambda(c_k(\pi')) = c_k(\pi)$ and c_k is a bijection, so $\pi' \in \lambda^{-1}(\pi)$. If, on the other hand, $\omega \notin \mathsf{b}^-(\alpha'_k)$ but $\omega \in \mathsf{b}^-(\alpha'_j)$, there is again a π' with $c_j(\pi') = \omega$, and $\lambda(c_j(\pi')) = c_k(\lambda(\pi'))$ by condition (diff.seg), so again $\pi' \in \lambda^{-1}(\pi)$.

The rules we have presented are part of a sound and complete semi-decision procedure for group parallelism constraints given in [3]. The omitted rules state additional properties of dominance constraints, ensure that correspondence functions are indeed bijective functions, and regulate the interaction between different correspondence functions.

Theorem 9. *There exists a saturation procedure GP which encompasses all instances of the rule schemata in Fig. 9, 10, and 11, such that each rule of GP is sound, and each GP-solved form of a constraint* φ *is satisfiable* **(soundness)**, *and for every solution* (τ, σ) *of* φ, *GP computes a GP-solved form of* φ *of which* (τ, σ) *is a solution* **(completeness)**.

Proving that *GP*-solved forms are satisfiable can be done by constructing a model and variable assignment explicitly. One then has to check that all literals are indeed satisfied, which requires a tedious case distinction. Proving completeness is nontrivial as well, but can be done along the lines of [9]. The proof is largely independent of the particularities of the rule system we employ.

7 The procedure in action

We illustrate the procedure of the previous section by solving the constraint in Fig. 12. It contains a non-linear lambda redex at (C, B, A) (similarly to Fig. 5) and a lambda binder at Y_1 which can either belong to the context C or argument A. The group parallelism constraint $(C, B, A, A){\sim}(C', B', A', A'')$ describes a beta-reduction step for the redex (C, B, A).

A record of the solving steps is given in Fig. 13 and 14. We only comment on the main steps. In step (4), we have $Y_1 \triangleleft^* Z \wedge X_0 \triangleleft^* Z$, and as trees do not branch upwards, one of Y_1, X_0 must dominate the other. This step effectively guesses whether Y_1, Y_2 are in C or in A. With choice (5c), we make two copies of Y_1 and Y_2 each. This is because A is parallel both to A' and A'': because X_ℓ binds two

$(C, B, A, A) \sim (C', B', A', A'')$
with $C = X/X_0$,
$C' = X'/X'_0$,
$B = X_t/X_1, X_2$,
$B' = X'_0/X'_1, X'_2$,
$A = X_a/$,
$A' = X'_1/$, and $A'' = X'_2/$.
$\lambda^{-1}(X_\ell) = \{X_1, X_2\}$, $\lambda^{-1}(Y_1) = \{Z\}$.

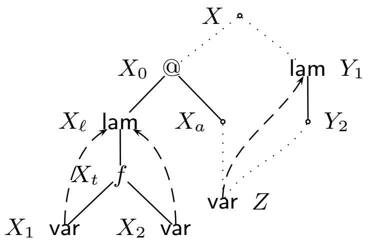

Fig. 12. A group parallelism constraint encoding a non-linear beta reduction step

variables, the argument is copied twice. On the other hand, with (7b) Y_1 and Y_2 are only copied once: they belong to the context C, which is parallel only to C'.

In Fig. 14, we continue case (5c) of Fig. 13, applying the lambda binding rules. All steps from (22) on prepare the determination of $\lambda^{-1}(Y_1')$ and $\lambda^{-1}(Y_1'')$ in (25) and (26). We know $\lambda^{-1}(Y_1)=\{Z\}$. Steps (22) and (23) determine $S_2 \cup S_3$ to be $\{Y_1', Y_1''\}$ for both (25) and (26). After (24) we know $\lambda(Z')\neq Y_1''$ and $\lambda(Z'')\neq Y_1'$, so we have all we need to infer the correct λ^{-1} information in the last steps.

8 Conclusion and future work

We have introduced *beta reduction constraints* and have presented a semi-decision procedure for processing them, in three steps: First, we have extended CLLS by group parallelism constraints. Second, we have expressed β-reduction constraints in this extension of CLLS. Third, we have lifted a known semi-decision procedure for CLLS to also deal with group parallelism constraints. It is an open question to what extent beta reduction constraints can conversely express parallelism.

This gives us a framework for investigating the more general problem of *underspecified beta reduction* [4]: How can we string together several reduction steps, as described by beta reduction constraints, until we arrive at descriptions of normal forms? In this broader setting, we can investigate properties such as confluence and termination on the underspecified level. Another problem, motivated by the application, is to modify the saturation procedure to perform as few case distinctions as possible during underspecified beta reduction. Finally, it will be interesting to find an efficient implementation of this operation, possibly employing concepts such as sharing and constraint programming.

References

1. M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.
2. R. Backofen, J. Rogers, and K. Vijay-Shanker. A first-order axiomatization of the theory of finite trees. *J. Logic, Language, and Information*, 4(1):5–39, 1995.

(1)	$Y_1 \neq X_0$	(D.lab.ineq)
(2)	$Y_1 \triangleleft^+ Y_2, X_0 \triangleleft^+ X_a$	(D.lab.dom)
(3)	$Y_1 \triangleleft^* Z, X_0 \triangleleft^* Z$	(D.dom.trans)
(4)	$X_0 \triangleleft^* Y_1 \ \vee\ Y_1 \triangleleft^* X_0$	(D.distr.notDisj)

(4a) $X_0 \triangleleft^* Y_1$:	(4b) $Y_1 \triangleleft^* X_0$:
(5) $X_0 = Y_1 \vee X_\ell \triangleleft^* Y_1 \vee X_a \triangleleft^* Y_1$ (D.distr.child)	(7) $Y_1 = X_0 \ \vee\ Y_2 \triangleleft^* X_0$ (D.distr.child)

(5a)	$X_0 = Y_1$:	(5b) $X_\ell \triangleleft^* Y_1$:	(7a)	$Y_1 = X_0$:	
...	both lead to **false**		(8)	**false**	(D.clash.ineq)
(5c)	$X_a \triangleleft^* Y_1$:		(7b)	$Y_2 \triangleleft^* X_0$:	
(9)	$\mathrm{co}(A, A')(X_a) = X_1'$	(P.init)			
(10)	$\mathrm{co}(A, A')(Y_1) = Y_1'$,	(P.new)			
	$\mathrm{co}(A, A')(Y_2) = Y_2'$,				
	$\mathrm{co}(A, A')(Z) = Z'$		(17)	$\mathrm{co}(C, C')(X) = X'$,	(P.init)
(11)	$X_1' \triangleleft^* Y_1', Y_2' \triangleleft^* Z'$	(P.copy.dom)		$\mathrm{co}(C, C')(X_0) = X_0'$	
(12)	Y_1':$\mathsf{lam}(Y_2')$	(P.copy.lab)	(18)	$\mathrm{co}(C, C')(Y_1) = Y_1'$,	(P.new)
(13)	$\mathrm{co}(A, A'')(X_a) = X_2'$	(P.init)		$\mathrm{co}(C, C')(Y_2) = Y_2'$	
(14)	$\mathrm{co}(A, A'')(Y_1) = Y_1''$,	(P.new)	(19)	$X' \triangleleft^* Y_1', Y_2' \triangleleft^* X_0'$	(P.copy.dom)
	$\mathrm{co}(A, A'')(Y_2) = Y_2''$,		(20)	Y_1':$\mathsf{lam}(Y_2')$	(P.copy.lab)
	$\mathrm{co}(A, A'')(Z) = Z''$				
(15)	$X_2' \triangleleft^* Y_1'', Y_2'' \triangleleft^* Z''$	(P.copy.dom)			
(16)	Y_1'':$\mathsf{lam}(Y_2'')$	(P.copy.lab)			

Fig. 13. Solving the group parallelism constraint in Fig. 12

3. M. Bodirsky, K. Erk, A. Koller, and J. Niehren. Beta reduction constraints, 2001. Full version. `http://www.ps.uni-sb.de/Papers/abstracts/beta.html`.
4. M. Bodirsky, K. Erk, A. Koller, and J. Niehren. Underspecified beta reduction, 2001. Submitted. `http://www.ps.uni-sb.de/Papers/abstracts/usp-beta.html`.
5. D. Duchier and J. Niehren. Dominance constraints with set operators. In *First International Conference on Computational Logic (CL2000)*, LNCS, July 2000.
6. M. Egg, A. Koller, and J. Niehren. The Constraint Language for Lambda Structures. *Journal of Logic, Language, and Information*, 2001. To appear.
7. M. Egg, J. Niehren, P. Ruhrberg, and F. Xu. Constraints over lambda-structures in semantic underspecification. In *In Proceedings COLING/ACL'98*, 1998.
8. K. Erk, A. Koller, and J. Niehren. Processing underspecified semantic descriptions in CLLS. *Journal of Language & Computation*, 2001. To appear.
9. K. Erk and J. Niehren. Parallelism constraints. In *Int. Conference on Rewriting Techniques and Applications*, volume 1833 of *LNCS*, pages 110–126, 2000.
10. G. P. Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–57, 1975.
11. J. Lamping. An algorithm for optimal lambda calculus reduction. In *ACM Symposium on Principles of Programming Languages*, pages 16–30, 1990.
12. J. Lévy. Linear second order unification. In *7th Int. Conference on Rewriting Techniques and Applications*, volume 1103 of *LNCS*, pages 332–346, 1996.
13. M. P. Marcus, D. Hindle, and M. M. Fleck. D-theory: Talking about talking about trees. In *Proceedings of the 21st ACL*, pages 129–136, 1983.
14. M. Pinkal. Radical underspecification. In *Proc. 10th Amsterdam Colloquium*, 1996.
15. U. Reyle. Dealing with ambiguities by underspecification: construction, representation, and deduction. *Journal of Semantics*, 10:123–179, 1993.

	Continuing (5c)		
(21)	$\lambda(Z') = Y_1'$, $\lambda(Z'') = Y_1''$ (L.same.seg)		
(22)	$Z \notin \mathsf{b}^-(B) \ \vee \ Z \in \mathsf{b}^-(B)$ (L.distr.2)		
(22a)	$Z \notin \mathsf{b}^-(B)$	(22b)	$Z \notin \mathsf{b}^-(B)$
(23)	$Z \notin \mathsf{b}^-(C) \ \vee \ Z \in \mathsf{b}^-(C)$ (L.distr.2)	...	**false**
(23a)	$Z \notin \mathsf{b}^-(C)$	(23b)	$Z \in \mathsf{b}^-(C)$
(24)	$Y_1' {\neq} Y_1'' \ \vee \ Y_1' {=} Y_1''$ (P.distr.eq)	...	**false**
(24a)	$Y_1' {\neq} Y_1''$	(24b)	$Y_1' {=} Y_1''$
(25)	$\lambda^{-1}(Z'') \neq Y_1'$	...	**false**
(26)	$\lambda^{-1}(Y_1'') = \{Z'\}$ (L.inverse)		
(27)	$\lambda^{-1}(Y_1'') = \{Z''\}$ (L.inverse)		

Fig. 14. Inverse Binding in case (5c)

16. M. Schmidt-Schauß and K. Schulz. Solvability of context equations with two context variables is decidable. In *Int. Conf. on Automated Deduction*, LNCS, 1999.
17. K. van Deemter and S. Peters. *Semantic Ambiguity and Underspecification*. CSLI Press, Stanford, 1996.

A The procedure *GP*

In this section, we list all rule schemata of the saturation procedure *GP* for group parallelism constraints.

We have said some words on saturation earlier, but have left things rather informal. We remedy that now. The *saturation rules* we work with are implications of the following form:

$$\varphi_0 \to \vee_{i=1}^{n} \exists V_i \varphi_i$$

where $n \geq 1$ and $\mathcal{V}(\varphi_i) - \mathcal{V}(\varphi_0) \subseteq V_i$ for all $1 \leq i \leq n$. A rule is called a *propagation rule* if $n = 1$ and a *distribution rule* otherwise.

A saturation algorithm S is a set of saturation algorithms. A *saturation step* $\to_S$ consists of one application of a rule in S:

$$\frac{\varphi' \subseteq \varphi \qquad \rho \in S}{\varphi \to_S \varphi \wedge \exists V_i \varphi_i} \text{ if } C_\rho(\varphi) \text{ where } \rho \text{ is } \varphi' \to \vee_{i=1}^{n} \exists V_i \varphi_i$$

Let $\mathcal{V}$ be the set of all node variables. Given a set V of variables and a constraint φ, we call a constraint $\zeta\varphi$ a *V-variant of* φ if $\zeta : V \to \mathcal{V}$ is some substitution of the variables in V. We call this variant *fresh* if $\zeta(V)$ is disjoint from $\mathcal{V}(\varphi)$. Then we let $C_{\varphi' \to \vee_{i=1}^{n} V_i \varphi_i}(\varphi)$ be true iff for all $1 \leq i \leq n$ and for all V_i-variants φ_i' of φ_i, we have $\varphi_i' \not\subseteq \varphi$. That is, we only apply a rule when it can add something new.

Let S be a set of saturation rules. As mentioned before, we call a constraint *saturated* (under S) if no further rule of S applies to it. We say that a constraint is in *S-solved form* if it is saturated under S and clash-free (i.e. it does not contain **false**).

The rules we present here only make use of part of the language of CLLS with group parallelism. We use $\triangleleft^*$, $\perp$ and $\neq$, but we do not process literals of the form $X(R{\cup}R')Y$ or $X(R{\cap}R')Y$. Instead, we interpret $\cup$ as disjunction and $\cap$ as conjunction. So XRY becomes another constraint abbreviation.

The saturation procedure *GP* that we present consists of four modules: The rule system D deals with constraints without parallelism and binding; rule system B states properties of λ binding, system P is concerned with parallelism, and rule system L handles the interaction of parallelism and binding.

We first list the rule schemata of system D, which deals with *Dominance Constraints*, the sublanguage of group parallelism constraints which only comprises dominance, labeling, disjointness, and inequality literals.

Solving dominance constraints: rule system D

(D.clash.ineq) $X{=}Y \wedge X{\neq}Y \rightarrow$ **false**

(D.clash.disj) $X{\perp}X \rightarrow$ **false**

(D.dom.refl) $\varphi \rightarrow X{\triangleleft^*}X$ where $X \in \mathcal{V}(\varphi)$

(D.dom.trans) $X{\triangleleft^*}Y \wedge Y{\triangleleft^*}Z \rightarrow X{\triangleleft^*}Z$

(D.eq.decom) $X{:}f(\overline{X}) \wedge Y{:}f(\overline{Y}) \wedge X{=}Y \rightarrow \wedge_{i=1}^{n} X_i{=}Y_i$

(D.lab.ineq) $X{:}f(\ldots) \wedge Y{:}g(\ldots) \rightarrow X{\neq}Y$ where $f \neq g$

(D.lab.disj) $X{:}f(\ldots X_i, \ldots, X_j, \ldots) \rightarrow X_i{\perp}X_j$ for $1 \leq i < j \leq n$

(D.prop.disj) $X{\perp}Y \wedge X{\triangleleft^*}X' \wedge Y{\triangleleft^*}Y' \rightarrow Y'{\perp}X'$

(D.lab.dom) $X{:}f(\ldots, Y, \ldots) \rightarrow X{\triangleleft^+}Y$

(D.distr.notDisj) $X{\triangleleft^*}Z \wedge Y{\triangleleft^*}Z \rightarrow X{\triangleleft^*}Y \vee Y{\triangleleft^*}X$

(D.distr.child) $X{\triangleleft^*}Y \wedge X{:}f(\overline{X}) \rightarrow Y{=}X \vee \bigvee_{i=1}^{n} X_i{\triangleleft^*}Y$

The rule system B is all we need to deal with lambda binding in the absence of parallelism. The rules state that λ is a function, that binders dominate their bound variables, that binders go from var-labeled nodes to nodes labeled lam, $\exists$ or $\forall$, and that a λ^{-1} literal specifies *all* variables bound by a certain lambda binder.

Properties of λ binding: rule system B

(B.func) $\lambda(X){=}Y \wedge \lambda(U){=}V \wedge X{=}U \rightarrow Y{=}V$

(B.total) $X{:}var \rightarrow \exists Z(\lambda(X){=}Z)$

(B.dom) $\lambda(X){=}Y \rightarrow Y{\triangleleft^*}X$

(B.var) $\lambda(X){=}Y \rightarrow X{:}var$

(B.lam) $\lambda(X){=}Y \rightarrow \exists Z\ (Y{:}lam(Z) \vee Y{:}\exists(Z) \vee Y{:}\forall(Z))$

(B.inv) $\lambda^{-1}(X){=}\{X_1, \ldots, X_n\} \rightarrow \bigwedge_{i=1}^{n} \lambda(X_i){=}X$

(B.all) $\lambda^{-1}(X){=}\{X_1, \ldots, X_n\} \wedge \lambda(Y){=}X \rightarrow \bigvee_{i=1}^{n} Y{=}X_i$

System P contains rules for handling parallelism, and rules that describe the interaction between different correspondence function. We first list the rules that deal with parallelism. As in Section 6, we use the formulas

$$
\begin{aligned}
X{\in}\mathsf{b}(A) &=_{\mathsf{df}} X_0{\lhd^*}X \wedge \textstyle\bigwedge_{i=1}^{n} X({\lhd^*}{\cup}{\perp})X_i \\
X{\in}\mathsf{b}^-(A) &=_{\mathsf{df}} X{\in}\mathsf{b}(A) \wedge \textstyle\bigwedge_{i=1}^{n} X{\neq}X_i \\
X{\notin}\mathsf{b}^-(A) &=_{\mathsf{df}} X({\lhd^+}{\cup}{\perp})X_0 \vee \textstyle\bigvee_{i=1}^{n} X_i{\lhd^*}X \\
X{\in}\mathsf{b}(A_1,\ldots,A_m) &=_{\mathsf{df}} \textstyle\bigvee_{i=1}^{m} X{\in}\mathsf{b}(A_i) \\
X{\in}\mathsf{b}^-(A_1,\ldots,A_m) &=_{\mathsf{df}} \textstyle\bigvee_{i=1}^{m} X{\in}\mathsf{b}^-(A_i)
\end{aligned}
$$

as well as

$$
\begin{aligned}
&\mathrm{co}^-(A,B)(X){=}Y =_{\mathsf{df}} \mathrm{co}(A,B)(X){=}Y \wedge X{\in}\mathsf{b}^-(A) \\
&\mathrm{co}_k^-(\overline{A},\overline{B})(X){=}Y =_{\mathsf{df}} \overline{A}{\sim}\overline{B} \wedge \mathrm{co}^-(A_k,B_k)(X){=}Y
\end{aligned}
$$

Let $\overline{A} = A_1,\ldots,A_n$ and $\overline{B} = B_1,\ldots,B_n$.

Properties of parallelism literals: rule system P

(P.symm) $\overline{A} \sim \overline{B} \rightarrow \overline{B} \sim \overline{A}$

(P.init) $\overline{A} \sim \overline{B} \rightarrow \mathsf{seg}(A_i) \wedge \mathrm{co}(A_i,B_i)(X_i^j){=}Y_i^j$ where $1 \le i \le n$, $A_i = X_i^0/X_i^1,\ldots,X_i^{m_i}$, $B_i = X_i^0/Y_i^1,\ldots,Y_i^{m_i}$, and $0 \le j \le m_i$

(P.new) $\overline{A} \sim \overline{B} \wedge U \in \mathsf{b}(A_i) \rightarrow \exists V\ \mathrm{co}(A_i,B_i)(U){=}V$ where V fresh, $1 \le i \le n$

(P.copy.lab) $\bigwedge_{i=0}^{m} \mathrm{co}(A,B)(X_i){=}Y_i \wedge X_0{:}f(X_1,\ldots,X_m) \wedge X_0{\in}\mathsf{b}^-(A) \rightarrow Y_0{:}f(Y_1,\ldots,Y_m)$

(P.copy.dom) $U_1\ R\ U_2 \wedge \bigwedge_{i=1}^{2} \mathrm{co}(A,B)(U_i){=}V_i \rightarrow V_1\ R\ V_2$

(P.distr.eq) $\varphi \rightarrow X{=}Y \vee X{\neq}Y$ for $X,Y \in \mathcal{V}(\varphi)$

(P.distr.hole) $\overline{A}{\sim}\overline{B} \wedge X_i^0{\lhd^*}X \rightarrow X \in \mathsf{b}(A_i) \vee \bigvee_{j=1}^{m_i} X_i^j{\lhd^+}X$ where $1 \le i \le n$, $A_i = X_i^0/X_i^1,\ldots,X_i^{m_i}$

In section 6 we have introduced correspondence literals $\mathrm{co}(A,B)(U){=}V$ as auxiliary literals to record correspondence. Actually, correspondence literals are just an abbreviation; what we really use are *path equalities*. With $A = X_0/X_1,\ldots,X_n$ and $B = Y_0/Y_1,\ldots,Y_n$, a path equality $\mathrm{p}\binom{X_0\ Y_0}{X\ \ Y}$ states, informally speaking, that X below X_0 corresponds to Y below Y_0. More precisely, the path equality relation $\mathrm{p}\binom{\cdot\ \cdot}{\cdot\ \cdot}$ on a tree θ is the greatest relation on quadruples of nodes such that the following holds: $\mathrm{p}\binom{\pi_1\ \pi_3}{\pi_2\ \pi_4}$ is true iff there exists a path π such that $\pi_2 = \pi_1\pi$ and $\pi_4 = \pi_3\pi$, and for each $\pi'{\lhd^+}\pi$, $L_\theta(\pi_1\pi') = L_\theta(\pi_3\pi')$. This way,

$$\mathrm{co}(A_i,B_i)(U){=}V \text{ stands for } \overline{A}{\sim}\overline{B} \wedge \mathrm{p}\binom{X_0\ Y_0}{U\ \ V} \wedge U{\in}\mathsf{b}(A)$$

for $1 \le i \le |\overline{A}|$, and $A_i = X_0/X_1,\ldots,X_n$ and $B_i = Y_0/Y_1,\ldots,Y_n$.

The main idea about using path equalities is that, as they possess a semantics of their own, they have properties that we can compute with, irrespective of which correspondence function the path equalities in question come from. This is exploited by the saturation rule schemata (P.path...), (P.trans...), and (P.diff...). This way, these schemata ensure the correct interaction of correspondence functions. See [9] for a comprehensive treatment of this topic.

Properties of Path Equality Constraints

(P.path.symm) $\mathrm{p}\left(\begin{smallmatrix} X & Y \\ U & V \end{smallmatrix}\right) \to \mathrm{p}\left(\begin{smallmatrix} Y & X \\ V & U \end{smallmatrix}\right)$

(P.path.dom) $\mathrm{p}\left(\begin{smallmatrix} X & Y \\ U & V \end{smallmatrix}\right) \to X \triangleleft^* U \wedge Y \triangleleft^* V$

(P.path.eq.1) $\mathrm{p}\left(\begin{smallmatrix} X_1 & X_3 \\ X_2 & X_4 \end{smallmatrix}\right) \wedge \bigwedge_{i=1}^{4} X_i{=}Y_i \to \mathrm{p}\left(\begin{smallmatrix} Y_1 & Y_3 \\ Y_2 & Y_4 \end{smallmatrix}\right)$

(P.path.eq.2) $\mathrm{p}\left(\begin{smallmatrix} X & X \\ U & V \end{smallmatrix}\right) \to U{=}V$

(P.trans.h) $\mathrm{p}\left(\begin{smallmatrix} X & Y \\ U & V \end{smallmatrix}\right) \wedge \mathrm{p}\left(\begin{smallmatrix} Y & Z \\ V & W \end{smallmatrix}\right) \to \mathrm{p}\left(\begin{smallmatrix} X & Z \\ U & W \end{smallmatrix}\right)$

(P.trans.v) $\mathrm{p}\left(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix}\right) \wedge \mathrm{p}\left(\begin{smallmatrix} X_2 & Y_2 \\ X_3 & Y_3 \end{smallmatrix}\right) \to \mathrm{p}\left(\begin{smallmatrix} X_1 & Y_1 \\ X_3 & Y_3 \end{smallmatrix}\right)$

(P.diff.1) $\mathrm{p}\left(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix}\right) \wedge \mathrm{p}\left(\begin{smallmatrix} X_1 & Y_1 \\ X_3 & Y_3 \end{smallmatrix}\right) \wedge X_2 \triangleleft^* X_3 \wedge Y_2 \triangleleft^* Y_3 \to \mathrm{p}\left(\begin{smallmatrix} X_2 & Y_2 \\ X_3 & Y_3 \end{smallmatrix}\right)$

(P.diff.2) $\mathrm{p}\left(\begin{smallmatrix} X_1 & Y_1 \\ X_3 & Y_3 \end{smallmatrix}\right) \wedge \mathrm{p}\left(\begin{smallmatrix} X_2 & Y_2 \\ X_3 & Y_3 \end{smallmatrix}\right) \wedge X_1 \triangleleft^* X_2 \wedge Y_1 \triangleleft^* Y_2 \to \mathrm{p}\left(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix}\right)$

Rule system L describes the interaction between parallelism and binding. We use the formula

$$\mathsf{distr}_{\overline{A}}(U) =_{\mathsf{df}} \bigwedge_{i=1}^{n} (U \in \mathsf{b}^-(A_i) \vee U \notin \mathsf{b}^-(A_i)).$$

in (L.distr.1) and (L.distr.2). (L.inverse) uses two more formulas, the first just being

$$\lambda(X){\neq}Y =_{\mathsf{df}} \exists Z(\lambda(X){=}Z \wedge Z{\neq}Y).$$

As explained in Section 6, the second formula collects, for a finite set S_1 of variables, all correspondents with respect to $\overline{A} \sim \overline{B}$. Let S_1, S_2 stand for finite sets of variables, and let $\overline{A} = A_1 \ldots, A_n$. Then

$$\begin{aligned} \mathrm{co}^-(\overline{A},\overline{B})(S_1){=}S_2 =_{\mathsf{df}} & \bigwedge_{i=1}^{n} \bigwedge_{X \in S_1} (X \notin \mathsf{b}^-(A_i) \vee \bigvee_{Y \in S_2} \mathrm{co}_i^-(\overline{A},\overline{B})(X){=}Y) \\ \wedge & \bigwedge_{Y \in S_2} \bigvee_{X \in S_1} \bigvee_{i=1}^{n} \mathrm{co}_i^-(\overline{A},\overline{B})(X){=}Y \end{aligned}$$

Interaction between parallelism and binding: rule system L

(L.same.seg) $\lambda(U_1){=}U_2 \wedge \bigwedge_{i=1}^{2} \mathrm{co}_k^-(\overline{A},\overline{B})(U_i){=}V_i \to \lambda(V_1){=}V_2$

(L.diff.seg) $\lambda(U_1){=}U_2 \wedge \bigwedge_{i=1}^{2} \mathrm{co}_{k_i}^-(\overline{A},\overline{B})(U_i){=}V_i \wedge U_2 \notin \mathsf{b}^-(A_{k_1}) \to \lambda(V_1){=}V_2$

(L.outside) $\lambda(U){=}Y \wedge \mathrm{co}_k^-(\overline{A},\overline{B})(U){=}V \wedge Y \notin \mathsf{b}^-(\overline{A}) \to \lambda(V){=}Y$

(L.hang) $\lambda(U_1){=}U_2 \wedge \overline{A}{\sim}\overline{B} \wedge U_2 \in \mathsf{b}^-(\overline{A}) \to U_1 \in \mathsf{b}^-(\overline{A})$

(L.distr.1) $\lambda(U_1){=}U_2 \wedge \overline{A}{\sim}\overline{B} \wedge U_1 \in \mathsf{b}^-(\overline{A}) \to \mathsf{distr}_{\overline{A}}(U_2)$

(L.distr.2) $\lambda(U_1){=}U_2 \wedge \overline{A}{\sim}\overline{B} \wedge U_2 \in \mathsf{b}^-(\overline{A}) \to \mathsf{distr}_{\overline{A}}(U_1)$

(L.equal) $\lambda(X_1){=}X_2 \wedge \bigwedge_{i=1}^{2} X_i{=}Y_i \to \lambda(Y_1){=}Y_2$

(L.inverse) $\lambda^{-1}(X){=}S_1 \wedge \mathrm{co}_k^-(\overline{A},\overline{B})(X){=}Y \wedge \mathrm{co}^-(\overline{A},\overline{B})(S_1){=}S_2 \cup S_3 \wedge$
$\bigwedge_{V \in S_2} \lambda(V){=}Y \wedge \bigwedge_{V \in S_3} \lambda(V){\neq}Y \to \lambda^{-1}(Y){=}S_2$