Fachbereich 14 Informatik
Universität des Saarlandes
Forschungsbereich Programmiersprachen

Beta Reduction Constraints

Diplomarbeit

Angefertigt unter der Leitung von Prof. Dr. Gert Smolka
Zweitgutachter: Prof. Dr. Manfred Pinkal
Betreuung: Dr. Joachim Niehren

Manuel Bodirsky

Eingereicht am 29. März 2001

Diese Arbeit wurde finanziell unterstützt vom Forschungsbereich Programmiersysteme, dem Procope Projekt und durch die Projekte NEP und CHORUS des Sonderforschungsbereiches 378 'Resourcenadaptive Kognitive Prozesse'.

Hiermit erkläre ich, daß ich diese Arbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Saarbrücken, 29. März 2001

Manuel Bodirsky

Kurzfassung

Diese Arbeit beschäftigt sich mit Constraintsprachen zur partiellen Beschreibung von Lambdatermen. Insbesondere werden Betareduktionsconstraints eingeführt, die ausdrücken daß ein Lambdaterm Redukt eines anderen ist. Wir untersuchen die Ausdrucksstärke von Betareduktionsconstraints, indem wir sie mit Parallelismusconstraints vergleichen. Schließlich zeigen wir Anwendungen von Betareduktionsconstraints bei der semantischen Unterspezifikation in der Computerlinguistik.

Abstract

This thesis investigates languages that partially describe lambda terms. In particular, beta reduction constraints are introduced, which express that one lambda term beta-reduces to another. We investigate the expressive power of beta reduction constraints by relating them to parallelism constraints, and give applications of beta reduction constraints in semantical underspecification of computational linguistics.

Acknowledgements

First I would like to thank professor Gerd Smolka, and the programming systems lab, for supporting me with a perfect research environment.

I am indebted to Joachim Niehren, who draw my attention to underspecification and constraint technology. He gave me motivation and during all the different phases of the work he was a great discussion partner. He always confirmed me in the right ideas and I learned a lot when he helped me bringing them into a better shape.

Then I want to thank Katrin Erk, Alexander Koller, Sebastian Pado, Prof. Manfred Pinkal and the other members of the CHORUS team for the work we did together, their commitment and their enthusiasm for the project.

Finally, I have to thank my fellow students Tobias Gärtner, Timo von Oertzen and Jan Schwinghammer, who brought fun into my studies in Saarbrücken, for their remarks on later versions of this thesis, and my dear family.

Contents

1 Introduction

This thesis investigates expressive constraint languages that partially describe lambda terms [Egg et al., 2001, Erk et al., 2001, Bodirsky et al., 2001a]. In particular, beta reduction constraints are introduced. We will study their expressiveness by relating them to parallelism constraints in the constraint language for lambda structures, CLLS. Solving beta reduction constraints will have applications in the field of semantic underspecification of computational linguistics.

The constraint languages presented here extend standard tree description languages [Marcus et al., 1983, Backofen et al., 1995, Duchier and Niehren, 2000] based on dominance constraints. Tree description languages are first-order languages that partially describe finite labeled trees. These languages have variables ranging over nodes in the tree; they provide constraints about edge relationships and labeling.

Tree descriptions have many applications in modern computational linguistics, in syntax as well as in semantics [Rogers and Vijay-Shanker, 1994, Duchier, 2000]. Since trees are the basic concept of syntactic analysis, parsing of natural language can be seen as constraint solving for tree description constraints [Duchier, 1999, Duchier and Thater, 1999].

One example for applications of the description languages in semantics is the constraint language for lambda structures (CLLS) [Egg et al., 1998, 2001, Erk et al., 2001]. CLLS is a first-order language to partially describe λ-terms. It was developed as a formalism for underspecified semantics. Semantic underspecification [Reyle, 1993, van Deemter and Peters, 1996, Pinkal, 1996] is an approach to deal with semantic ambiguity in natural language sentences. A single CLLS constraint may be satisfied by many λ-terms, and can thus represent several readings of a sentence at the same time.

The idea to describe λ-terms with (extensions of) tree description languages is to see a λ-term as a tree equipped with additional binding edges (see Figure 1.1 on the following page). One can then describe a λ-term partially as one would describe a standard tree structure. But in addition to dominance constraints in standard tree description languages CLLS also provides parallelism [Erk and Niehren, 2000]

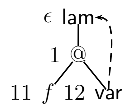

Figure 1.1: The lambda structure of $\lambda x.@(f\ x)$

and binding constraints.

An important problem for CLLS is *underspecified beta reduction.* Given a CLLS constraint representing many lambda terms, the aim is to efficiently compute a compact description of the reducts after reducing at all specified redexes. In particular, we want to avoid enumerating and individually beta-reducing the described lambda terms. In this thesis we show that *beta reduction constraints* lay the foundation for underspecified beta reduction.

Beta reduction constraints conservatively extend CLLS. They declaratively describe a single β-reduction step. With beta reduction constraints, we can lift β-reduction to descriptions of λ-terms. These descriptions are at first very implicit, and can be made explicit by *solving* the constraints. To this end, we express beta reduction constraints with *group parallelism constraints.* Parallelism constraints describe, that some segments of a tree look the same, i.e. segments are structurally equal. We will slightly extend parallelism that is already an integral part of CLLS to the notion of *group parallelism.* This will enable us to adapt a known semi-decision procedure for CLLS to also deal with beta-reduction constraints.

Next, we will express a large fragment of parallelism constraints with beta reduction constraints. Thus it is likely that parallelism constraints are not an overhead for the treatment of beta reduction constraints. CLLS including parallelism is equally expressive as context unification [Niehren et al., 1997, Niehren and Koller, 1998], whose decidability is an open problem in theoretical computer science [RTA-List, 1998, Lévy, 1996, Schmidt-Schauß and Schulz, 1999].

Plan of the Introduction: We will now go through an example in the application of computational linguistics to give intuitions and motivation for the later formal sections of this thesis. First, we will explain why the lambda calculus and beta reduction is an important tool in computational semantics. Then we will motivate underspecified semantic descriptions for computational linguistics. After that we can formulate the problem of *underspecified beta reduction.*

1.1 Lambda Calculus in Computational Semantics

Lambda terms and beta reduction are the subject of the classical lambda calculus [Barendregt, 1984], and also play an important rôle in higher order logic (HOL) theorem proving [Huet, 1975]. Seen as a syntactic operation, β-reduction serves as the basic evaluation step in operational semantics of many programming languages, since (among many other reasons) it models fundamental concepts such as variable binding and scope. We now want to demonstrate by example how the lambda calculus is used in semantics of computational linguistic.

Consider the natural language sentences

Every student laughs. (1.1)

Peter and Marc love mary. (1.2)

We would like to assign them some meaning representation in an automated process. If we choose classical predicate logic to represent the meaning of these sentence, the representations look like

$$\forall x.(\mathsf{stud}(x) \rightarrow \mathsf{laugh}(x)) \qquad (1.3)$$

$$\mathsf{love}(\mathsf{peter}, \mathsf{mary}) \wedge \mathsf{love}(\mathsf{marc}, \mathsf{mary}) \qquad (1.4)$$

Lets look at 1.3: here we state in logic formulas that for all x, if x is a student, is also laughs. Note that some parts in the natural language sentence, such as the quantifier Every in 1.1, influence the shape of the formula at several positions. Here, Every translates to the logical quantifier $\forall x$ and the implication $\rightarrow$ between the restriction and the scope of the quantifier. In the second sentence, one of the constituents occurs twice in the semantic representation, namely love, since both peter and Marc love Mary.

We say, the semantic construction does not follow the principle of *compositionality.* We would rather like to derive the semantics of a natural language sentence by combining the semantics of its components in a simple way.

In 1.5 on the next page, we show a higher order analysis of the first of the above sentences (due to [Montague, 1970, 1974]; see [Gamut, 1991] for an introduction) that is derived compositionally.

The idea behind using lambda calculus for compositional analysis is to understand a quantifier as a function, that takes two properties as argument and gives back a truth value. For universal quantification this value is true, if every element that has the first property also has the second property. In the case of *every* this reads in λ-terms as follows[1]:

$$\mathsf{Every} = \lambda P.\lambda Q.(\forall x.(@(P\ x) \to @(Q\ x)))$$

In our example, the property to be a student is called the *restriction*, and the property to pay attention is called the *scope* of the quantifier *Every*.

$$@(@(\mathsf{Every}\ \mathsf{Student})\ \mathsf{Payatt}) \tag{1.5}$$

Student and Payatt are themselves λ-terms, where we abstract to use them as properties:

$$\mathsf{Student} = \lambda x.@(\mathsf{stud}\ x) \tag{1.6}$$
$$\mathsf{Payatt} = \lambda x.@(\mathsf{payatt}\ x) \tag{1.7}$$

Note that every part in the natural language sentence translates directly to a λ-term, and the different parts of the sentences are simply connected with application symbols according to the structure of the sentence.

Beta reduction is a reduction relation that is defined on λ-terms: We replace a λ-term $C(@(\lambda x.B\ A))$ by its reduct $C(B[x/A])$. If we β-reduce the λ-term in 1.5 several times, we will get the formula in 1.3 [2].

$$\begin{aligned}
@(@(\mathsf{Every}\ \mathsf{Student})\ \mathsf{Payatt}) &\to_\beta @(\lambda Q.(\forall x.(@(\mathsf{Student}\ x)) \to @(Q\ x))\ \mathsf{Payatt})\\
&\to_\beta \forall x.(@(\lambda x.@(\mathsf{stud}\ x)\ x) \to @(\mathsf{Payatt}\ x))\\
&\to_\beta \forall x.@(\mathsf{stud}\ x) \to @(\lambda x.@(\mathsf{stud}\ x)\ x)\\
&\to_\beta \forall x.@(\mathsf{stud}\ x) \to @(\mathsf{payatt}\ x)
\end{aligned}$$

This means that we can do both compositional analysis *and* get the desired simple representations. This is, why λ-terms play an important rôle when doing semantics. Now we want to motivate, why it is so interesting to have languages that partially describe λ-terms.

[1] In higher order logics it is standard to only use constant symbols (of appropriate type) to represent relations. The formula $\mathsf{stud}(x)$ thus becomes $@(\mathsf{stud}, x)$, where @ stands for application. We will recall some basics of the λ-calculus in Chapter 2 on page 19.

[2] except the difference in analyzing relations that we already mentioned

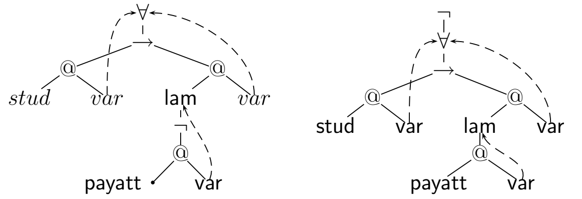

Figure 1.2: The two readings of *Every student did not pay attention.*

1.2 Semantic Underspecification

One of the challenging problems in computational linguistics is to deal with ambiguity. Ambiguity is omnipresent in natural language. We will give two examples.

Every student does not pay attention. (1.8)

Peter read his paper. And so did Marc. (1.9)

In the first case there is a so called *scope ambiguity*. There is the reading that there is a student who does not pay attention. The other possible meaning is that none of the students pays attention. We say that the negation *not* can have wide scope (it is not the case that every student does pay attention.) or narrow scope (for every student it is not the case that every student pays attention).

The second ambiguity is a result of interaction between an ellipsis and an anaphorical reference (*his* paper). There is the reading that Marc also read Peters Paper. And the reading that Marc also read his own paper.

There are much more potential sources of ambiguities. Since usually they disambiguate independently from each other, the total number of readings grows exponentially in the number of ambiguities. Thus, enumerating all of them for further processing steps in applications is not feasible.

In this thesis, whenever using linguistic examples for illustration, we are considering only scope ambiguities. Even if we restrict ourselves to only one source of ambiguity, it is not difficult to find examples with huge numbers of readings, for instance the sentence *Every student of a university in some German city does not want to apply in every department of a company.* The reader may try to spell out the different meanings.

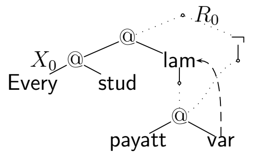

Figure 1.3: Underspecified Description of 'Every student did not pay attention'

We want to take a closer look at the simpler example *Every student does not pay attention*. The two readings can be represented in predicate calculus as

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

The systematically and compositionally derived λ-terms for this sentence look as follows:

$$@(@(\mathsf{Every\ Student})\ \lambda x.(\neg @(\mathsf{payatt}\ x)))$$
$$@(@(\neg((\mathsf{Every\ Student})\ \lambda x.@(\mathsf{payatt}\ x)))$$

The idea of underspecification is to work with a single, compact representation that represents all the different readings [Reyle, 1993, van Deemter and Peters, 1996, Pinkal, 1996]. One approach to underspecification is to separate a meta language and an object language: The object language is a formalism that describes the meaning of a sentence, the meta language describes formulas in this formalism.

An elegant formalism for such a meta language in semantic underspecification is the constraint language for λ-structures, CLLS [Pinkal, 1996, Erk et al., 2001] (see [Koller, 1999] for comparison with other formalisms). CLLS describes λ-structures, i.e. tree structures with an additional binding function (see Figure 1.1 on page 10). They represent (untyped) λ-terms as known from classical lambda calculus (for a formal definition of all these notions see Chapter 2 on page 19 and Chapter 3 on page 29). CLLS provides constraints for labeling, dominance, lambda binding and *parallelism*, i.e. structural similarity of tree segments.

It is easy to specify CLLS constraints by drawing its constraint graph. The CLLS constraint for the above example is given in Figure 1.3. A solid line represents a parent-relationship, dotted lines stand for the dominance relation. Note that the visualization of constraints are graphs, and in contrast to their models not necessarily trees, since dotted lines for dominance edges can lead to the same node. Note that both λ-terms shown above are models of the constraint in 1.3.

Of course the constraint has much more solutions, some of which are clearly

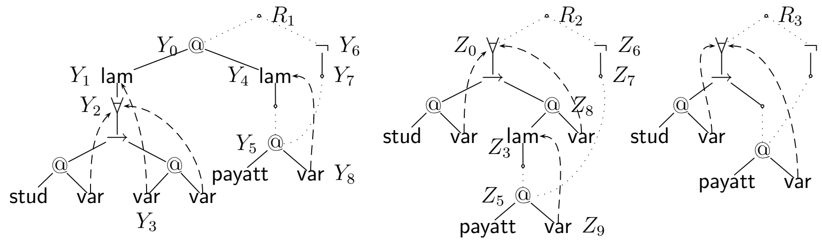

Figure 1.4: Underspecified β-reduction steps for 'Every student did not pay attention'

not readings of the initial sentence. But this does not limit the approach, since it mostly suffices to consider the solutions that do not introduce *new material*, i.e. there is a labeling constraint for every node in the model. However, in this form the approach is flexible enough to deal with phenomena like *reinterpretation* [Koller et al., 2000b, Striegnitz, 1999], where new material has to be inserted to get reasonable readings of the sentence.

We only have advantages of underspecified representations, if we are able to avoid enumerating all readings when performing further processing steps. As we have already seen, an important processing step on semantic representations is β-reduction. Since β-reductions simplify the λ-structures greatly, we are interested in *underspecified beta reduction* [Bodirsky et al., 2001b]: Is it possible to lift β-reduction to descriptions of λ-structures in CLLS? We will see some examples showing what underspecified beta reduction should do, and why the problem is nontrivial.

We would like to give a description of the models in figure 1.3 on the preceding page that are β-reduced as far as possible, i.e. we would like to β-reduce the description itself.

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 1.4. 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 Figure 1.4. For the last 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. Finally the result is precisely a description of the two readings as first-order formulas (Figure

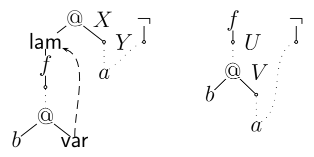

Figure 1.5: Problems with rewriting of descriptions

1.2).

Judging from this first example, the problem does not look too difficult. Twice, we did not even know what exactly the parts of the redex looked like, but it was still easy to derive correct descriptions of the reducts. But this is not always the case. Consider Figure 1.5, an abstract but simple example. In the left description, there are two possible positions for the ¬: above X or below Y. Proceeding naïvely as above, we arrive at the right description in Figure 1.5. 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, since the resulting descriptions may have the wrong solutions.

We attack this problem using *beta reduction constraints*. They describe that a certain subtree is a *beta*-reduct of another subtree. Thus, in contrast to the graph rewriting approach mentioned above, with beta reduction constraints we can have a sound description of the reducts of a constraint per definition.

But to really solve the problem of underspecified beta reduction, we have to find out how to make these descriptions more explicit. What is the underspecified description of the results after subsequently reducing at several specified redexes?

1.3 Contributions of this Thesis

We introduce β-reduction constraints in Chapter 4 and show that β-reduction constraints can be translated into group parallelism constraints, a generalization of parallelism constraints in CLLS. This is an important result, since an existing solution procedure for CLLS can be extended to group parallelism. We thus have a method to make the tree descriptions containing β-reduction constraints more explicit. Explicit information about the structure of the solutions might be important for further processing steps in natural language applications.

In chapter 5 of the thesis, we investigate the relation between parallelism and

β-reduction and the expressive power of their constraint languages. We show that β-reduction constraints can express similarity constraints. Using this, we express non self-overlapping parallelism constraints. It is thus very likely that parallelism solvers are not an overkill for beta reduction constraints.

Chapter 6 will present an algorithm for underspecified β-reduction. Beta reduction constraints only describe a single β-reduction step. In practice, however, we are interested in describing the result of several β-reduction steps. For instance in the application of underspecified semantics of natural language, we can assume simply typed λ-structures. Thus, on every model of an underspecified description, every sequence of beta-reduction steps must terminate. So we are interested in describing for a given underspecified description the result after reducing at *all* specified redexes. The algorithm presented here uses a saturation procedure to solve β-reduction constraints that builds on an existing solver for parallelism constraints. The procedure avoids distribution rules by additional propagation rules, and therefore is truly underspecified.

Related Publications: This work contains results published in [Bodirsky et al., 2001a]. A paper about the procedure in Chapter 6 has been submitted [Bodirsky et al., 2001b].

2 Lambda Structures

In this chapter, we will define lambda structures. Since lambda structures will represent higher order logic (HOL) formulas, we will first very briefly introduce some elements of the classical lambda calculus. We then define some important notions for lambda structures and finally show how they correspond to the respective notions for λ-terms.

2.1 Prerequisites

Since lambda structures can represent higher order logic formulas, we will introduce the syntax of higher order logic (HOL) in this section. HOL is an extension of first order logic. The idea of the extension is to provide constructions for more complicated mathematical objects in the logical language, and on the other side to introduce types to ensure the consistency of the system. It is a formalism that is popular in computational linguistic, since it has the expressiveness of first order logic while avoiding involved set theoretic encodings. Thus HOL allows natural formalizations of semantics of natural language sentences.

Furthermore, there is a nice *syntactically defined* relation on HOL-formulas, the β-reduction relation. Beta reduction has very nice properties as a reduction system, e.g. it is confluent. If the HOL-formulas (that are also called λ-terms) are simply typed they have a unique normal form with respect to β-reduction.

We do not worry about types of λ-terms in this thesis. We see λ-terms as a syntactic objects, where a particularly interesting relation is defined on. Lets assume a countable signature $\Sigma = \{f, g, \dots\}$ of function symbols, each equipped with an arity $\mathsf{ar}(f) \geq 0$, that we sometimes indicate by superscript f^n. Symbols of arity 0 are constants, written as $a, b, \dots$. Finally we need a countable set of variables $x, y, \dots$. Then (untyped) λ-terms are defined as follows:

1. Every variable and every constant symbol is a λ-term.

2. If A and B are λ-terms, then so is $@(A\ B)$
   (A *applied to* B).

3. If A is a λ-term and x is a variable, then $\lambda x.A$ is also a λ-term.

4. If $A_1, \ldots, A_n$ are λ-terms, then $f^n(A_1\ \ldots\ A_n)$ is a λ-term.

We define free, bound and global variables as usual.

If lambda structures represent HOL formulas, we represent the connectives $\forall$, $\rightarrow$, and $\neg$ as logical symbols (i.e. as elements in Σ), and we allow var-labeled nodes to be bound by a $\forall$- or $\exists$-labeled node. This makes the examples much more readable. In the formal parts of this thesis we adopt the standard approach of analyzing them as constants of appropriate type in the signature, and hence leafs in the structure; thus, wherever needed we can assume that all var-labeled nodes are bound by a lam-labeled node.

The beta reduction step on λ-terms 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. An λ-term $@(\lambda x.B\ A)$ is called a redex (*red*ucible *ex*pression). We call C the context, B the body, and A the argument of the reduction step. By *x has to be free for A* we mean that there is binder in B that would bind any variable in A. For instance, one cannot simply β-reduce $(\lambda x.\lambda y.x)y$ without renaming the bound occurrence of y beforehand. Otherwise, the global variable y in the argument got captured by the binder λy in the body.

We can always ensure this by using different variable names for every binder. Thus it makes sense to consider equality up to α-conversion $=_\alpha$, i.e. consistent renaming of locally bound variables.

$$C(\lambda x.B) \quad =_\alpha \quad C(\lambda y.B) \quad \text{for all variable symbols } y \text{ that are not free in} B$$

A useful notion when working with λ-terms are *contexts* $A, B, C, \ldots$. Contexts are λ-terms built over a signature Σ that also contains an additional constant symbol $\bullet$. They provide an easy way to specify certain functions from λ-terms to λ-terms. For instance $f(a\ \bullet)$ is a function which maps $g(b)$ to $f(a\ g(b))$.

Definition 1 (Contexts). *A context $A, B, C, \ldots$ is a λ-term built over the signature Σ containing the 0-ary constant symbol $\bullet$.*

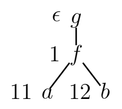

Figure 2.1: The tree structure of $g(f(a\ b))$.

Let n be the number of occurrences of $\bullet$ in a context A. Then A specifies a n-ary function f_A from λ-terms to λ-terms. Let $A_1, \ldots, A_n$ be λ-terms. The image of $f_A(A_1, \ldots, A_n)$ is defined as A where the i-th occurrence of the symbol $\bullet$ in A is replaced by A_i.

Note that although two contexts are α-equivalent, they might specify different functions. Consider for instance the context $\lambda x.\bullet$ that is α-equivalent to $\lambda y.\bullet$. But if we apply them to the λ-term x, the results are $\lambda x.x$ and $\lambda y.x$ and they are not α-equivalent.

In the next chapter, we will see how to represent λ-terms with a tree structure and an additional binding function.

2.2 Lambda Structures

We first repeat the standard definition of trees, where trees are defined using a tree domain and a labeling function. In the second part of this section, we extend tree structures to lambda structures, that can deal with variable binding. This is necessary, since we want to interpret the lambda structures as logical formulas.

Let $(\mathbb{N}^+)^*$ be the set of all words generated over the positive natural numbers. A tree domain D is a non-empty subset of $(\mathbb{N}^+)^*$, such that

- D is prefix-closed: if $w \in D$, and w' is a prefix of w, then $w' \in D$.
- D always contains the older brothers of any of it's nodes:

$$\forall w \in (\mathbb{N}^+)^*, \forall i > 0 : w{\cdot}i \in D \Rightarrow \forall 0 < j < i : w{\cdot}j \in D$$

We will call the elements of D *positions* or *nodes*. Next, we assume a signature $\Sigma = \{f, g, \ldots\}$ of function symbols, each equipped with an arity $\mathsf{ar}(f) \geq 0$, that we sometimes indicate by superscript f^n. Symbols of arity 0 are constants, written as $a, b, \ldots$

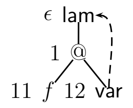

Figure 2.2: The lambda structure of $\lambda x.@(f\ x)$

Definition 2. *A Σ-tree structure θ is a pair (D_θ, L_θ) of a tree domain D_θ and a labeling function L_θ from D_θ to Σ, such that for all $w \in D_\theta$, if the arity of $L_\theta(w)$ is n, then $w{\cdot}n \in D_\theta$ but $w{\cdot}(n+1) \notin D_\theta$.*

We write $\theta.\pi$ for the subtree at position π. ϵ is the empty path, and $\pi_1{\cdot}\pi_2$ the concatenation of π_1 and π_2. π is a *prefix* of a path π' if there is a (possibly empty) π'' such that $\pi{\cdot}\pi'' = \pi'$.

A Σ-tree structure uniquely corresponds to a ground term over Σ, thus we also specify trees by expressions like $g(f(a\ b))$. One then obtains the tree domain D_θ, i.e. the set of all nodes, by recursively traversing the term structure:

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

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). The tree uses these symbols to reflect the structure of the λ-term.

The binding function λ explicitly maps var-labeled nodes to lam-labeled nodes binding them. For example, Fig. 2.2 shows a representation of the term $\lambda x.f(x)$. Here $\lambda(12) = \epsilon$. Such an extension of a tree structure by a binding function is called a lambda structure.

Definition 3. A λ-*structure* τ is a triple $(D_\theta, L_\theta, \lambda)$ of a tree $\theta = (D_\theta, L_\theta)$ and a binding function $\lambda : L_\theta^{-1}(\mathsf{var}) \to L_\theta^{-1}(\mathsf{lam})$ such that $\lambda(\pi)$ is always a prefix of π.

A λ-structure corresponds uniquely to a closed λ-term modulo α-renaming, i.e. consistent renaming of locally bound variables. We will freely consider λ-structures as first-order model 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 parallelism and β-reduction relations on top of these. The labeling relation $\pi{:}f(\pi_1, \ldots, \pi_n)$ holds in a λ-structure τ if $L_\theta(\pi) = f^n$ and $\pi_i = \pi i$ for all $1 \leq i \leq n$. *Dominance* $\lhd^*$ is the prefix relation between paths of D_θ; inequality $\neq$ is simply inequality of paths; *disjointness* $\pi \perp \pi'$ holds if neither $\pi \lhd^* \pi'$ nor $\pi' \lhd^* \pi$.

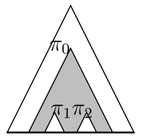

Figure 2.3: The tree segment $\pi_0/\pi_1, \pi_2$

We will also consider intersections, unions, and complements of these relations; for instance, *proper dominance* $\lhd^+$ is $\lhd^* \cap \neq$, and *equality* $=$ is $\lhd^* \cap \rhd^*$.

2.3 Contexts in Lambda Structures

It is clear that lambda structures can be translated in closed lambda terms and vice versa. But we would also like to have a notion on lambda structures that corresponds to the notion of contexts in lambda terms.

To talk about parts of trees we now define *tree segments*. Intuitively, a tree segment is a part of a tree which starts at a certain node, but potentially with some subtrees missing (see Figure 2.3). For instance the context, body, and argument of a redex in a λ-structure will all be tree segments.

Definition 4. *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 (we read this: the tree segment π_0* up to *π_1 to π_n), such that $\pi_0 \lhd^* \pi_i$ and $\pi_i(\perp\cup=)\pi_j$ holds in τ 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*

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

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

Now, we define how to read off contexts from λ-structures. It is clear that we can assign a context to every tree segment. The holes of the tree segment then correspond to the occurrences of $\bullet$ in the λ-term representing the context. The only thing one has to be careful about is the treatment of variable binding. In λ-terms, binding is specified by names for the variables modulo α-equivalence, whereas λ-structures provide an explicit binding function. If we want to translate from λ-structures into λ-terms, we have to consistently find names for the variables

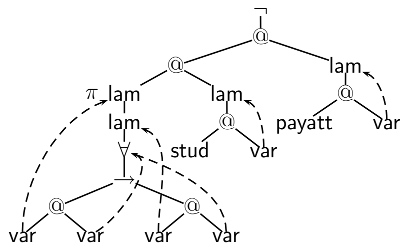

Figure 2.4: Description of a HOL-formula for 'Every student did not pay attention'

at the var-labeled positions in the λ-structure. Assigning variable names that are indexed with the position of the binder to the var-labeled nodes will do the trick.

Definition 5. *Let τ be a λ-structure and $\alpha = \pi_0/\pi_1, \ldots \pi_n$ be a tree segment in τ. We now define the context $T_\tau(\alpha)$ of the tree segment α using the fresh variable names x_π. We define T inductively using $T_\tau(\alpha) =_{\mathsf{df}} T_\alpha(\pi_0)$, where*

$$T_\alpha(\pi) =_{\mathsf{df}} \begin{cases} \bullet & \pi = \pi_i, 1 \leq i \leq n \\ x_{\lambda(\pi)} & L(\pi) = \mathsf{var} \\ \lambda x_\pi . T_\alpha(\pi \cdot 1) & L(\pi) = \mathsf{lam} \\ f(T_\alpha(\pi \cdot 1) \ldots T_\alpha(\pi \cdot n)) & L(\pi) = f^n \end{cases}$$

Examples: We could also define $\mathsf{Every} = \lambda P.\lambda Q.(\forall x(@(P\ x) \rightarrow @(Q\ x)))$ by reading it off the λ-structure in Figure 2.4, using the new definitions of Section 2.3 on the preceding page:

$$\mathsf{Every} = T_\tau(\pi/)$$

The context $\neg(\bullet)$, assigning every formula its negation, can be read off the tree segment at top of the tree in Figure 2.4:

$$T_\tau(\epsilon/1\ ) = \neg(\bullet)$$

2.4 Correspondence Functions

We defined what it means for two tree segments 'to look the same'. This is the case when the contexts read off at the tree segment are equal up to local renaming

of variables. We would rather like to have a direct and simple characterization of structural equality of tree segments. To this end, we introduce the important concept of *correspondence functions*, that will also play an crucial rôle when defining and processing beta reduction and parallelism constraints in later chapters.

Definition 6. *A* correspondence function *between the tree segments α, β in a λ-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 π in $\mathsf{b}^-(\alpha)$ and label f:*
$$\pi{:}f(\pi 1, \ldots, \pi n) \Leftrightarrow c(\pi){:}f(c(\pi 1), \ldots c(\pi n))$$
4. *Internally bound variables are bound correspondingly. For $\pi \in \mathsf{b}^-(\alpha)$ such that $\lambda(\pi) \in \mathsf{b}^-(\alpha)$:*
$$\lambda(c(\pi)) = c(\lambda(\pi)).$$

Note that we also require internal binding to be consistent (Property 4), in contrast to [Bodirsky et al., 2001a]. Since we have to prove some technical results, it is much nicer to have this property for correspondence functions, for instance it is necessary in Lemma 1.

The next lemma will be used very often, so we do not mention it explicitly in some later proofs. It will give us the possibility to either use term terminology or to use correspondence functions when speaking about structurally equal contexts in a lambda structure. Very briefly, the lemma says that there is a correspondence function between two tree segments if and only if the context functions for these tree segments are the same. This is very easy to see, but one has to be careful about binders. Especially correspondence functions do not impose any restrictions on global variables. Therefore we only get equality of contexts up to some renaming of the global variables of the context.

Lemma 1. *Let τ be a λ-structure, and $\delta = \pi_0/\pi_1, \ldots, \pi_n$, $\delta' = \pi'_0/\pi'_1, \ldots, \pi'_n$ tree segments in τ. Then there exists a correspondence function between δ and δ' if and only if there is a renaming ρ of the global variables such that*

$$\rho(T_\tau(\delta)) =_\alpha \rho(T_\tau(\delta'))$$

Proof. Let c be such a correspondence function between δ and δ' in τ. We will show by induction over the tree structure, that $\rho(T_\delta(\omega)) =_\alpha \rho(T_{\delta'}(c(\omega)))$ for $\omega \in \mathsf{b}^-(\delta)$, where

$$\rho(x_\pi) =_{\mathsf{df}} \begin{cases} x_{c(\pi)} & \text{if } \pi \in \mathsf{b}^-(\delta) \\ \mathsf{var} & \text{otherwise} \end{cases}$$

In particular, $\rho(T_\delta(\pi_0)) =_\alpha \rho(T_{\delta'}(\pi_0'))$ and therefore $\rho(T_\tau(\delta)) =_\alpha \rho(T_\tau(\delta'))$ (see Definition 5 on page 24).

We distinguish four different cases:

1. The node ω is in one of the nodes $\{\pi_1, \dots, \pi_n\}$. Thus $\rho(T_\delta(\omega)) = \rho(T_\delta(\pi_i)) = \rho(A_i) = \bullet$ for some $1 \le i \le n$. Since c is a correspondence function we know that $\rho(T_{\delta'}(c(\pi_i))) = \rho(T_{\delta'}(\pi_i')) = \rho(A_i) = \bullet$.
2. The node ω is labeled $\omega{:}f(\omega_1, \dots, \omega_m)$ for some $f^m \in \Sigma$. Thus $\rho(T_\delta(\omega)) = f(\rho(T_\delta(\omega_1)) \dots \rho(T_\delta(\omega_m)))$ which by induction hypothesis equals $f(\rho(T_{\delta'}(c(\omega_1))) \dots \rho(T_{\delta'}(c(\omega_m))))$. As c is a correspondence function, $c(\omega){:}f(c(\omega_1) \dots c(\omega_m))$, and therefore $\rho(T_\delta(\omega)) = f(\rho(T_{\delta'}(c(\omega_1))) \dots \rho(T_{\delta'}(c(\omega_m)))) = \rho(T_{\delta'}(c(\omega)))$.
3. ω is a var-labeled node. Here, either ω is bound in $\mathsf{b}^-(\delta)$, and then $\rho(T_\delta(\omega)) = \rho(x_{\lambda(\omega)}) = x_{c(\lambda(\omega))} = x_{\lambda(c(\omega))} = T_{\delta'}(c(\omega)) = \rho(T_{\delta'}(c(\omega)))$, again we used that c is a correspondence function. Or ω is bound above π_0. Then $\rho(T_\delta(\omega)) = \rho(x_\omega) = \mathsf{var} = \rho(T_{\delta'}(c(\omega)))$.
4. There is an abstraction $\omega : \mathsf{lam}(\omega_1)$ at position ω. This implies $\rho(T_\delta(\omega)) = \lambda x_\omega.\rho(T_\delta(\omega_1))$ and by induction hypothesis this is $\lambda x_\omega.\rho(T_{\delta'}(c(\omega_1)))$. Local consistent renaming gives us the desired equation

$$\rho(T_\delta(\omega)) = \lambda x_\omega.\rho(T_{\delta'}(c(\omega_1))) =_\alpha \rho(\lambda x_{c(\omega_1)}.T_{\delta'}(c(\omega_1)) = \rho(T_{\delta'}(c(\omega))).$$

For the other direction, suppose that $\rho(T(\delta)) =_\alpha \rho(T(\delta'))$ for some renaming ρ. We have to show, that there is a correspondence function c between δ and $\mathsf{b}(\delta')$. We choose

$$c(\pi_0 \cdot \omega) =_{\mathsf{df}} \pi_0' \cdot \omega \qquad \text{for all } \omega \text{ such that } \pi_0\omega \in \mathsf{b}^-(\delta)$$

and show that c preserves the root, holes, labeling and internal binding (Definition 6 on the preceding page). To do so we first show by induction that

$$\rho'(T_\delta(\pi)) =_\alpha \rho'(T_{\delta'}(c(\pi))) \qquad \pi \in \mathsf{b}^-(\delta) \tag{2.1}$$

where $\rho'(x_\omega) = \rho(x_\omega)$, if ω is not in $\mathsf{b}^-(\delta)$, and $\rho'(x_\omega) = x_c(\omega)$ else.

We cannot use ρ instead of ρ', since some of the local variables might become global variables during the induction. The base case is already given (for $T_\delta(\pi_0)$ there is no difference between ρ and ρ'). For the induction step, suppose $\rho'(T_\delta(\pi)) = \rho'(T_\delta(c(\pi)))$. It follows immediately that $\rho'(T_\delta(\pi \cdot i)) = \rho'(T_\delta(\pi) \cdot i) = \rho'(T_\delta(c(\pi \cdot i)))$ for all $\pi \cdot i \in \mathsf{b}^-(\delta)$.

Now we use 2.1 to prove that c in fact is a correspondence function, i.e. we have to check the following properties of Definition 6:

1. It is clear that c maps the root of δ to the root of δ'.
2. The i-th hole in δ is mapped to the i-th hole in δ': Let π_i be in $\{\pi_1, \dots, \pi_n\}$. We know that $\rho'(T_\delta(\pi_i)) = \rho'(\bullet) = \bullet = \rho'(T_\delta(c(\pi_i)))$ by 2.1. But since c is monotonic with respect to the lexicographic ordering on the nodes in $\mathsf{b}^-(\delta)$ the i-th bullet $\bullet$ in $T_\tau(\delta)$ must correspond to the i-th bullet $\bullet$ in $T_\tau(\delta')$, and this can only be the case if $c(\pi_i) = \pi_i'$.
3. The function c preserves labeling. First, let π be labeled with f^m that is not var. Then we know that $\rho(T_\delta(\pi)) = \rho(f(T_\delta(\pi \cdot 1) \dots T_\delta(\pi \cdot m))) = f(\rho(T_\delta(\pi \cdot 1) \dots \rho(T_\delta(\pi \cdot m)))) =_\alpha \rho(T_{\delta'}(c(\pi)))$ by 2.1. If π is var-labeled we obtain $\rho(T_\delta(\pi)) = \rho(x_\lambda(\pi)) = \rho(T_{\delta'}(c(\pi)))$ also by 2.1, thus $c(\pi)$ must also be var-labeled.
4. The function c also preserves internal binding. Let π be var-labeled, and bound in $\mathsf{b}(\delta)$. By 2.1 we know that $\rho(T_\delta(\lambda(\pi))) =_\alpha \rho(T_{\delta'}(c(\lambda(\pi))))$. Since we are only allowed to consistently rename local variables this equals $\rho(T_{\delta'}(\lambda(c(\pi))))$. Thus $c(\lambda(\pi)) = \lambda(c(\pi))$.

Thus there exists a correspondence function between δ and δ', and we have proven the equivalence. □

Now that we have correspondence functions, it is easy to define the parallelism relation, as it is given in [Egg et al., 2001]. The parallelism relation $\alpha \sim \alpha'$ holds between two tree segments α, α' iff there exist a correspondence function between α and α', and additionally some natural restrictions concerning variable binding hold. These restrictions were developed when modeling linguistic phenomena. This thesis shows that they are again essential if one wants to relate parallelism to the concept of beta reduction, and if we want to formulate underspecified beta reduction.

Definition 7. *Let τ be a λ-structure, and α, α' be two tree segments of the same arity. Then the parallelism relation $\alpha \sim \alpha'$ holds, iff there is a correspondence function c between α and α', such that*

1. *for a* var*-labeled node bound outside a parallel tree segment, the corresponding node is bound at the same place:*

$$\begin{array}{rl} \lambda(\pi) \notin \mathsf{b}(\alpha) \ \Rightarrow \lambda(c(\pi)) = \lambda(\pi) & \qquad \pi \in \mathsf{b}^-(\alpha) \\ \lambda(\pi') \notin \mathsf{b}(\alpha') \ \Rightarrow \lambda(c^{-1}(\pi')) = \lambda(\pi') & \qquad \pi' \in \mathsf{b}^-(\alpha') \end{array}$$

2. *there are no hanging binders:*

$$\begin{array}{rr} \lambda^{-1}(\pi) \subseteq \mathsf{b}^-(\alpha) & \qquad \pi \in \mathsf{b}^-(\alpha) \\ \lambda^{-1}(\pi') \subseteq \mathsf{b}^-(\alpha') & \qquad \pi' \in \mathsf{b}^-(\alpha') \end{array}$$

If $\alpha = \pi/$ and $\alpha' = \pi'/$, i.e. the parallelism relation holds between tree segments without holes, we write $\pi \sim \pi'$ and say that the similarity relation holds between the nodes π and π'.

3 The Constraint Language for Lambda Structures

In the last chapter, we saw lambda structures. Now we will present a constraint language, that is interpreted over lambda structures: CLLS, the constraint language for lambda structures. First, we will introduce its syntax and semantics. Then we will illustrate with more examples from semantic underspecification the applications of parallelism, the most expressive constraint in CLLS.

3.1 Syntax and Semantics of CLLS

We introduce CLLS, the constraint language for lambda structures [Egg et al., 2001, 1998, Erk et al., 2001], which contains literals for dominance, λ-binding constraints and parallelism such that lambda structures can be described partially. We omit anaphoric linking constraints, that are also contained in CLLS to model linguistic phenomena that we do not consider here (they are for instance necessary in Example 1.9 of the introduction).

The symbols X, Y, Z are variables that will denote nodes.

Then the abstract syntax of CLLS-constraints looks as follows:

$$\begin{array}{rcll} \varphi, \psi & ::= & \varphi \wedge \psi \mid \exists X \varphi \mid \textbf{false} & \\ & \mid & X R Y \mid X{:}f(X_1, \ldots, X_n) & (\mathsf{ar}(f) = n) \\ & \mid & X_0/X_1 \ \sim Y_0/Y_1 & \\ & \mid & \lambda(X){=}Y \mid \lambda^{-1}(X_0){=}\{X_1, \ldots, X_n\} & \\ R, R' & ::= & \triangleleft^* \mid \triangleright^* \mid \perp \mid \neq \mid R{\cup}R' \mid R{\cap}R' & \end{array}$$

In this work we could restrict ourselves to fewer types of literals (dominance, disjointness, labeling, parallelism and binding); but for processing, one also needs inequality [Althaus et al., 2001, Koller et al., 2000a], or even set operators for rela-

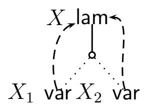

Figure 3.1: The constraint graph of $\lambda^{-1}(X)=\{X_1,X_2\} \wedge X\triangleleft^* X_1 \wedge X\triangleleft^* X_2$

tion descriptors R (see also [Duchier and Niehren, 2000]). We use the abbreviations $X=Y$ and $X\perp Y$ for $X(\{\triangleleft^*\}\cap\{\triangleright^*\})Y$ and $X\{\perp\}Y$.

Note that inverse binding $\lambda^{-1}(X)=\{X_1,\ldots,X_n\}$ makes a stronger statement than a conjunction of single lambda binding constraints $\lambda(X_1)=X\wedge\ldots\wedge\lambda(X_n)=X$ which does not express that *only* the X_i are bound by X. In the application to computational linguistics, and in all examples shown in this paper, we always have an inverse binding literal for every variable for which we have a labeling constraint with lam; that is, we have complete information about variable binding.

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 $\Phi \models\!\!\!\mid \Phi'$ is mutual entailment.

The precise complexity of the satisfiability of CLLS-constraints is still unknown. Since they can express context unification, where the decidability is an open problem, this seems to be a difficult task. But there are various results for several fragments of CLLS. Dominance constraints [Koller et al., 1998, Duchier and Niehren, 2000] comprise only labeling and node relationships, but not binding and parallelism literals. The type of dominance constraints that occurs in applications of semantic underspecification is called normal dominance constraints [Althaus et al., 2001, Koller et al., 2000a]. Pure dominance constraints do not contain labeling literals [Cornell, 1994]. Figure 3.2 shows the complexity of all these constraint languages.

We draw constraints as graphs (Fig. 3.1 and 3.3 on the facing page) where 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 graphically.

The syntactic equivalent of tree segments are *segment terms* $A =$

Fragment	Complexity	Literature
Dominance Constraints without set operators	NP-complete	[Koller et al., 1998]
Dominance Constraints including set operators	NP-complete	[Duchier and Niehren, 2000]
Normal Dominance Constraints	in P	[Althaus et al., 2001, Koller et al., 2000a]
Pure Dominance Constraints (including set operators)	open	[Cornell, 1994]

Figure 3.2: Fragments of CLLS and the complexity of their satisfiability problem

$X_0/X_1,\ldots,X_n$, where the X_i are variables of a constraint φ such that $X_0\triangleleft^* X_i\in\varphi$ for $1\le i\le n$. The root of A is again $r(A)=X_0$, the holes are $hs(A)=X_1,\ldots,X_n$.

The fact that a *segement term* really denotes a tree segment can be axiomatized in CLLS with the following constraint:

$$\mathsf{seg}(A) =_{\mathsf{df}} \bigwedge_{i=1}^{n} X_0\triangleleft^* X_i \;\wedge \bigwedge_{1\le i<j\le n} X_i\{\perp\cup=\}X_j$$

If a constraint Φ entails that the label of some variable X of the constraint equal f^n, one also has syntactic access to the children nodes. By adding $X{:}(X_1,\ldots,X_n)$ with fresh variables $X_1,\ldots,X_n$ to the constraint, we do not change the solutions of Φ. We call the tree segment terms that are 'syntactically reachable' in this sense *fragments* of the constraint graph. Every variable in a constraint Φ is in some fragment. Note that the same argument does not work for dominance constraints $X\triangleleft^* X'$, since X' may not exist uniquely. It is very easy to find the fragments in a constraint graph: They are just the parts connected by solid lines.

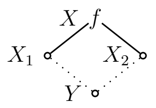

Figure 3.3: The unsatisfiable constraint graph of $X{:}f(X_1,X_2) \wedge X_1\triangleleft^* Y \wedge X_2\triangleleft^* Y$

3.2 Applications of Parallelism Constraints

Another nice features of this approach to underspecification is that it also models the interaction between scope ambiguities and natural language phenomena like ellipsis.

We will show a simple elliptic example to explain the basic idea of how to treat this in CLLS.

Peter sleeps. Marc does too.

In the second (*target*) sentence, the actual activity of Marc is not mentioned explicitly, but it is referred to the first (*source*) sentence. Thus we can construct the semantics of the target sentence if we use the semantics of the source sentence: *Peter sleeps and Marc sleeps*. In CLLS this can be done using parallelism constraints:

$$\begin{aligned} &X_0{:}@(X_1, X_2) \wedge X_1{:}\mathsf{sleep} \wedge X_2{:}\mathsf{peter} \\ &\wedge X_0/X_2 \sim Y_0/Y_2 \wedge Y_2{:}\mathsf{marc} \end{aligned}$$

A famous class of examples where ellipsis and scope ambiguities interact are the so called *Hirschbühler* sentences [Hirschbühler, 1982].

Every man loves a woman. Several gorillas do, too.

The main observation there is that a scope ambiguity in the target and the source sentence of an ellipsis do not disambiguate independently, but they behave *parallel*. This was historically the reason to take parallelism constraints into CLLS. If we describe the semantics of the target sentence by referring to the source sentence via a parallelism constraint, and if after some time you hear the sentence: *Her name was Mary*, we do not only know the reading of the source sentence, namely that every man loves Mary, but we also know that there is only one woman that every gorilla loves.

3.3 Summary of Notation

In the last two chapters we introduced lambda terms, lambda structures and the constraint language over lambda structures. Thus we accumulated quite a bit of notation, and want to give a short overview over the used symbols:

- f, g are the elements of Σ.
- $x, y, \ldots$ stand for the variables in λ-terms.
- $A, B, C, \ldots$ are contexts (λ-terms).
- We use $\tau_1, \tau_2, \ldots$ for lambda structures, and
- π, ω for the nodes in lambda structures.
- $a, b, \ldots$ are the elements of Σ of arity 0.
- γ, β, α denote tree segments.
- $c_1, c_2, \ldots$ are correspondence functions.
- Φ, Ψ stand for constraints in CLLS.
- $X, Y, Z, \ldots$ are the variables in the constraint language CLLS.
- $A, B, C, \ldots$ also denote tree segment terms. There is little danger to mix them with lambda terms, since the latter are a semantic notion and occur in other contexts.
- σ is the assignment function that assigns nodes to constraint variables.

When drawing constraints, we use the following convention:

- Dashed arrows indicate the binding relation.
- Solid lines stand for the parent-child relation.
- Dotted lines stand for dominance.

4 Beta Reduction Constraints

In this chapter we will introduce beta reduction constraints. Very briefly, the beta reduction relation holds between two nodes π and π' in a lambda structure, iff the lambda term starting at π' is the beta reduct of the lambda term starting at π.

We will define this formally using correspondence functions. Then we prove that the definition in fact does what is known from the classical beta reduction step on lambda terms.

Next, we extend the definition of parallelism, such that it is expressive enough to deal with lambda binding in a flexible way. Especially, group parallelism constraints can describe the beta reduction constraint. This is important, since existing procedures for CLLS can be extended such that they can deal with group parallelisms.

4.1 The Beta Reduction Relation

In this section, we add the *β-reduction relation* to lambda structures and then extend the constraint language with *β-reduction constraints* to talk about it. The β-reduction relation on nodes of a lambda structure corresponds exactly to traditional beta reduction on lambda terms. This will be shown formally in section 4.3 on page 38.

We define the β-reduction relation on λ-structures to be a relation between nodes in the *same* λ-structure (see Figure 4.1). 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 figure 4.1 on the following page, these are the segments rooted by $r(\gamma)$ and $r(\gamma')$ respectively.

A *redex* in a lambda structure is a sequence of segments (γ, β, α) in this lambda

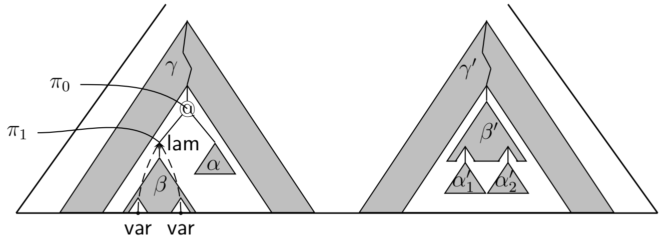

Figure 4.1: The one step beta reduction relation for $n = 2$.

structure that are connected by nodes π_0, π_1 with the following properties.

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

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 Figure 4.1 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 8 (Beta Reduction). 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 at positions 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 c between δ, δ' among these functions and each $\pi \in \mathsf{b}^-(\delta)$, the following conditions hold:

1. if π is bound in the context then the corresponding node is bound by a binder node that c_γ-corresponds to the binder of π.
$$\lambda(\pi) \in \mathsf{b}^-(\gamma) \Rightarrow \lambda(c(\pi)) = c_\gamma(\lambda(\pi))$$
2. corresponding var-labeled nodes with binders outside the group segments are bound by the same binder node:
$$\lambda(\pi) \notin \mathsf{b}(\pi_r/) \Rightarrow \lambda(c(\pi)) = \lambda(\pi)$$

Capturing in β-reduction on λ-terms in classical λ-calculus is avoided by a freeness condition, as we have seen it in section 2.1 on page 19. The following proposition states that the analogous problem can never arise with the β-reduction relation on λ-structures.

Proposition 1 (No Capturing). *Global variables in the argument are never captured by a λ-binder in the body: with the notation of Definition 8 this is 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 the binding property of the correspondence function, in the second property 1 on the facing page and in the third case property 2 on the preceding page of the beta reduction definition leads to a contradiction. □

The β-reduction relation conservatively extends λ-structures. We extend our constraint syntax similarly by adding *β-reduction literals*

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

that are interpreted by the β-reduction relation. We call a CLLS-constraint that contains β-reduction literals, but not contains parallelism literals a *beta reduction constraint.*

4.2 Examples

Figure 4.2 on the following page shows an example of a λ-structure containing both a reducing tree below π_0 and its reduct below π'_1, and thus the beta reduction relation holds on the corresponding nodes:

$$(\pi_0/\pi_1\ , \pi_4/\pi_6\ , \pi_6/)\ \rightarrow_{\beta}\ (\pi'_0/\pi'_1\ , \pi'_1/\pi'_6\ , \pi'_6/)$$

Note the different cases the variables are bound with respect to the reducing tree at π_0: There are global variables bound above π_0 and π'_0, a variable bound at π_1 in the context and the redex variable at π_4.

All the global variables are bound at the same place. The correspondent of the node bound in the context is bound at the corresponding place. Thus, the binding properties for the beta reduction relation holds.

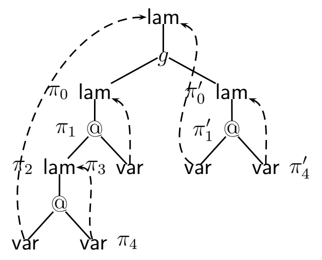

Figure 4.2: The subterm $\lambda y.(@(\lambda x.@(z\ \ x)\ \ y)$ beta reduces to the subterm $\lambda y.@(z\ y)$.

Lets again consider the constraint example in fig. 1.5 on page 16, where rewriting the constraint graph to obtain a description of the reducts was not possible. If you allow yourself beta reduction constraints, such a description is trivially possible (see fig. 4.3).

The submodels of this constraint starting at the node denoted by R' are exactly the reducts of the submodels below the node denoted by R. This will be a consequence of Theorem 1 in the next chapter.

4.3 Soundness of the Definition

We would like to show that the definition of the beta reduction relation does exactly what is known from the beta reduction step in classical lambda calculus. In particular, we want to show the *correctness* of the definition of β-reduction on λ-structures: If a λ-structure satisfies the beta reduction relation between π and π', the λ-term starting at π' is the β-reduct of the λ-term starting at π in the classical

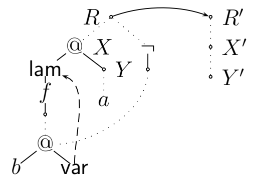

Figure 4.3: Describing the reducts with a beta reduction constraint

sense.

The definition should also be *complete* in the following sense. Suppose, we are given two λ-terms τ_1 and τ_2, such that $\tau_1 \rightarrow_\beta \tau_2$. Consider a λ-structure containing the two λ-terms. Note that such a λ-structure can always be found! Then, the β-reduction relation on λ-structure holds between the respective positions.

These two properties, correctness and completeness, are implied by the following theorem:

Theorem 1 (Soundness). *Let τ be a λ-structure, π and π' two positions in this structure, τ_1 and τ_2 two λ-terms, and ρ a bijective renaming of the global variables such that $\rho(\tau_1) =_\alpha T_\tau(\pi/)$ and $\rho(\tau_2) =_\alpha T_\tau(\pi'/)$, then the following equivalence holds:*

$$\begin{aligned}\tau_1 \rightarrow_\beta \tau_2 \quad \Leftrightarrow \quad & \exists \gamma, \beta, \alpha, \gamma', \beta', \alpha_1', \ldots, \alpha_n' : \quad r(\gamma) = \pi \wedge r(\gamma') = \pi' \\ & \wedge\ (\gamma, \beta, \alpha) \rightarrow^\beta (\gamma', \beta', \alpha_1', \ldots, \alpha_n')\end{aligned}$$

Before we prove this theorem, we would explain why we need to rename the global variables with ρ. This is a technical detail. When reading off λ-terms from tree segments, the function T of Definition 5 on page 24 might choose names for the global variables that are different from the names of the global variables in τ_1 and τ_2.

Proof. First, assume that the right side holds. We have to show that $\tau_1 \rightarrow_\beta \tau_2$. The beta reduction relation implies that the tree below π is of the form $C(@(\lambda x_{\pi_l}.B(x_{\pi_l} \ldots x_{\pi_l})\ A))$. The correspondence relations $c_\gamma, c_\beta, c_\alpha^i$, $1 \leq i \leq n$ and Lemma 1 on page 25 imply that the tree below τ_2 is $C'(B'(A_1', \ldots, A_n'))$, where C', B' and $A_1', \ldots A_n'$ look like their unprimed counterparts, except that the symbol x_π is replaced by $x_{c_\gamma(\pi)}, x_{c_\beta(\pi)}$ and $x_{c_\alpha^1(\pi)}, \ldots, x_{c_\alpha^n(\pi)}$, respectively. It remains to be shown that binding is correct:

- the global variables in τ_2 have the same names as the corresponding global variables in τ_1.
- that internally bound variables in τ_1 and τ_2 are bound correspondingly.

Let x be a global variable in τ_1, and ω its position in τ. We know that $\rho(x) = T_\tau(\omega)$. Let δ be one on the tree segments $\gamma, \beta, \alpha_1 \ldots \alpha_n$. Then by Property 2 of Definition 8 on page 36, $T_\tau(\omega) = T_\tau(c_\delta(\omega))$. Thus we have shown that corresponding global variables in τ and τ' indeed have the same name.

Let x be a locally bound variable of τ_1, except the variable of the redex. Let ω_0 be the position of the binder in the corresponding λ-structure τ_1, and $\omega_1, \ldots, \omega_m$ the positions of the occurrences of the variable. Thus, $\lambda^{-1}(\omega_0) = \{\omega_1, \ldots, \omega_m\}$. Let ω be one of these variable occurrences, i.e. $\omega \in \{\omega_1, \ldots, \omega_m\}$, and δ its tree segment among $\gamma, \beta, \alpha_1, \ldots, \alpha_n$. If $\omega \in \mathsf{b}^-(\gamma)$ then $c_\delta(\omega_i)$ is bound at $c_\gamma(\omega_0)$ by property 1 on page 36. Else, if $\omega \notin \mathsf{b}^-(\gamma)$, by Lemma 1 on page 37 the only remaining possibility is that ω is also bound in δ. In this case $c_\delta(\omega_i)$ is bound at $c_\delta(\omega_0)$. Therefore we obtain that the nodes in τ_2 corresponding to some occurrence of x in τ_1 are bound correspondingly.

For the other direction, assume that $\tau_1 \rightarrow_\beta \tau_2$. We have to check the β-reduction relation between π and π'. It is easy to see that there must be a redex $\mathsf{redex}_{\pi,\pi_0}(\gamma, \beta, \alpha)$ in τ. Now we have to give correspondence functions $c_\gamma, c_\beta, c_\alpha^1, \ldots, c_\alpha^n$ that satisfy Properties 2 and 1 on page 36 of the β-reduction definition.

Since τ_1 reduces to τ_2, we know that τ_1 is of the form $C(@(\lambda x_{\pi_l}.B(x_{\pi_l} \ldots x_{\pi_l})\ A))$ and τ_2 of the form $C(B(A_1 \ldots A_n))$ modulo α-renaming. By Lemma 1 we know that there exist correspondence functions $c_\gamma : \mathsf{b}(\gamma) \rightarrow \mathsf{b}(\gamma')$, $c_\beta : \mathsf{b}(\beta) \rightarrow \mathsf{b}(\beta')$, $c_\alpha^i : \mathsf{b}(\alpha_i) \rightarrow \mathsf{b}(\alpha_i')$ for $1 \leq i \leq n$.

Since the renaming of the global variables of τ_1 and τ_2 is bijective, and the global variables of τ_1 and the corresponding global variables of τ_2 are named the same, we have shown that for any var-labeled node bound outside of τ_1 the corresponding node in C' is bound at the same place. Thus we also have proven Property 2.

To check 1, let ω be a var-labeled node in $\mathsf{b}(\delta)$, where δ is in $\{\gamma, \beta, \alpha_1, \ldots, \alpha_n\}$. Let ω be bound at ω_0 in γ. We have to show that $\lambda(c_\delta(\omega)) = c_\gamma(\omega_0)$. Because of $T_{\tau_1}(\omega) = x_{\omega_0}$ we know that $T_{C(B(A_1 \ldots A_n))}(\omega) = x_{\omega_0}$. Since $\tau_2 =_\alpha C(B(A_1 \ldots A_n))$ it follows that $T_{\tau_2}(c_\delta(\omega)) = x_{c_\gamma(\omega_0)}$ and thus $\lambda(c_\delta(\omega)) = c_\gamma(\omega_0)$. □

4.4 Group Parallelism

In the last section we saw how to describe redexes and reducs in a single lambda structure using beta reduction constraints. However, these constraints remain very implicit, i.e. the structure of the reduct is only described via the beta reduction constraint and not by explicit labeling and dominance constraints.

In this section, we extend CLLS with *group parallelism constraints* (Definition 9), a generalization of the parallelism constraints in CLLS that we already introduced. Then we show that CLLS with group parallelism can express β-reduction constraints (Theorem 2).

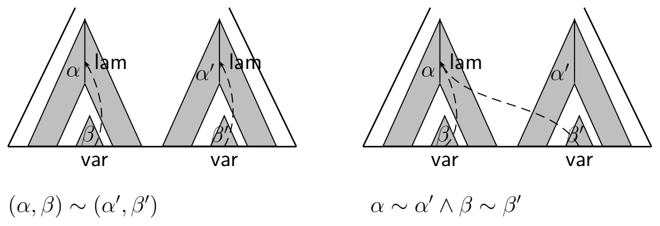

Figure 4.4: Binding in a group parallelism vs. a conjunction of ordinary parallelisms.

Since an existing solution procedure for parallelism constraints [Erk and Niehren, 2000] can be extended to deal with group parallelism, this gives us a procedure that makes beta reduction constraints more explicit.

Both parallelism and group parallelism constraints require the structural similarity of tree segments. A *group* is a sequence of (not necessarily different) tree segments. Group parallelism relates two groups $(\alpha_1, \ldots \alpha_n)$ and $(\alpha_1', \ldots \alpha_n')$ if each pair of segments α_i, α_i' is structurally isomorphic. The only difference between a group parallelism and a conjunction of ordinary parallelisms is the treatment of the binding functions.

If var-labeled nodes in one segment are bound in another tree segment of the group, in a group parallelism the corresponding node is bound by the *corresponding* binder (Figure 4.4). This cannot be expressed by ordinary parallelism constraints for two reasons. Consider the group parallelism $(\alpha, \beta) \sim (\alpha', \beta')$, and compare this to $\alpha \sim \alpha' \wedge \beta \sim \beta'$. Suppose there is a var-labeled node π in α bound in β. This is a hanging binder with respect to $\beta \sim \beta'$. Moreover, Property 2 of the definition of the parallelism relation $\alpha \sim \alpha'$ requires π to be bound at the same place as its correspondent. But in group parallelisms we want the correspondent to be bound by the *corresponding node* of the binder with respect to the parallelism between β and β' (compare Figure 4.4).

Definition 9. *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 for all $1 \leq k \leq n$ that there is a correspondence function $c_k : \mathsf{b}(\alpha_k) \rightarrow \mathsf{b}(\alpha_k')$ satisfying the following properties for all $1 \leq i, j \leq n$ and $\pi \in \mathsf{b}^-(\alpha_i)$:

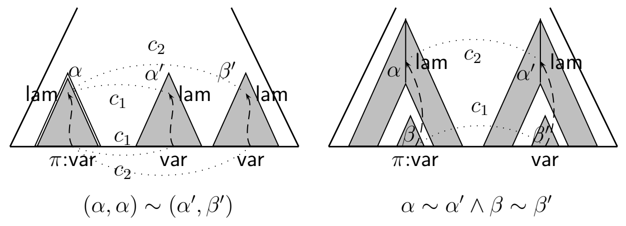

Figure 4.5: Binding into the same segment vs. binding into another group segment.

1. *for a* var*-labeled bound outside α_i but inside α_j, the correspondent is bound at the corresponding place with respect c_j:*

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

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

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

3. *there are no hanging binders:*

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

Such a greatest symmetric relation always exists, since the union of two relations satisfying the above requirements again satisfies them. The parallelism relation (Definition 7 on page 27) is a special case of group parallelism, where $n = 1$.

Note that Condition 1 of Definition 9 must not apply to variables that are bound in the same tree segment, see Figure 4.5. If 1 was applicable in the left picture, it would enforce $\lambda(c_1(\pi)) = c_2(\lambda(\pi))$, which is not the case.

We extend CLLS by group parallelism literals that are interpreted by the group parallelism relation. A *group parallelism literal* has the following form, where $A_1, \dots A_m, A'_1, \dots, A'_m$ are segment terms:

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

A *group parallelism constraint* contains CLLS-constraints and group parallelism literals, but does not contain β-reduction literals. (Similarly, we call a constraint that does not contain parallelism or group parallelism literals a *beta reduction constraint.*)

4.5 Describing Beta Reduction with Group Parallelism

The beta reduction relation on λ-structures basically states that certain parts in a λ-structure look the same, namely the context in the reducing tree to the context in the reduct etc. Both the beta reduction and the parallism relation are formulated using correspondence functions. But the definition of parallelism is too restrictive concerning its binding properties. There might be variables in the body that are bound in the context, so every parallelism literal that only considers the context has hanging binders. Group Parallelism is much more flexible with respect to this.

Although very simple in principle, theorem 2 is one of the key statements of this thesis, since it will enable us to use existing algorithms for parallelism in CLLS to also process β-reduction constraints.

We will now show how to encode beta reduction constraints in CLLS. First, we axiomatize a redex in CLLS. For segment terms $C = X/X_0$, $B = X_3/X_4, \dots, X_n$ and $A = X_2/,$, we set:

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

This reads: the context, body and argument segment terms really denote tree segments; we have an @-labeled node with a λ-labeled node as a first child; we do know all occurrences of the variables bound there.

Next, we define reduct-like groups. Let $C = X/X_0$, $B = X'_0/X'_1, \dots, X'_n$ and $A_i = X_i/$ for $1 \le i \le n$, then we define:

$$\begin{aligned} \mathsf{reductlike}(C,B,A_1,\dots,A_n) \;=_{\mathsf{df}}\; & \mathsf{seg}(A_1) \wedge \cdots \wedge \mathsf{seg}(A_n) \wedge \mathsf{seg}(B) \wedge \mathsf{seg}(C) \\ & \wedge X_0{=}X'_0 \wedge X_1{=}X'_1 \wedge \cdots \wedge X_n{=}X'_n \end{aligned}$$

We already mentioned that not every group that satisfies reductlike is a potential reduct, since we cannot enforce that there is no binder from the argument into the body (capturing, see Lem. 1). But in the following encoding this will be implied by the group parallelism constraints.

Theorem 2. *Beta reduction constraints can be expressed in CLLS with group parallelism via the following equivalence:*

$$\begin{aligned} (C,B,A) \to^{\beta} (C',B',A'_1,\dots,A'_n) \quad \models\!\mid \quad & \exists\, X_0, X_1 : \mathsf{redex}_{X_0,X_1}(C,B,A) \\ & \wedge\; (C,B,A,\dots,A) \sim (C',B',A',\dots,A') \\ & \wedge\; \mathsf{reductlike}(C',B',A'_1,\dots,A'_n) \end{aligned}$$

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. To show that it also solves the β-reduction literal on the left hand side, we define $\alpha = \sigma(A)$, $\alpha'_i = \sigma(A'_i)$, $\beta = \sigma(B)$, $\beta' = \sigma(B')$, $\gamma = \sigma(C)$, and $\gamma' = \sigma(C')$. The tree segments (γ, β, α) form a redex and the group $(\gamma', \beta', \alpha'_1, \ldots, \alpha'_n)$ is $\mathsf{reductlike}$ since this is explicitely stated in the constraint on the right hand side.

Properties 1 and 2 for the beta reduction relation (Definition 8) are then subsumed by the corresponding properties of the group parallelism relation (Definition 9).

For the other direction, let (τ, σ') solve the beta-reduction literal on the left hand side. Let $\gamma, \gamma', \beta, \beta', \alpha$ and α'_i for $1 \leq i \leq n$ be defined as above. According to (Definition 8) the tree segments (α, β, γ) form a reducing tree, where we call the redex position π_0 and binder position π_1.

It remains to check whether (τ, σ) solves the group parallelism literal on the right hand side. We consider the following symmetric relation $\approx$ which relates the group $(\gamma, \beta, \alpha, \ldots, \alpha)$ with $(\gamma', \beta', \alpha'_1, \ldots, \alpha'_n)$ and conversely. We show that $\approx$ satisfies all conditions in the definition of group parallelism (Definition 9) so that $\approx$ is subsumed by the group parallelism relation $\sim$ since the latter is maximal with that property. It then follows that (τ, σ) satisfies $(C, B, A, \ldots, A) \sim (C', B', A'_1, \ldots A'_n)$.

First of all, both above groups satisfy condition 3. This is clear for the group $(\gamma', \beta', \alpha'_i, \ldots, \alpha'_n))$ which covers the complete subtree below $\sigma(r(\gamma))$ by definition of γ', β', and α'_i. A similar argument applies for $(\gamma, \beta, \alpha, \ldots, \alpha)$ which covers the whole tree below $r(\gamma')$ except the @-labeled node π_0, the lam-labeled node π_1 and the var-labeled nodes $hs(\alpha).1, \ldots, hs(\alpha).n$ in the redex. But these variables are bound by π_1.

Second, we have to check properties 1 and 2 for group parallelisms (Definition 9) for appropriate correspondence functions. Let $D =_{\mathsf{df}} \mathsf{b}(\gamma) \cup \mathsf{b}(\beta) \cup \mathsf{b}(\alpha)$ and $D' = \mathsf{b}(r(\gamma')/)$. The semantics of the β-reduction literal implies the existence of suitable correspondence functions c_γ between segments γ, γ', c_β between β, β' and c^i_α between α, α'_i for $1 \leq i \leq n$. Since $\approx$ is symmetric, we also have to check properties 1 and 2 for all inverse correspondence functions $c^{-1} : \delta' \to \delta$:

1. Let π' be a var-labeled node in $\mathsf{b}^-(\delta')$ bound in δ'. The correspondent π of π' must also be var-labeled. Thus, it must be bound somewhere. If it is bound outside of all group segments, by property 2, we have $c(\lambda(\pi)) \notin D'$, a contradiction to the above assumption. If it is bound in another segment, which in our case is only possible if π lies in α or in β, and $\lambda(\pi)$ is in $\mathsf{b}^-(\gamma)$, the correspondent of the binder has to be bound in γ'. Then we also derive a contradiction since γ' is strictly above β' and α'_i. The last case, π is bound in $\mathsf{b}^-(\delta)$, implies $\lambda(c(\pi)) = c(\lambda(\pi))$. Hence $c^{-1}(\lambda(\pi')) = c^{-1}(c(\lambda(\pi))) = \lambda(c^{-1}(\pi'))$ and we are done.

2. Let π' be a var-labeled node in $\mathsf{b}(\delta')$, bound in another group segment, which is not δ'. This implies that δ is either β' or α'_i: Since, if δ was γ, it cannot be bound in another group segment, since the others are all below γ. There are three cases to be considered: $\lambda(\pi') \in \mathsf{b}^-(\beta')$ or $\lambda(\pi') \in \mathsf{b}^-(\alpha'_j)$ for some $1 \leq j \leq n$ or $\lambda(\pi') \in \mathsf{b}^-(\gamma')$.

   The first case is impossible: If δ' is α'_j, it is forbidden since we have no hanging binders (Proposition 1). If δ' is β', this contradicts to the fact that the binder has to be in another group segment.

   The second case is also impossible: If δ' is α'_j for some $1 \leq j \leq n$, the holes of the segment β' are disjoint or equal (Definition 4). If δ' is β', the third case is impossible because β' lies above α'_j.

   Thus, we have that $\lambda(\pi') \in \mathsf{b}^-(\gamma')$. Let π be the corresponding node of π'. As π also has to be var-labeled, it must be bound either in $\mathsf{b}^-(\delta)$, $D_\tau - D$ or $\mathsf{b}^-(\gamma)$.

   The first case is not possible, since property (1) of the beta reduction relation implies that π' is bound in δ', which is located strictly below γ'.

   The second case is also not possible: Assume π is bound in $D_\tau - D$, i.e. outside the group segments. Be property 2 on page 36 we know that $\lambda(\pi) = \lambda(\pi')$.

   Therefore we know that π lies below $\lambda(\pi')$ and thus must be in a parallel group segment γ', β' or α'_j for some $1 \leq j \leq n$. In every case this contradicts the assumption that π is bound outside all of the group segments.

   So let π be bound in γ. Thus property 1 of the beta reduction relation yields $\lambda(c_\delta(\pi)) = c_\gamma(\lambda(\pi))$ i.e. $c_\gamma^{-1}(\lambda(\pi')) = c_\gamma^{-1}(c_\gamma(\lambda(\pi'))) = \lambda(c_\delta^{-1}(\pi'))$, as required.

3. Let π' be a var-labeled node in $\mathsf{b}(\delta')$, bound outside D'. Again, the correspondent π must be bound somewhere. If it is bound in D, it is bound in some group segment δ_2, and because c_{δ_2} is a correspondence function, $c_{\delta_2}(\pi') \in D'$, a contradiction. Thus, π is bound outside of D. Property 2 of the beta reduction relation enforces $\lambda(\pi) = \lambda(\pi')$.

□

5 Expressiveness of Beta Reduction Constraints

Now that we have defined the beta reduction constraint, we are interested in its expressive power. It is clear that the beta reduction constraint can express structural identity between certain tree segments. We would like to know whether they can express parallelism constraints with all its binding properties.

The encodings given in this chapter do not only preserve satisfiability, but they in principle preserve the solutions. We will define a appropriate notion of expressiveness of constraint languages, and we express a relevant fragment of parallelism constraints with beta reduction constraints and prove the correctness of the encodings.

Since we will use group parallelism constraints in the next chapter to also solve beta reduction constraints, one could ask whether using group parallelism constraints is an overhead for beta reduction constraints. Already parallelism constraints are equally expressive to context unification. But satisfiability of a context unification problem is not known to be decidable [RTA-List, 1998, Lévy, 1996, Schmidt-Schauß and Schulz, 1999].

Since beta reduction constraints can encode a large fragment of parallelism constraints, the results of this section show that it is likely that parallelism constraints are not an overhead for the treatment of beta reduction constraints.

5.1 Expressiveness of Constraint Languages

We want to express parallelism. One cannot expect for every parallelism constraint an equivalent beta reduction constraint, since the solution of a parallelism constraint might not contain any @-labeled nodes, but you enforce them when using beta reduction constraints. But we still want to have a notion of expressiveness, which is stronger than equivalence with respect to satisfiability. The solutions should preserve the structure of the solutions of the original constraint, more precisely, they should contain the old solution as a *subsolution*.

First, we will define submodels and subsolutions.

Definition 10 (Submodels). *Let $\tau_1 = (\theta_1, \lambda_1)$ and $\tau_2 = (\theta_2, \lambda_2)$ be λ-structures. Then τ_1 is a* submodel *of τ_2, iff the underlying tree θ_1 of τ_1 is a subtree of θ_2, and the binding function coincides on D_{τ_1}:*

$$\tau_1 \prec \tau_2 \Leftrightarrow_{\mathsf{df}} \exists \pi : \tau_2.\pi = \tau_1 \ \wedge \ \lambda_2|_{D_{\theta_1}} = \lambda_1$$

A solution (τ, σ) of a constraint Φ is a subsolution of a solution (τ', σ') of a constraint Φ' where Φ' contains at least the variables in Φ, if τ is a submodel of τ', the interpretations coincide on the common variables and the new variables are interpreted in the new parts of the model.

Definition 11 (Subsolutions). *Let Φ, Φ' be constraints such that $\mathcal{V}(\Phi) \subseteq \mathcal{V}(\Phi')$, let τ, τ' be λ-structures, and $\sigma : \mathcal{V}(\Phi) \to \tau$ and $\sigma' : \mathcal{V}(\Phi') \to \tau'$ be variable assignments. Then we define*

$$(\tau, \sigma) \prec (\tau', \sigma')$$

if and only if

1. *The model of the first solution is a submodel of the second one:*
$$\tau \prec \tau'$$
2. *The second variable assignment equals the first one, if we restrict it to the variables of the first constraint:*
$$\sigma'|_{\mathcal{V}(\Phi)} = \sigma$$
3. *The additional variables in the second constraint are not mapped into the submodel τ of τ':*
$$\sigma'(\mathcal{V}(\Phi') - \mathcal{V}(\Phi)) \cap \tau = \emptyset$$

In this case we say that (τ, σ) is a subsolution of (τ', σ'), and conversely that (τ', σ') is a supsolution of (τ, σ).

Note that the subsolution relation is transitive and reflexive.

The least one would expect when defining whether a constraint Φ is expressed by a constraint Φ' is that Φ' should entail Φ, and that Φ' should be satisfiable if and only if Φ is satisfiable. But in the reductions that follow in the next chapters, we have even stronger properties. The solutions of Φ are subsolutions of solutions in Φ'. Moreover, for every solution of Φ' we can find a solution of Φ that is subsolution.

Definition 12 (Expressiveness). *A constraint Φ is expressed by a constraint Φ', if*

1. *for every solution (τ', σ') of Φ' there is a solution (τ, σ) of Φ such that*
$$(\tau, \sigma) \prec (\tau', \sigma')$$
2. *for every solution (τ, σ) of Φ there is a solution (τ', σ') of Φ' such that*
$$(\tau, \sigma) \prec (\tau', \sigma')$$

Suppose we want to show that for every formula Φ containing a certain type of literal we can find a formula Φ' not containing this type of literal. Because of the transitivity of the subsolution it suffices to show that one of these literals can be expressed. By induction, we know that we can eliminate all of these literals one by one.

5.2 Expressing Similarity

The similarity relation $\pi \sim \pi'$ was defined in Section 2.4 on page 24 as a parallelism relation between the tree segments $\pi/$ and $\pi'/$ that have no holes.

We also write $X \sim X'$ instead of $X/ \sim X'/$ and these special form of parallelism literals we call *similarity* literals. *Similarity Constraints* are parallelism constraints where the only parallelism literals are similarity literals.

In this section we will show how to express similarity constraints with beta reduction constraints. This will serve as an example for an encoding using the notion of expressiveness of Definition 12. Additionally, will will in turn use a similarity literal when encoding parallelism constraints with β-reduction constraints in Section 5.3.

Theorem 3. *For every similarity constraint Φ, there is a β-reduction constraint Φ' that expresses Φ'.*

Proof. As we mentioned in the last chapter, it suffices to express a single similarity literal $X_1 \sim X_2$. The idea is to make use of the capability of beta reduction to *copy* its argument. The two segments of the tree whose similarity has to be expressed will play the rôle of the two occurrences of an argument.

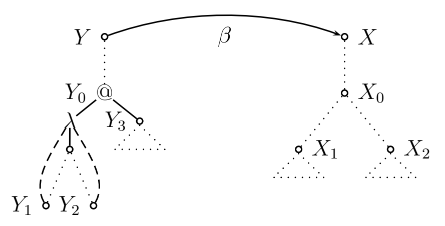

Figure 5.1: Encoding of Similarity.

The encoding looks as follows:

$$\begin{aligned}\Phi' =_{\mathsf{df}} \exists\, Y, Y_0, Y_1, Y_2, Y_3, X, X_0 :&\\ &((Y/Y_0\,), (Y_0/Y_1, Y_2\,), Y_3/) \rightarrow^{\beta} ((X/X_0\,), (X_0/X_1, X_2\,), X_1/, X_2/)\\ &\wedge\ X \bot Y\end{aligned}$$

We start proving Property 2 of Definition 12 on the page before. Let (τ, σ) be a solution of $X_1 \sim X_2$. We have to construct a solution (τ', σ') of Φ', such that $(\tau, \sigma) \prec (\tau', \sigma')$.

Let $\sigma(X_1)$ be π_1 and $\sigma(X_2)$ be π_2. We call π_0 the greatest common ancestor of π_1 and π_2, i.e. π_0 is the maximal path such that $\pi_0 \lhd^* \pi_1$ and $\pi_0 \lhd^* \pi_1$. Since $\pi_1 \sim \pi_2$, we have no hanging binders in the tree segment $\pi_0/\pi_1, \pi_2$. We can now define

$$\begin{aligned} \gamma &:= \pi/\pi_0 & C &= T_\tau(\gamma)\\ \beta &:= \pi_0/\pi_1, \pi_2 & B &= T_\tau(\beta)\\ \alpha_1 &:= \pi_1/ & A_1 &= T_\tau(\alpha_1)\\ \alpha_2 &:= \pi_2/ & A_2 &= T_\tau(\alpha_2)\end{aligned}$$

Clearly, $\tau = C(B(A_1\ A_2)$. We now define $\tau' = f(C(@(\lambda x.B(x\ x)\ A_1)\ \tau)$. The choice of A_1 is arbitrary, since A_1 equals A_2 modulo α-renaming. There are no free variables in this λ-term, since τ has no free variables and the variables in A_1, A_2 and B are all bound in C. So we can identify τ' with its λ-structure, which will serve us as a model for Φ'. To see this, extend σ to σ' by interpreting X with $\epsilon{\cdot}2$, Y with $\epsilon{\cdot}1$ and X_0 by π_0. Y_0, Y_1 and Y_2 are interpreted by the corresponding positions of π_0, and $\sigma'(Y_3) = \sigma'(X_0){\cdot}2$. Using the soundness of the definition of beta reduction it is obvious that (τ', σ') satisfies Φ'. Moreover, it is clear that (τ', σ') is a subsolution of (τ, σ), since the new variables are all interpreted over the first branch in the model, and the old variables are interpreted as in σ.

For Property 1 of Definition 1, let (τ', σ') be a solution of Φ'. We will show that the solution $(\tau, \sigma) = (\tau'|_{\sigma'(X)}, \sigma'|_{\{X_1, X_2\}})$ satisfies $X_1 \sim X_2$. Because of the disjointness literal in Φ' it is clear that (τ, σ) is a subsolution of (τ', σ').

By definition of the β-reduction constraint, there exist π'_0, π'_a in τ' such that $\pi'_0 : @(\pi'_0{\cdot}1, \pi'_a)$ and correspondence functions c_1 between $\pi'_a/$ and $\pi_1/$ and c_2 between $\pi'_a/$ and $\pi_2/$. The composition $c =_{\mathsf{df}} c_1^{-1} \circ c_2$ between these function is again a correspondence function. We show that c satisfies all the properties given in the group parallelism definition. Since we only look at the restricted case of similarity, it suffices to check the properties of internal and external binders for c and for its inverse c^{-1}. Let ω_1 be a var-labeled node in $\mathsf{b}(\pi_1/)$, and ω_2 its correspondent in $\mathsf{b}(\pi_2/)$, and c_γ be the correspondence functions between $\sigma(Y)/\sigma(Y_0)$ and $\sigma(X)/\sigma(X_0)$.

1. Let ω_1 be bound in $\mathsf{b}(\pi_1/)$. Thus, $\lambda(c_1^{-1}(\omega_1)) \in \mathsf{b}(\pi'_a/)$ and we have $\lambda(c_2(c_1^{-1}(\pi_1)) = \lambda(c(\omega_1)) = \lambda(\omega_2) \in \mathsf{b}(\pi_2/)$, as required.
2. Let ω_1 be bound above $\mathsf{b}(\pi_1/)$. In this case we have to show that $\lambda(\omega_1) = \lambda(\omega_2))$. Depending on the location of the binder of ω_1 we distinguish three cases (see Figure 5.1 on the facing page):
   a) The binder of ω_1 is above $\sigma(X)$:
   By Property 2 on page 36 of the beta reduction relation, $\lambda(\omega_1) = \lambda(c_1^{-1}(\omega_1)) = \lambda(c_2(c_1^{-1}(\omega_1))) = \lambda(\omega_2)$.
   b) The binder of ω_1 is bound between $\sigma(X)$ and $\sigma(X_0)$:
   i.e. $\sigma(X) \lhd^* \lambda(\omega_1) \lhd^+ \sigma(X_0)$. If $c_1^{-1}(\omega_1)$ would not be bound at $c_\gamma^{1-}(\lambda(\omega_1))$, we have a contradiction by Property 1 of the beta reduction relation. Again by Property 1 we conclude that $c_2(c_1^{-1}(\omega_1)$ is bound at $c_\gamma(c_\gamma^{-1}(\lambda(\omega_1))$ and thus, $\lambda(\omega_2) = \lambda(\omega_1)$.
   c) The binder of π_1 is bound between $\sigma(X_0)$ and $\sigma(X_1)$. But this can never happen, since we do not have capturing of argument variables (Proposition 1 on page 37).

Because the construction is symmetric, the proof for the inverse c^{-1} is the same. We have thus proven that (τ', σ') satisfies the similarity literal, which concludes the proof. □

5.3 Expressing non overlapping Parallelism

We now present a reduction of parallelism constraints, that are non overlapping, i.e. for all solutions (τ, σ) of a constraint containing the parallelism literal $A \sim B$, it holds that $\mathsf{b}^-(\sigma(A)) \cap \mathsf{b}^-(\sigma(B)) = \emptyset$ or $\sigma(A) = \sigma(B)$. This is interesting, since non overlapping beta reduction constraints can in turn be encoded by a non overlapping group parallelism.

Definition 13. *Let τ be a λ-structure, and α, α' two tree segments in τ. Then α and α' are called non overlapping, iff*

$$\mathsf{b}^-(\alpha) \cap \mathsf{b}^-(\alpha') = \emptyset, \ \textit{or } \alpha = \alpha'$$

For a given constraint Φ two segment terms A and B are non overlapping, iff $\sigma(A)$ and $\sigma(B)$ are non overlapping for all solutions (σ, τ) of Φ.

We start with two lemmas that will be extensively used later on:

Lemma 2. *Let Φ be an arbitrary satisfiable constraint, then for X_0, X_1 in $\mathcal{V}(\Phi)$ such that $\sigma(X_0/X_1\ )$ does not have any hanging binders for every solution (τ, σ) of Φ the following constraint is also satisfiable:*

$$\exists\ X, C', B', A' : (C', B', A') \rightarrow^\beta (X/X_0\ , X_0/X_1\ , X_1/)$$

If for every model of Φ there is a tree segment $\beta = \pi_0/\pi_1$ that does not have any hanging binders, the corresponding existential constraint is satisfiable:

$$\exists\ X, X_0, X_1, C', B', A' : (C', B', A') \rightarrow^\beta (X/X_0\ , X_0/X_1\ , X_1/)$$

Proof. Since Φ is satisfiable, there is a solution (τ, σ) of Φ, where σ assigns $\beta = \pi_0/\pi_1$ to B. We define $\gamma =_{\mathsf{df}} \epsilon/\pi_0$ and $\alpha =_{\mathsf{df}} \pi_1/$, and claim that the lambda structure of the following closed term

$$\tau' =_{\mathsf{df}} f(T_\tau(\gamma)\ @(\lambda x.T_\tau(\beta)(x)\ T_\tau(\alpha))$$

is a model for the given constraint, provided a two-ary function symbol f in our signature, and a fresh variable name x. To see this, let σ' be an extension assigning $\epsilon/1{\cdot}\pi_0$ to C', $1{\cdot}\pi_0/1{\cdot}\pi_1$ to B', $1{\cdot}\pi_1/$ to A' and $2{\cdot}\sigma(Z)$ to all other variables $Z \in V$. □

Lemma 3. *Let Φ be an arbitrary satisfiable constraint containing a redex $\mathsf{redex}^1_{X_0,X_1}(C, B, A)$ then the following constraint is also satisfiable:*

$$\exists C', A', B' : (C, B, A) \rightarrow^\beta (C', B', A')$$

This lemma is proven analogously by constructing a model.

Theorem 4. *Let Φ be a parallelism literal with only non overlapping parallelism literals, i.e. if $A \sim B$ then A and B are non overlapping. Then there is a β-reduction constraint expressing Φ.*

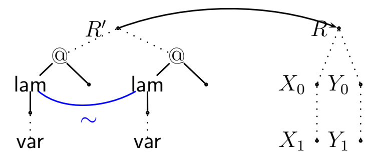

Figure 5.2: Idea of the encoding using a parallel reduction.

We will again give a encoding of a parallelism literal (5.4 on page 55 and 5.5 on page 58). But before we prove the correctness of this encoding we would like to explain the ideas – and problems of the encoding.

The main idea is to employ similarity, that we already can express by beta reduction constraints. To express a parallelism literal we use two redexes in the encoding that have potentially different arguments but impose a similarity constraint on their bodies (see Figure 5.2). After beta reducing at the two redexes, the different arguments are placed under the necessarily identical bodies, and we have thus expressed a parallelism up to one hole.

To make this idea work, we have to express a *parallel beta reduction*, to describe the result after reducing at these two redexes. The concept of parallel beta reduction steps has already been used in classical lambda calculus, for instance when studying confluence. The idea is to simultaneously perform beta reduction steps at several disjoint positions in the tree.

There is a problem with the naïve approach, which is to describe the reduct and from there on to describe the reduct of the reduct. If we describe the reduct we do not have access to the positions in the constraint that correspond to the second redex. So we have no variables for the positions in the reduct to write down the second beta reduction constraint, see Figure 5.3.

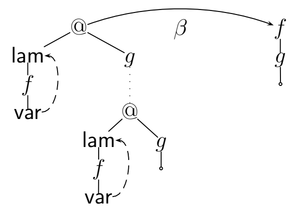

Figure 5.3: It is not possible to add a parallelism constraint that describes the reduct of the reduct.

$$\begin{array}{lll}
\exists X_1^1,\dots,X_{11}^5,Y_2^1,\dots,Y_{10}^5 : & & \\
& \wedge\, (X_7,X_7^2,X_7^3,X_7^4) \rightarrow_\beta (X_6,X_6,X_6^1) & \wedge\, (X_6,X_6^0,X_6^1,X_6^2) \rightarrow_\beta (X_5,X_5^0,X_5^0) \\
& \wedge\, (X_5,X_5^0,X_5^1,X_5^2) \rightarrow_\beta (X_4,X_4^0,X_4^1) & \wedge\, (X_4,X_4,X_4^1,X_4^1) \rightarrow_\beta (X_3,X_3,X_3^1) \\
& \wedge\, (X_3,X_3^0,X_3^1,X_3^2) \rightarrow_\beta (X_2,X_2^0,X_2^0) & \wedge\, (X_2,X_2^0,X_2^1,X_2^2) \rightarrow_\beta (X_1,X,X') \\
& \wedge\, (X_7,X_7,X_7^1,X_7^2) \rightarrow_\beta (Y_6,Y_6,Y_6^1) & \wedge\, (Y_6,Y_6^0,Y_6^1,Y_6^2) \rightarrow_\beta (Y_5,Y_5^0,Y_5^0) \\
& \wedge\, (Y_5,Y_5^0,Y_5^1,Y_5^2) \rightarrow_\beta (Y_4,Y_4^0,Y_4^1) & \wedge\, (Y_4,Y_4,Y_4^1,Y_4^1) \rightarrow_\beta (Y_3,Y_3,Y_3^1) \\
& \wedge\, (Y_3,Y_3^0,Y_3^1,Y_3^2) \rightarrow_\beta (Y_2,Y_2^0,Y_2^0) & \wedge\, (Y_2,Y_2^0,Y_2^1,Y_2^2) \rightarrow_\beta (X_1,Y,Y') \\
& \wedge\, (X_7,X_7^2,X_7^3,X_7^4) \rightarrow_\beta (X_8,X_8^0,X_8^2) & \wedge\, (X_8,X_8^1,X_8^2,X_8^3) \rightarrow_\beta (X_9,X_9^0,X_9^0) \\
& \wedge\, (X_9,X_9,X_9^1,X_9^2) \rightarrow_\beta (X_{10},X_{10},X_{10}^1 & \wedge\, (X_{10},X_{10}^0,X_{10}^1,X_{10}^2) \rightarrow_\beta (X_{11},X_{11}^0,X_{11}^0) \\
& \wedge\, (X_7,X_7,X_7^1,X_7^2) \rightarrow_\beta (Y_8,Y_8^0,Y_8^2) & \wedge\, (Y_8,Y_8^1,Y_8^2,Y_8^3) \rightarrow_\beta (Y_9,Y_9^0,Y_9^0) \\
& \wedge\, (Y_9,Y_9,Y_9^1,Y_9^2) \rightarrow_\beta (Y_{10},Y_{10},Y_{10}^1 & \wedge\, (Y_{10},Y_{10}^0,Y_{10}^1,Y_{10}^2) \rightarrow_\beta (X_{11},Y_{11}^0,Y_{11}^0) \\
& \wedge\, X_{11}^0{:}@(X_{11}^1,X_{11}^2) \ \wedge\ Y_{11}^0{:}@(Y_{11}^1,Y_{11}^2) & \wedge\, X_{11}^1 \sim Y_{11}^1 \\
& \wedge\, \bigwedge_{i=1}^{11} X_1 \perp X_i & \wedge\, \bigwedge_{i=2}^{10} X_1 \perp Y_i
\end{array}$$

Figure 5.4: The encoding Φ of $X/X' \sim Y/Y'$. See also 5.5.

The problem with the previous example is that the only tree segments that are accessible via some fragment do not necessarily contain the tree segment of the second redex in the reduct, and thus we do not have access to the positions where we want to reduce next.

The idea to overcome this problem is to use the lambda binding constraints. With the help of additional redexes, we can link the different fragments. To implement this we describe a subtree, which reduces in two reduction chains to both R_1 and R_{11}, i.e. to the reducing tree and and the reduct of the parallel reduction (see Figure 5.5 on page 58).

We avoid the problem of the missing access to the positions of the second redex by introducing extra redexes that we use to *activate* certain positions in the tree. These *activators* are applications of identities, which we additionally reduce.

Altogether, we express a disjoint parallelism by the rather complex constraint Φ' presented in figure 5.4 on the facing page. It is visualized by its constraint graph in Figure 5.5 on page 58. We also added some labeling and binding constraints that are entailed by Φ'. Notice how the access to some parts of the described models gets lost on the reduction step from X_4 to X_3, Y_4 to Y_3, X_9 to X_{10} and Y_9 to Y_{10}. Nonetheless, every relevant position in R_1 or R_{11} can be accessed since we have described the submodels there twice.

Proof of Theorem 4. Let Φ' be the constraint drawn in Figure 5.5 on page 58. We first show that for every solution (τ, σ) of the parallelism constraint we can construct a solution (τ', σ') of Φ', such that $(\tau, \sigma) \prec (\tau', \sigma')$.

If we follow the two reduction chains in fig. 5.5 on page 58 from X_1 back to X_7 and then from there there on forwards to X_11, we construct a model for all of the beta reduction constraints, where we use the lemmas 2 on page 52 and 3 on page 52 several times. In a next step, we argue that the similarity constraint $X^0 \sim Y^0$ is satisfied in this model.

We have to carefully argue for every step, why the lemmas are applicable. We will do this first for the reduction chain $X_7 \rightarrow X_6 \rightarrow X_5 \rightarrow X_4 \rightarrow X_3 \rightarrow X_2 \rightarrow X_1$.

The requirements for Lemma 2 on page 52 are given for the tree below X_1 and the tree segment denoted by X/X' , since the parallelism relation explicitly forbids hanging binders. The next step applies this lemma on the empty tree segment, starting and ending at the node corresponding to the redex of the last reduction, denoted by X_2. Thus, hanging binders are also impossible. Then we apply the lemma to the tree segment starting at the node denoted by X_3 ending at the identity function that we applied in the last reduction step. In this case, hanging binders are impossible since the argument contains only one variable, which is bound internally. To satisfy the beta reduction constraint between X_4 and X_5 we again make use of the fact that the parallelism constraint $X_0/X_1 \sim Y_0/Y_1$ has no hanging binders. But this time we have to use Lemma 2 in its existential form, since we do not have variables in the constraint, that denote the correspondents of Y/Y' below X^4. But these correspondents exists, since we only followed beta reductions with nondisapearing arguments. These correspondents still form a tree segment. This is only the case, because we are encoding non-overlapping parallelism. Obviously, these tree segments are still without hanging binders. The next reduction step again has an empty body, and in the reduction $X_7 \rightarrow X_6$ the argument does only contain one locally bound variable, so that we can apply Lemma 2 again.

The reasoning in the other branch of reductions, $X_7 \rightarrow Y_6 \rightarrow Y_5 \rightarrow Y_4 \rightarrow Y_3 \rightarrow Y_2 \rightarrow X_1$, is similar to the above construction, except the step $X_7 \rightarrow Y_6$, where we *join* the other branch again.

We know from beta reduction on linear lambda terms that the order in which we perform beta reduction steps doesn't matter. Since the arguments are only used once, no redex is copied, and all reduction sequences have linear length. Moreover, no matter in which order all the reductions are made, we always arrive at the same resulting term. We use this fact to prove the beta reduction relation between Y_6 and X_7. Since on the reduction chains from X_1 to R_7 exactly the same (linear) reductions are made, the last step on both chains leads to the same resulting tree.

Starting from X_7, it is very easy to see that Lemma 3 on page 52 is applicable on both branches of the reduction chain, and we again reach the same resulting tree, denoted by the variable X_{11}. So let (τ', σ') be a solution of all the constraints except the similarity constraint. Let $\sigma'(X_1) = \pi_1$, $\sigma'(X_1') = \pi_1'$, $\sigma'(Y_1) = \omega_1$, $\sigma'(Y_1') = \omega_1'$, $\sigma'(X_{11}) = \pi_{11}$, $\sigma'(X_{11}') = \pi_{11}'$, $\sigma'(Y_{11}) = \omega_{11}$, $\sigma'(Y_{11}') = \omega_{11}'$.

The only thing that remains to be checked is whether the similarity relation holds between π_{11} and ω_{11}. If we follow the reduction chains from $\sigma'(X_{11})$ back to $\sigma'(X_1)$, the tree segment π_1/π_1' and its correspondents always falls completely in the body or in the context of the reductions (notice that we used the fact that the parallelism holds between non overlapping segments!), and the same holds for ω_1/ω_1' . Let c_1 and c_2 denote the respective composition of all these correspondence functions, and let c be the correspondence function of the parallelism constraint.

We can define a correspondence function c' between $\pi_{11}{\cdot}1/\pi_{11}'$ and $\omega_{11}{\cdot}1/\omega_{11}'$ by composing all these correspondence functions:

$$c' =_{\mathsf{df}} c_1 \circ c \circ c_2^{-1}$$

We extend this correspondence function by $c'(\pi_{11}) = \omega_{11}$. Since $L(\pi_{11}') = L(\omega_{11}') = \mathsf{var}$ and $c(\lambda(\pi_{11}')) = c(\pi_{11}) = \omega_{11}' = \lambda(\omega_{11}') = \lambda(c(\pi_{11}')))$ we know that c' is also a correspondence function between the larger tree segments $\pi_{11}/$ and $\omega_{11}'/$. Global variables with respect to the similarity constraint are either bound above $\sigma'(X_{11})$ or in the tree segment $\sigma(X_{11})/\pi_{11}, \omega_{11}$. In the first case, the variable is global for all of the roots of the reduction sequence and thus is bound at the same place as all its correspondents. In the second case, let π be a variable bound below $\sigma'(X_{11})$. Because of the parallelism constraint $c_1(\pi_1)$ and $c_2(c'(\pi_1))$ are bound at the same place and therefore, π_1 and $c'(\pi_1)$ are bound at the same place. Therefore we have shown all the binding properties of the similarity relation and thus know that $\pi_{11} \sim \omega_{11}'$. It is clear that (τ', σ') is also a subsolution of (τ, σ): We know that τ is a submodel of τ', σ' coincides with σ on the old variables and we interpret the new variables in the new parts of the constraint only.

Now we turn to Property 1 of Definition 12 on page 49. Let (τ', σ') be a solution of Φ'. We will show that the submodel below the node denoted by X_1 is a model of the parallelism literal. Since the new variables of Φ' will be interpreted in submodels of the constraint that are enforced to be disjoint to X_1, we will have immediately a subsolution that satisfies the parallelism constraint.

Let $\sigma(X_1/X_1'\ ) = \alpha_1'$, $\sigma(Y_1/Y_1'\ ) = \alpha_1'$, $\sigma(X_{11}/X_{11}\ ) = \alpha_{11}$, $\sigma(Y_{11}/Y_{11}\ ) = \alpha_{11}'$. Let c be the correspondence function between α_{11} and α_{11}', which exists because of the similarity constraint. Let c_1 and c_2 be the composition of the respective correspondence functions on one of the reduction chains from X_1 to X_{11} for the segment α_1 and α_2. Again, we define the correspondence function c' between

π_1/π_1' and ω_1/ω_1' by $c' =_{\mathsf{df}} c_2 \circ c \circ c_1^{-1}$. Lets check the additional properties of the parallelism relation. Since the reductions at X_2 and X_5 do not allow that variables in the argument are captured by binders in the body, we do not have hanging binders for π_1/π_1' and ω_1/ω_1' . A node π in $\mathsf{b}^-(\alpha_1)$ that is var-labeled and bound outside the tree segment α_1 is bound at the same place like its correspondents $c(\pi)$, since already $c_2(\pi)$ is bound at the same place like $c_1^{-1}(c(\pi))$ because of property (outside) of the similarity constraint. $\square$

Note that the encoding in turn also implies, that the tree segments α, α' denoted by X/X' and Y/Y' can not overlap. To see this, assume that there is a node in a solution of the constraint that belongs to both $\mathsf{b}^-(\alpha)$ and $\mathsf{b}^-(\alpha')$. This implies that there is a node that is both below the node denoted by X_{11}^1 and below Y_{11}^1. But since $X_{11}^1 \sim Y_{11}^1$, this implies that $X{=}Y$ is entailed by Φ', and thus $\alpha = \alpha'$.

The encoding can be lifted to segments with arbitrary many holes, but makes the picture even more complex but does not require any new ideas.

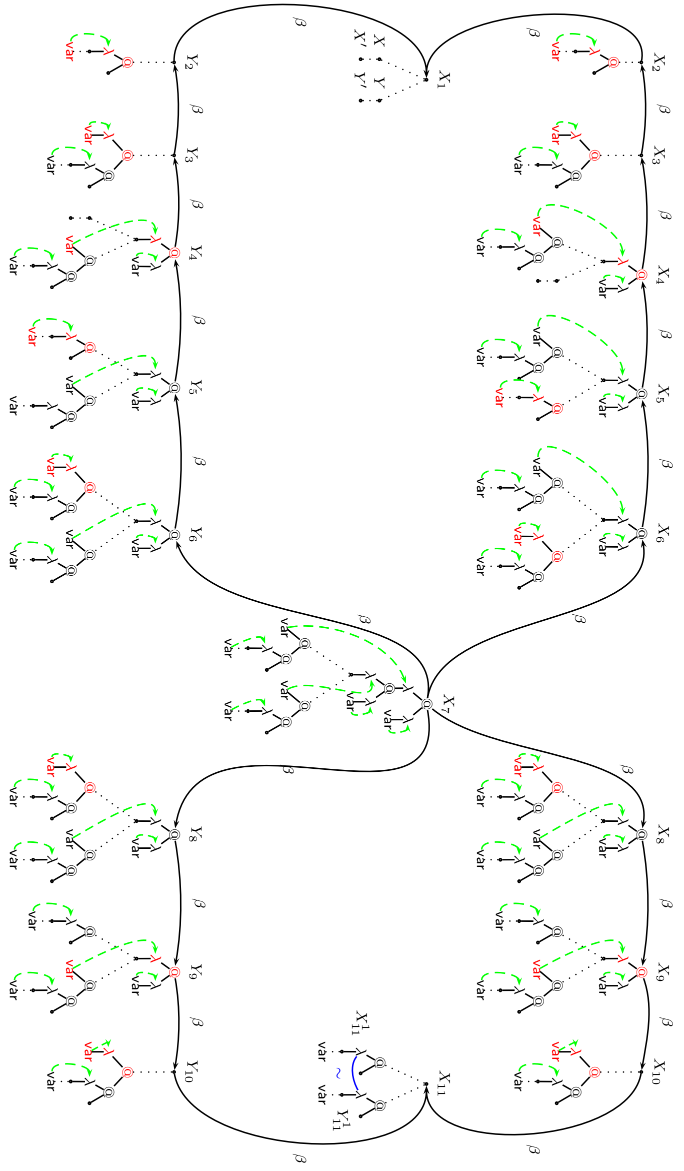

Figure 5.5: Encoding of $X/X' \sim Y/Y'$

6 Underspecified Beta Reduction

In this chapter, we argue that beta reduction constraints are the basis for a proper formulation of *underspecified beta reduction*. This formulation was not possible before and has important applications in computational linguistics.

We already saw the idea of underspecified beta reduction in Figure 1.4 on page 15. Here we first explain in which sense we want to perform underspecified beta reduction. We then present an algorithm, that is parametrized with another procedure to solve beta reduction constraints. The solver in [Bodirsky et al., 2001a] for group parallelism is a candidate for such a procedure. We improve this solver by specifying propagation rules, that use the special form of the group parallelism constraints that encode a beta reduction literal. Finally, we will illustrate the power of the procedures for underspecified beta reduction with two examples from semantic underspecification.

Beta reduction constraints only describe a single beta reduction step. In practice, however, we are interested in describing the results of several beta reduction steps. For instance in the application of underspecified semantics of natural language, we can assume simply typed lambda structures, and thus on every model of an underspecified description, every sequence of beta-reduction steps must terminate. In this case we would like to describe the λ-term after performing *all* beta reduction steps that are possible.

We are now interested in describing the results after performing as many beta reduction steps as possible. The general idea is the same as in Chapter 5 on page 47: We do not describe the set of λ-structures that we are interested in directly by a constraint, but as submodels of the solutions of a constraint. This means, if we syntactically find a redex $\mathsf{redex}^n_{X_0,X_1}(C,B,A)$ in a constraint Φ we add a β-reduction constraint $(C,B,A) \rightarrow (C',B',A'_1,\ldots,A'_n)$, where $C',B',A'_1,\ldots,A'_n$ are segment terms with new variables. The λ-structures that we are interested in can now be found below $r(C')$.

But now it will be difficult to add another beta reduction constraint, since we do not know the position of the other redexes in the reduct. Thus, we have to

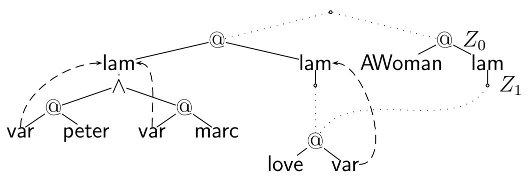

Figure 6.1: "Peter and Marc love a woman."

make information explicit in order to describe the next beta reduct. This is where theorem 2 on page 43 comes into play: by the encoding of beta reduction constraints into group parallelism we can make use of the solvers for group parallelism.

So this chapter is organized as follows: First, we describe the main procedure of underspecified beta reduction. This procedure uses the solution procedure for parallelism developed in [Erk and Niehren, 2000], that was was extended in [Bodirsky et al., 2001a] to deal with group parallelism. On the one hand, the procedure is sound and complete, but on the other hand it disambiguates a lot, and we want to avoid disambiguations. In Section 6.5, we present an alternative procedure. This procedure never disambiguates, but it cannot solve all the cases that the other solver can.

We will go through a nontrivial example to show how the complete procedure works and we will then demonstrate with another example, how the extended solver avoids disambiguation.

6.1 A Procedure for Underspecified Beta Reduction

In this section we present the procedure `usb` that for a given constraint ϕ constructs a CLLS description of a sequence of underspecified β-reduction steps. The procedure uses another procedure `solve` for solving β-reduction constraints. In the following sections we will then discuss possible candidates for `solve`.

The algorithm `usb` is shown in Figure 6.2. It starts with a constraint φ and a variable X which denotes the root of the current λ-term to be reduced. The procedure then selects an syntactic redex below X and adds a description of its reduct at a new position. Then the `solve` procedure is applied to resolve the β-reduction constraint. It returns a set of constraints because it might disambiguate; an empty set indicates an inconsistency. Finally, `usb` is called recursively with the

```
usb($\varphi$, $X$, solve)
if there is a formula redex$^{n}_{X_0,X_1}(X/X_0, B, A)$ in $\phi$
    then add $(C, B, A) \rightarrow^{\beta} (C', B', A'_1, \ldots, A'_n)$ to $\phi$
            where $C', B', A'_1, \ldots, A'_n$ are new segment terms with fresh variables
            let $\{\phi_1, \ldots, \phi_m\} = \texttt{solve}(\phi)$
            return $\bigcup_{j=1}^{m} \texttt{usb}(\phi_i, r(C'))$
    else return $\{(\phi, X)\}$
```

Figure 6.2: Underspecified β-reduction

new constraint and the root variable of the new λ-term.

If the process stops and the result contains (ψ, Y), the submodels of all the solutions below the node denoted by Y are reducts of the solutions of ϕ. Conversely, if we have a solution of ϕ, then the results of the reduction sequence are the submodels below the node denoted by R_f. Thus, we found a description of the set of all reducts as the set of all submodels found below a certain node.

It is easy to see that underspecified beta reduction of sec. 6.1 with the solver of Sec. 6.2 always terminates on if we interprete CLLS over λ-structures that represent simply typed lambda terms. 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.

6.2 Solving Group Parallelism Constraints

We now present a sound and complete semi-decision procedure for CLLS with group parallelism, which solves β-reduction constraints because of Theorem 2 on page 43. This section follows the presentation in [Bodirsky et al., 2001a]. We will then illustrate the proceeding of the procedure by an example in Section 6.4.

The procedure is obtained by extending an existing semi-decision procedure for CLLS [Erk et al., 2001] 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

$$\varphi_0 \rightarrow \vee_{i=1}^{n} \varphi_i \qquad \text{for some } n \geq 1$$

Solving dominance constraints: rule system D

(D.clash.ineq)	$X{=}Y \wedge X{\neq}Y \rightarrow \mathbf{false}$
(D.clash.disj)	$X{\perp}X \rightarrow \mathbf{false}$
(D.dom.refl)	$\varphi \rightarrow X{\lhd^*}X$ where $X \in \mathcal{V}(\varphi)$
(D.dom.trans)	$X{\lhd^*}Y \wedge Y{\lhd^*}Z \rightarrow X{\lhd^*}Z$
(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)	$X{\perp}Y \wedge X{\lhd^*}X' \wedge Y{\lhd^*}Y' \rightarrow Y'{\perp}X'$
(D.lab.dom)	$X{:}f(\ldots, Y, \ldots) \rightarrow X{\lhd^+}Y$
(D.distr.notDisj)	$X{\lhd^*}Z \wedge Y{\lhd^*}Z \rightarrow X{\lhd^*}Y \vee Y{\lhd^*}X$
(D.distr.child)	$X{\lhd^*}Y \wedge X{:}f(\overline{X}) \rightarrow Y{=}X \vee \bigvee_{i=1}^{n} X_i{\lhd^*}Y$

Figure 6.3: Saturation rules for dominance constraints

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 fresh variables, and disjunction translates to 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 $\varphi \rightarrow \Phi$ is *sound* if $\varphi \models \Phi$. Apart from that, we have saturation rules of the form $\varphi_0 \rightarrow \exists X \varphi_1$, 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**).

Figure 6.3 contains a set of saturation rules for dealing with dominance constraints (CLLS constraints of Section 3.1 without binding and parallelism). A proper treatment of dominance constraints with set operators can be found in [Duchier and Niehren, 2000, Erk et al., 2001].

The rule system B in Figure 6.4 on the next page is all we need for binding, if there are no parallelism constraints. 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.

To deal with parallelism, we first introduce some formulas that describe membership in (proper) segments and groups. Note that the terms $\mathsf{b}(A)$, $\mathsf{b}^-(A)$,

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

Figure 6.4: Treatment of binding in the absence of parallelism

$\mathsf{b}(A_1, \ldots, A_m)$ 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 $\overline{C} = (A_1, \ldots, A_n)$, $\overline{D} = (B_1, \ldots, B_n)$, and $1 \leq k \leq n$.

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

The second lets us talk about correspondence functions for a group parallelism, picking out the k-th correspondence function. In that respect, $\mathrm{co}_k^-(\overline{C}, \overline{D})$ matches the c_k of Definition 9 (except that $\mathrm{co}_k^-(\overline{C}, \overline{D})(X){=}Y$ additionally demands $X{\in}\mathsf{b}^-(A_k)$ for convenience).

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

Figure 6.5: Abbreviations for constraints describing membership in segments and groups

The rules for handling parallelism are given in Figure 6.7. The rules (P.init) and (P.new) introduce correspondence literals; between them, they state that each node in a parallel segment 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 Definition 6 demands for a correspondence function.

Figure 6.6 shows saturation rules for the interaction of group parallelism and lambda binding. The first four rule schemata directly express the conditions of Definition 9. 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 $\overline{C} = (A_1, \ldots, A_n)$:

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

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 $\overline{C} \sim \overline{D}$. Let S_1, S_2 stand for finite sets of variables, and let $\overline{C} = A_1 \ldots, A_n$.

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

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 2 (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* $1 \le k \le n$ *and all* $\pi \in \mathsf{b}^-(\alpha_k)$,

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

(L.same.seg)	$\lambda(U_1){=}U_2 \wedge \bigwedge_{i=1}^{2} \mathrm{co}_k^-(\overline{C},\overline{D})(U_i){=}V_i \rightarrow \lambda(V_1){=}V_2$
(L.diff.seg)	$\lambda(U_1){=}U_2 \wedge \bigwedge_{i=1}^{2} \mathrm{co}_{k_i}^-(\overline{C},\overline{D})(U_i){=}V_i \wedge U_2{\notin}\mathsf{b}^-(A_{k_1}) \rightarrow \lambda(V_1){=}V_2$
(L.outside)	$\lambda(U){=}Y \wedge \mathrm{co}_k^-(\overline{C},\overline{D})(U){=}V \wedge Y{\notin}\mathsf{b}^-(\overline{C}) \rightarrow \lambda(V){=}Y$
(L.hang)	$\lambda(U_1){=}U_2 \wedge \overline{C}{\sim}\overline{D} \wedge U_2{\in}\mathsf{b}^-(\overline{C}) \rightarrow U_1{\in}\mathsf{b}^-(\overline{C})$
(L.distr.1)	$\lambda(U_1){=}U_2 \wedge \overline{C}{\sim}\overline{D} \wedge U_1{\in}\mathsf{b}^-(\overline{C}) \rightarrow \mathsf{distr}_{\overline{C}}(U_2)$
(L.distr.2)	$\lambda(U_1){=}U_2 \wedge \overline{C}{\sim}\overline{D} \wedge U_2 \in \mathsf{b}^-(\overline{C}) \rightarrow \mathsf{distr}_{\overline{C}}(U_1)$
(L.equal)	$\lambda(X_1){=}X_2 \wedge \bigwedge_{i=1}^{2} X_i{=}Y_i \rightarrow \lambda(Y_1){=}Y_2$
(L.inverse)	$\lambda^{-1}(X){=}S_1 \wedge \mathrm{co}_k^-(\overline{C},\overline{D})(X){=}Y \wedge \mathrm{co}^-(\overline{C},\overline{D})(S_1){=}S_2 \cup S_3 \wedge$ $\bigwedge_{V \in S_2} \lambda(V){=}Y \wedge \bigwedge_{V \in S_3} \lambda(V){\neq}Y \rightarrow \lambda^{-1}(Y){=}S_2$

Figure 6.6: Lambda binding rules for group parallelism

Proof. Let $\omega \in \lambda^{-1}(c_k(\pi))$. The "no hanging binders" condition (hang) of Definition 9 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 Definition 6 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 Definition 9. 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 $\pi' \in \lambda^{-1}(\pi)$. □

6.3 Dealing with Correspondence Functions

In the last section 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 $\pi' \triangleleft^+ \pi$, $L_\theta(\pi_1\pi') = L_\theta(\pi_3\pi')$. This way,

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

for $1 \le i \le |\overline{C}|$, 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. . .).

(P.symm)	$\overline{C} \sim \overline{D} \to \overline{D} \sim \overline{C}$
(P.init)	$\overline{C} \sim \overline{D} \to \mathsf{seg}(A_i) \wedge \mathrm{co}(A_i, B_i)(X_i^j){=}Y_i^j$ where $1 \leq i \leq n$, $A_i = X_i^0/X_i^1, \ldots, X_i^{m_i}$, $B_i = X_i^0/Y_i^1, \ldots, Y_i^{m_i}$, and $0 \leq j \leq m_i$
(P.new)	$\overline{C} \sim \overline{D} \wedge U \in \mathsf{b}(A_i) \to \exists V\ \mathrm{co}(A_i, B_i)(U){=}V$ where V fresh, $1 \leq i \leq n$
(P.copy.lab)	$\bigwedge_{i=0}^m \mathrm{co}(A, B)(X_i){=}Y_i \wedge X_0{:}f(X_1, \ldots, X_m) \wedge X_0{\in}\mathsf{b}^-(A) \to Y_0{:}f(Y_1, \ldots, Y_m)$
(P.copy.dom)	$U_1\ R\ U_2 \wedge \bigwedge_{i=1}^2 \mathrm{co}(A, B)(U_i){=}V_i \to V_1\ R\ V_2$
(P.distr.eq)	$\varphi \to X{=}Y \vee X{\neq}Y$ for $X, Y \in \mathcal{V}(\varphi)$

Figure 6.7: Saturation rules, where $\overline{C} = A_1, \ldots, A_n$ and $\overline{D} = B_1, \ldots, B_n$

These schemata ensure the correct interaction of correspondence functions. See [Erk and Niehren, 2000] for a comprehensive treatment of this topic.

The rules we have presented form a sound and complete semi-decision procedure for group parallelism constraints.

Theorem 5. *There exists a saturation procedure GP which encompasses all instances of the rule schemata in Figure 6.3, 6.7, and 6.6, 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 [Erk and Niehren, 2000]. The proof is largely independent of the particularities of the rule system we employ.

6.4 The Procedure in Action

We illustrate the procedure of the previous section by solving the constraint in Figure 6.8. It contains a non-linear lambda redex at (C, B, A) (similarly to Figure 6.1) 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 Figure 6.9 and 6.10. We only comment on the main steps. In step (4), we have $Y_1 {\lhd^*} Z \wedge X_0 {\lhd^*} Z$, 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

$(C, B, A, A) \sim (C', B', A', A'')$
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 $A'' = X'_2/$.
$\lambda^{-1}(X_\ell) = \{X_1, X_2\}$, $\lambda^{-1}(Y_1) = \{Z\}$.

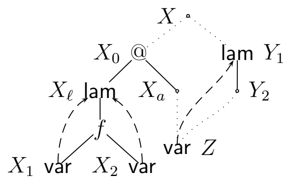

Figure 6.8: A group parallelism constraint encoding a non-linear beta reduction step

Y_2 each. This is because A is parallel both to A' and A'': because X_ℓ binds two 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 Figure 6.10, we continue case (5c) of Figure 6.9, 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.

6.5 Avoiding Disambiguation

The solver of the last sections makes extensive use of distribution rules. This means that the algorithm enumerates lots of different cases. Since we are interested in single and compact representations, we want to avoid this. For instance in the example of Figure 6.13 on page 70 we did not have to disambiguate the position of the negation before beta reducing the constraint.

In this section we will show how disambiguation can be avoided in many cases by specifying additional propagation rules. They do not work for general group parallelism constraints, but make sense for beta reduction constraints, since the group parallelism that is used to encode the beta reduction relation is of a very special form.

Finally we will go through the example of Figure 6.13 on page 70 again and will see that the description of the reducts can be derived without distribution rules.

The idea to gain stronger propagation is to use *underspecified correspondence*

(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)	$\mathrm{co}(A, A')(Y_1){=}Y'_1$, $\mathrm{co}(A, A')(Y_2){=}Y'_2$, $\mathrm{co}(A, A')(Z){=}Z'$	(P.new)	(17)	$\mathrm{co}(C, C')(X){=}X'$, $\mathrm{co}(C, C')(X_0){=}X'_0$	(P.init)
(11)	$X'_1{\triangleleft^*}Y'_1, Y'_2{\triangleleft^*}Z'$	(P.copy.dom)	(18)	$\mathrm{co}(C, C')(Y_1){=}Y'_1$, $\mathrm{co}(C, C')(Y_2){=}Y'_2$	(P.new)
(12)	$Y'_1{:}\mathsf{lam}(Y'_2)$	(P.copy.lab)	(19)	$X'{\triangleleft^*}Y'_1, Y'_2{\triangleleft^*}X'_0$	(P.copy.dom)
(13)	$\mathrm{co}(A, A'')(X_a){=}X'_2$	(P.init)	(20)	$Y'_1{:}\mathsf{lam}(Y'_2)$	(P.copy.lab)
(14)	$\mathrm{co}(A, A'')(Y_1){=}Y''_1$, $\mathrm{co}(A, A'')(Y_2){=}Y''_2$, $\mathrm{co}(A, A'')(Z){=}Z''$	(P.new)			
(15)	$X'_2{\triangleleft^*}Y''_1, Y''_2{\triangleleft^*}Z''$	(P.copy.dom)			
(16)	$Y''_1{:}\mathsf{lam}(Y''_2)$	(P.copy.lab)			

Figure 6.9: Solving the group parallelism constraint in Figure 6.8

literals

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

Here X is a variable that occurs in one of the tree segments denoted by $C_1, \ldots, C_n$, but we do not know to which one. For instance if we look at the situation in Figure 6.12, we do not know whether Z_0 belongs to the context or the argument of the redex. But it must belong to one of them, so in any case there will be a corresponding node in the reducing tree. Thus it makes sense to have a variable Y that denotes the corresponding node.

Formally, the underspecified correspondence 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 Figure 6.11 on the facing page and in the following text, we write beta_n for the constraint $(C, B, A) \to^\beta (C', B', A'_1, \ldots, A'_n)$ and corr for $\mathrm{co}(\{(C, C'), (B, B'), (A, A'_i)\})$, for any $1 \leq i \leq n$.

We have to remark several things at this point: The underspecified propagation rules do only apply if n, the number of occurrences of the vaiable bound at the redex, is zero. We will discuss the problems occuring with disappearing arguments in Chapter 7 on page 73.

	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)			

Figure 6.10: Inverse Binding in case (5c)

Note also that a inverse binding literal in the reduct can only be derived for a var-labeled node Z_0, if we have a linear redex, i.e. if the argument is not copied more than once. The linear case is especially nice: If we compare the reducing tree and the reduct, only the @- and lam-labeled nodes of the redex disappear, and no new nodes are created. Thus every lam-labeled node still binds the same number of variables. If we have to perform nonlinear beta reduction, we still often succed with propagation rules in 6.6 on page 65. But in general, we do need distribution rules and the more involved binding rules of the last section to derive all inverse binding literals.

We explain the other rules in Figure 6.11 by beta reducing the (linear) example 1.4 on page 15 of the introduction.

(UB.solve)	$(C, B, A) \overset{\beta}{\to} (C', B', A'_1, \ldots, A'_n) \to \mathsf{redex}_{X_0,X_1}(C, B, A) \wedge (C, B, A, \ldots, A) \sim (C', B', A'_1, \ldots, A'_n)$
(UB.Var)	$\mathrm{beta}_n \wedge r(C){\triangleleft^*}Z \wedge \mathsf{redex}^n_{X_0,X_1}(C, B, A) \wedge Z{\neq}X_1 \to \exists Z'.\mathrm{corr}(Z){=}Z' \quad 1 \leq i \leq n$
(UB.Copy.Label)	$\mathrm{beta}_n \wedge Z_0{:}f(Z_1, \ldots, Z_\ell) \wedge \bigwedge_{k=0}^{\ell} \mathrm{corr}(Z_k){=}Z'_k \wedge \mathsf{redex}^n_{X_0,X_1}(C, B, A) \wedge Z_0{\neq}X_0 \wedge \bigwedge_{k=1}^{n} Z_0{\neq}hs(B).k \to Z'_0{:}f(Z'_1, \ldots, Z'_\ell) \quad 1 \leq i \leq n$
(UB.Copy.Dom)	$\mathrm{beta}_n \wedge \bigwedge_{k=1}^{2} \mathrm{corr}(Z_k){=}Z'_k \wedge Z_1{\triangleleft^*}Z_2 \to Z'_1{\triangleleft^*}Z'_2 \quad 1 \leq i \leq n$
(UB.Copy.LambdaInv)	$\mathrm{beta}_n \wedge \lambda^{-1}(Z_0){=}\{Z_1, \ldots, Z_m\} \wedge \mathsf{redex}^1_{X_0,X_1}(C, B, A) \wedge \bigwedge_{k=1}^{m} \mathrm{corr}(Z_k){=}Z'_k \wedge \bigwedge_{k=1}^{m} Z_k{\neq}X_0 \wedge \bigwedge_{j=1}^{n} \bigwedge_{k=1}^{m} Z_k{\neq}hs(B).j \to \lambda^{-1}(Z'_0){=}\{Z'_1, \ldots, Z'_m\}$

Figure 6.11: The new propagation rules. See Section 6.5 on page 67 for an explanation of the abbreviations.

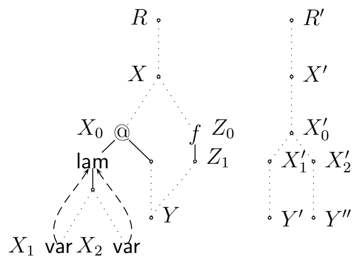

Figure 6.12: Underspecified beta redex. The position of Z_0 and Z_1 can either be in the context or the argument of the redex at X_0.

6.6 An Example

Let us again consider Figure 6.13 we aleady saw in the introduction.

We want to explain the new rules by going through the first reduction step in Figure 6.13. The β-reduction constraint that belongs to this reduction is $(C, B, A) \rightarrow^\beta (C', B', A'_1)$ with $C = R_1/Y_0$, $B = Y_1/Y_3$, $A = Y_4/$, and $C' = R_2/Z_0$, $B' = Z_0/Z_3$, $A'_1 = Z_3/$. Now the saturation algorithm proceeds as follows:

Copying variables. Using (UB.Var), we check for each variable dominated by R_1 if it has a correspondent with respect to corr. Consider for instance Y_6; it has a correspondent if it lies in one of $\mathsf{b}(C), \mathsf{b}(B), \mathsf{b}(A)$. Using the dominance rules from [Erk et al., 2001] (which are themselves part of [Bodirsky et al., 2001a]), we can infer $Y_6 \neq Y_1$ because the two variables must bear different labels. Hence

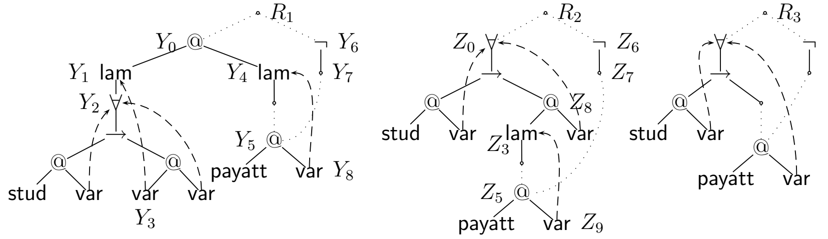

Figure 6.13: Underspecified β-reduction steps for 'Every student did not pay attention'

(UB.Var) applies, and we can introduce a fresh variable Z_6 with corr(Y_6)=Z_6. We can introduce a fresh variable Z_7 and a literal corr(Y_7)=Z_7 similarly.

Copying labeling literals. The rule (UB.Copy.Label) can copy labeling literals. Consider Y_6:$\neg(Y_7)$. The first two conditions in the rule are clearly satisfied. The third and fourth require us to verify that the labeling literal is neither the application labeling for Y_0 nor the var labeling for Y_3. We have already inferred $Y_0 \neq Y_6$ above; $Y_3 \neq Y_6$ can be derived by a similar argument. Hence the rule applies, and we can add the literal Z_6:$\neg(Z_7)$.

Copying dominance literals. The rule (UB.copy.dom) serves to copy dominance literals, without requiring any additional inequality tests. In our case, (UB.copy.dom) adds (among others) $R_2 \triangleleft^* Z_6$ and $Z_7 \triangleleft^* Z_5$.

Copying dominance literals. To be able to continue with underspecified beta reduction, there has to be a redex in the reduct. In fact, in our example we can derive another redex. The most interesting part is the inverse binding literal. With the above rules, it is easy to derive that there is a variable Z_3 below R_2 that is lam-labeled corresponding to Y_4, and a var-labeled variable Z_9 corresponding to Y_8, which is the only bound variable of Y_4. Therefore we can apply the rule (UB.copy.lambdaInv) and get $\lambda^{-1}(Z_3) = Z_9$.

7 Conclusion and Outlook

In this thesis we introduced *beta reduction constraints*. We showed how to process them in two steps : first, we expressed beta reduction constraints with CLLS and *group parallelism*. Then we extended an existing solver for parallelism in CLLS to also deal with group parallelism, and wrote propagation rules that avoid disambiguation in many cases.

Next, we expressed non overlapping parallelism constraints with beta reduction constraints. Thus it is likely that parallelism is not an overhead for the treatment of beta reduction constraints.

This work emphasizes two points: First, the thesis shows the proper treatment of binding when working with expressive constraints such as group parallelism and beta reduction constraints. At first sight it is not clear at all why the binding properties of parallelism are appropriate. We show in this thesis that the properties presented here model exactly global variables, local variables and elegantly avoid capturing. The no-hanging-binders property ensures that inverse lambda binding constraints can easily be inferred with constraint solvers. For the encodings of beta reduction into group parallelism constraints and of parallelism into beta reduction constraints it was the most difficult part to model binding correctly.

The second point is the idea to describe sets of lambda terms with constraint descriptions of *larger* lambda structures. In these larger lambda structures, we only consider the submodels starting at a specified position. This was the central idea of the definition of expressiveness of constraint languages, and this is exactly what we do when performing underspecified beta reduction.

Future Work: There are several interesting questions for future work. We would like to know whether not only parallelism, but also group parallelism is expressible with beta reduction constraints. This seems very unlikely, since group parallelism constraints in contrast to beta reduction constraints do not impose any restrictions about the relative positions of the group segments. This difference becomes meaningless, if we are allowed to use disjunction in our constraint language. But still then, the task seems not easy.

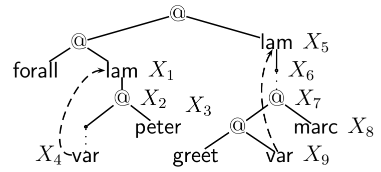

Figure 7.1: This example does not satisfy $X_2/X_3 \sim X_7/X_8$

It might also be that there is a nicer formulation of what our encodings really can express, for instance if we look at the first order theories of parallelism and beta reduction constraints, or if it is possible in the constraint language to also talk about trees and subtrees and not only about nodes.

Group Parallelism is an interesting new concept by its own. Since it is much more flexible concerning binding in λ-structures, there are potential applications in linguistics. Consider for instance the elliptic construction

John greeted everyone that Bill did.

Suppose we want to analyse this as in Figure 7.1. Unfortunately, the above constraint is unsatisfiable. Binding property (outside) of the parallelism relation requires that the nodes denoted by X_4 and X_9 are bound by the same binder [1].

In contrast, group parallelism allows to express the intended structural similarity without violating any binding restrictions by the constraint:

$$(X_1/X_2\ , X_2/X_3\ ) \sim (X_5/X_6\ , X_7/X_8\ )$$

But we have not studied how group parallelisms can be derived automatically from natural language. Moreover, it is not even clear whether group parallelism is expressive enough to cover all the phenomena occurring in complicated elliptic sentences.

Concerning underspecified beta reduction, we have important questions on both the practical and the theoretical side.

[1]After β-reduction, these two var-labeled nodes will stand for the same variables. So [Egg et al., 2001] suggest to introduce a notion of *binding equivalence*. This might still be a useful notion when deriving parallelism constraints from natural language input. But anyway, we cannot use standard parallelism constraints to write down the structural identity that holds in the semantic analysis shown in Figure 7.1.

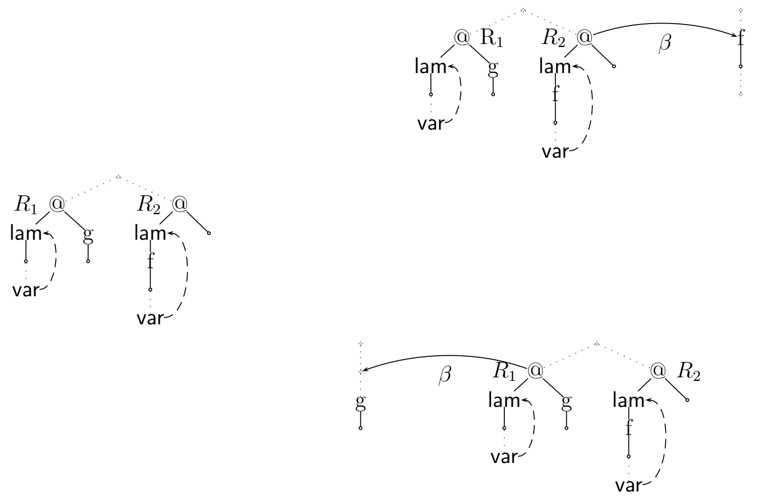

Figure 7.2: Underspecified beta reduction (without distribution rules) is not confluent: There are two different normal forms.

On the theoretical side, we want to investigate the formal properties of underspecified beta reduction. Beta reduction for the classical λ-calculus for instance is confluent: Let $\rightarrow_\beta^*$ denote the transitive closure of $\rightarrow_\beta$. Then the property of confluence says that for any λ-terms $B_1 \leftarrow_\beta^* A \rightarrow_\beta^* B_2$ there exists a λ-term C such that $B_1 \rightarrow_\beta^* C \leftarrow_\beta^* B_2$.

The notion of confluence for underspecified beta reduction remains to be defined. It is not straightforward, because one has to consider that we are only interested in submodels of solutions at the node starting at a certain solution. There is also the problem that underspecified beta reduction in general might use distribution rules and therefore works on sets of constraints.

For any underspecified beta reduction that does not use distribution rules it is very unlikely that confluence holds because of the example in Figure 7.2. There the two syntactical redexes in the constraint might denote the same nodes. If they denote different nodes, it does matter which redex we reduce first. But we cannot perform a second reduction, because they might have been equal and there is no redex in the first reduct any more. A similar argument holds for nonlinear beta reduction. If an argument is copied twice, and there is a redex for which it is not clear whether it belongs to the argument or not, the two copies of this redex might be identical or not. In the case of disappearing arguments, i.e. the binder in the redex does not bind any variable at all, the case is even worse: Every redex that could be in the argument must not be further reduced.

If we use the full solver for group parallelism, it is intuitively clear that some form of confluence holds, since the procedure might use distribution rules until the relations between all variables are determined. Such constraints directly correspond to certain minimal solutions, and then underspecified beta reduction corresponds to the known beta reduction relation with all its properties. However, this argument still has to be worked out formally.

Another interesting topic is to look for fragments of CLLS-constraints, where underspecified beta reduction is well behaved. The most promising candidate is the fragment of normal, linear beta reduction constraints. Here we do not allow that redexes that are syntactically present denote the same nodes, and we require every inverse binding literal to be a singleton. We conjecture that there is a solution procedure such that first, the underspecified beta reduction is complete in the sense that the redexes that can be found syntactically in the original constraint will be beta reduced. Second, confluence (in a way still to be defined) holds.

On the practical side we would like to mention two particularly interesting topics. 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 underspecified beta reduction. This is very promising, since CLLS can express structural similarity. Thus underspecified beta reduction can concentrate on beta reducing one argument. We use the similarity constraint to describe that all the other occurrences of the argument are simultaneously reduced. Therefore we avoid the exponential blow up, and also save underspecified reduction steps. This might also be the key to get an underspecified beta reduction procedure (in which every step describes simultaneous beta reductions at different positions) that is confluent.

Finally, we would like to apply constraint programming technology to implement underspecified beta reduction. Implementing underspecified semantic processing with constraint programming approaches is very elegant for several reasons. On the one hand we have a clear separation between the modeling and the implementation language, and are therefore very flexible. Constraint systems are modular in the sense that we can easily integrate different sources of information each of which contributes some constraints. Constraint propagation models a bidirectional flow between the various sources of information and eliminates spurious readings.

One the other hand implementations are often very efficient. Dominance constraints for example can be implemented in a nice fashion using *set constraints* [Duchier and Niehren, 2000], that are provided by a standard package of the Mozart platform for constraint programming (for instance in [Mozart]). We would like to have a similar flexible *and* efficient implementation for underspecified beta reduction.

Bibliography

E. Althaus, D. Duchier, A. Koller, K. Mehlhorn, J. Niehren, and S. Thiel. An efficient algorithm for the configuration problem of dominance graphs. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms*, pages 815–824, Washington, DC, 7–9Jan. 2001.

R. Backofen, J. Rogers, and K. Vijay-Shanker. A first-order axiomatization of the theory of finite trees. *Journal of Logic, Language, and Information*, 4:5–39, 1995.

H. P. Barendregt. *The λ-calculus, its Syntax and Semantics*. North Holland, 1984.

M. Bodirsky, K. Erk, A. Koller, and J. Niehren. Beta reduction constraints. In *International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science, Utrecht, The Netherlands, May 22–24 2001a. Springer-Verlag, Berlin.

M. Bodirsky, K. Erk, A. Koller, and J. Niehren. Underspecified beta reduction. Technical report, Universität des Saarlandes, Programming Systems Lab, 2001b. Submitted. Version of January 2001 available at `http://www.ps.uni-sb.de/Papers/abstracts/usp-beta.html`.

T. Cornell. Determining the consistency of partial descriptions of trees. In *Proceedings of the 32nd ACL*, pages 163–170, Las Cruces, New Mexico, 1994.

D. Duchier. Axiomatizing dependency parsing using set constraints. In *Sixth Meeting on Mathematics of Language*, pages 115–126, Orlando, Florida, July 1999.

D. Duchier. Constraint programming for natural language processing. Lecture Notes, European Summer School for Logic, Language and Computation (ESSLLI 2000), 2000.

D. Duchier and J. Niehren. Dominance constraints with set operators. In *Proceedings of the First International Conference on Computational Logic (CL2000)*, LNCS. Springer, July 2000.

D. Duchier and S. Thater. Parsing with tree descriptions: a constraint-based approach. In *Sixth International Workshop on Natural Language Understanding and Logic Programming (NLULP'99)*, pages 17–32, Las Cruces, New Mexico, December 1999.

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

M. Egg, J. Niehren, P. Ruhrberg, and F. Xu. Constraints over lambda-structures in semantic underspecification. In *Proceedings of the 17th International Conference on Computational Linguistics and 36th Annual Meeting of the Association for Computational Linguistics (COLING/ACL'98)*, pages 353–359, Montreal, Canada, August 1998.

K. Erk, A. Koller, and J. Niehren. Processing underspecified semantic representations in the constraint language for lambda structures. *Journal of Language and Computation*, 2001. To appear.

K. Erk and J. Niehren. Parallelism constraints. In *International Conference on Rewriting Techniques and Applications*, volume 1833 of *Lecture Notes in Computer Science*, pages 110–126, Norwich, U.K., 2000. Springer-Verlag, Berlin.

L. T. F. Gamut. *Logic, Language, and Meaning*. University of Chicago Press, Chicago and London, 1991.

P. Hirschbühler. VP deletion and across the board quantifier scope. In J. Pustejovsky and P. Sells, editors, *North East Linguistic Society (NELS 12)*, Univ. of Massachusetts, 1982.

G. P. Huet. An unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–57, 1975.

A. Koller. Constraint languages for semantic underspecification. Diplom thesis, Universität des Saarlandes, Saarbrücken, Germany, 1999. `http://www.coli.uni-sb.de/~koller/papers/da.html`.

A. Koller, K. Mehlhorn, and J. Niehren. A polynomial-time fragment of dominance constraints. In *Proceedings of the 38th Annual Meeting of the Association of Computational Linguistics*, Hong Kong, 2000a.

A. Koller, J. Niehren, and K. Striegnitz. Relaxing underspecified semantic representations for reinterpretation. *Grammars*, 3(2/3):217–241, 2000b. Special Issue on MOL'99.

A. Koller, J. Niehren, and R. Treinen. Dominance constraints: Algorithms and complexity. In *Third International Conference on Logical Aspects of Computational Linguistics (LACL '98)*, Grenoble, France, December 1998.

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

J. Lévy. Linear second order unification. In *7th International Conference on Rewriting Techniques and Applications*, volume 1103 of *LNCS*, pages 332–346, 1996.

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.

R. Montague. Universal Grammar. In R. Thomason, editor, *Formal Philosophy: Selected Papers of Richard Montague*, pages 222–246. Yale University Press, New Haven and London, 1970.

R. Montague. The proper treatment of quantification in ordinary English. In R. Thomason, editor, *Formal Philosophy. Selected Papers of Richard Montague*, pages 247–270. Yale University Press, New Haven and London, 1974.

Mozart. The Mozart System of Oz. Freely available at `www.mozart-oz.org`. Mozart Consortium: Universität des Saarlandes, DFKI, SFB 378, SICS, Université de Louvain.

J. Niehren and A. Koller. Dominance constraints in context unification. In *Third International Conference on Logical Aspects of Computational Linguistics (LACL '98)*, Grenoble, France, December 1998.

J. Niehren, M. Pinkal, and P. Ruhrberg. On equality up-to constraints over finite trees, context unification, and one-step rewriting. In *Proceedings 14th CADE*. Springer-Verlag, Townsville, 1997.

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

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

J. Rogers and K. Vijay-Shanker. Obtaining trees from their descriptions: An application to tree-adjoining grammars. *Computational Intelligence*, 10:401–421, 1994.

RTA-List. Decidability of context unification. The RTA list of open problems, number 90, `http://www.lri.fr/~rtaloop/`, 1998.

M. Schmidt-Schauß and K. Schulz. Solvability of context equations with two context variables is decidable. In *16th International Conference on Automated Deduction*, volume 1632 of *Lecture Notes in Computer Science*, 1999.

K. Striegnitz. On modeling meaning shifts by relaxing underspecified semantic representations. Diplom thesis, Universität des Saarlandes, Saarbrücken, Germany, 1999. `http://www.coli.uni-sb.de/cl/projects/chorus/papers/kris99.html`.

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