

博士論文

論文題目

Theory and application of a meta lambda calculus with cross-level computation

(レベル横断的計算機構を持つメタラムダ計算の理論と応用)

氏名

飛澤 和則

Copyright © 2015, 飛澤 和則.

This work is based on his work: A Meta Lambda Calculus with Cross-Level Computation, in Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, ACM, 2015. DOI:10.1145/2676726.2676976

Abstract

We propose meta lambda calculus λ^* as a basic model of textual substitution via metavariables. The most important feature of the calculus is that every β -redex can be reduced regardless of whether the β -redex contains meta-level variables or not. Such a meta lambda calculus has never been achieved before due to difficulty to manage binding structure consistently with α -renaming in the presence of meta-level variables. We overcome the difficulty by introducing a new mechanism to deal with substitution and binding structure in a systematic way without the notion of free variables and α -renaming.

Calculus λ^* enables us to investigate cross-level terms that include a certain type of level mismatch. Cross-level terms have been regarded as meaningless terms and left out of consideration thus far. We find that some cross-level terms behave as quotes and ‘eval’ command in programming languages. With these terms, we show a procedural language as an application of the calculus, which sheds new light on the notions of stores and recursion via meta-level variables.

Contents

Chapter 1	Introduction	1
1.1	Metavariables and Meta Lambda Calculi	1
1.2	Problem in Designing a Meta Lambda Calculus	2
1.3	Approaches in Previous Works	3
1.4	Our Purpose	5
Chapter 2	Overview of Our Approach	6
2.1	An Analysis of Calculation in the Metalanguage	6
2.2	Indexed Variables	7
2.3	Skip Operators	7
2.4	A Sketch of Meta Lambda Calculus λ^*	8
2.5	Blocks to Variable Binding	9
Chapter 3	Meta Lambda Calculus λ^*	10
3.1	Terms and Substitutions	10
3.2	Action of Substitutions on Terms	11
3.3	Reduction	15
3.4	Equivalence	15
3.5	Algebraic Properties	19
3.6	Confluence	23
Chapter 4	An Application	30
4.1	Procedural Language PROC	30
4.2	Implementation of PROC in λ^*	31
4.3	Recursion via Names and Textual Substitution	33
4.4	A Comment from Viewpoint of Type Systems	34
Chapter 5	Related Works	36
5.1	Meta Lambda Calculi	36
5.2	Other Works	37
Chapter 6	Conclusion	38
	Acknowledgements	39
	Bibliography	40

Chapter 1

Introduction

1.1 Metavariables and Meta Lambda Calculi

When discussing a formal language, we use metavariables. A metavariable is a symbol that stands for some syntactic object in a discussed language. For example, we use the following expression when defining β -reduction of the lambda calculus:

$$“(\lambda x.M)N \rightarrow_{\beta} M\{N/x\} \text{ for variables } x \text{ and terms } M, N.”$$

In this case, ‘ x ’, ‘ M ’ and ‘ N ’ are metavariables. If we instantiate the metavariables ‘ x ’, ‘ M ’, ‘ N ’ with specific syntactic objects in the lambda calculus, for example, \mathbf{x} , \mathbf{xy} , \mathbf{z} respectively, then we get ‘ $(\lambda \mathbf{x}.\mathbf{xy})\mathbf{z} \rightarrow_{\beta} (\mathbf{xy})\{\mathbf{z}/\mathbf{x}\}$ ’ as an instance of β -reduction. Note that the expression ‘ $(\mathbf{xy})\{\mathbf{z}/\mathbf{x}\}$ ’ signifies the syntactic object \mathbf{zy} in the lambda calculus, since we can calculate as $(\mathbf{xy})\{\mathbf{z}/\mathbf{x}\} = (\mathbf{x}\{\mathbf{z}/\mathbf{x}\})(\mathbf{y}\{\mathbf{z}/\mathbf{x}\}) = \mathbf{zy}$ after the instantiation of the metavariables.

Recently, several formal systems are proposed as extensions of the lambda calculus to internalize the notion of metavariables [17, 18, 9, 10, 11]. In this paper, such formal systems are collectively called **meta lambda calculi**. The most important feature of meta lambda calculi is that they include textual substitution to model instantiation of metavariables. The ordinary substitution in the lambda calculus is performed with α -renaming to avoid incidental variable binding, whereas textual substitution is performed without α -renaming by replacing each occurrence of a variable simply with a term. Thus textual substitution generates new bindings *dynamically*.

We illustrate a sketch of meta lambda calculi. Consider an extension of the syntax of the lambda calculus defined by the following BNF presented in [12]:

$$\text{Terms } M ::= x \mid \lambda x.M \mid M M \mid X \mid \delta X.M \mid M \odot M$$

where x ranges over the set of object-level variables and X ranges over the set of meta-level variables. The first three in the above BNF represent the object-level constructs, namely, the constructs in the lambda calculus. The last three represent meta-level constructs as counterparts of the object-level constructs. In the syntax, we have the following reduction sequence corresponding to the example discussed before:

$$(\delta \mathbf{M}.((\delta \mathbf{N}.(\lambda \mathbf{x}.\mathbf{M})\mathbf{N})\odot \mathbf{z}))\odot \mathbf{xy} \Rightarrow_{\beta} (\delta \mathbf{M}.(\lambda \mathbf{x}.\mathbf{M})\mathbf{z})\odot \mathbf{xy} \Rightarrow_{\beta} (\lambda \mathbf{x}.\mathbf{xy})\mathbf{z} \rightarrow_{\beta} \mathbf{zy},$$

where upper-case letters \mathbf{M} and \mathbf{N} are meta-level variables, and lower-case letters \mathbf{x} , \mathbf{y} and \mathbf{z} are object-level variables. In this paper, a meta-level application containing a meta-level abstraction as the left part is called a meta-level β -redex, and an object-level counterpart

is called an object-level β -redex. For instance, $(\delta M.(\lambda x.M)N) \odot z$ is a meta-level β -redex, and $(\lambda x.M)N$ is an object-level β -redex. Meta-level β -reduction, denoted by \Rightarrow_β in Section 1 and Section 2, is performed on a meta-level β -redex in a similar way to the ordinary β -reduction but by textual substitution. In the above example, meta-level variable M is instantiated with the term xy by meta-level β -reduction, and the object-level variable x is dynamically bound by the binder λx .

1.2 Problem in Designing a Meta Lambda Calculus

When trying to define a meta lambda calculus formally, we face a subtle problem resulting from coexistence of the ordinary substitution with textual substitution [12]. Consider the following cases to examine object-level β -redexes containing meta-level variables.

- A. Consider a term $(\lambda x.M)z$. We cannot reduce the term simply by the ordinary substitution as $(\lambda x.M)z \rightarrow_\beta M\{z/x\} = M$, since such reduction gives us the following two reduction sequences that are not confluent:

$$\begin{aligned} (\delta M.(\lambda x.M)z) \odot xy &\rightarrow_\beta (\delta M.M) \odot xy \Rightarrow_\beta xy, \\ (\delta M.(\lambda x.M)z) \odot xy &\Rightarrow_\beta (\lambda x.xy)z \rightarrow_\beta zy. \end{aligned}$$

The term zy in the second line is the intended result of the term $(\delta M.(\lambda x.M)z) \odot xy$ as shown before. It is obviously a mistake to consider that $M\{z/x\} = M$, since M is a meta-level variable to be instantiated later with a term that may contain the object-level variable x .

- B. Consider a term $(\lambda x.\lambda y.x)M$. We cannot reduce the term simply as $(\lambda x.\lambda y.x)M \rightarrow_\beta (\lambda y.x)\{M/x\} = \lambda y.(x\{M/x\}) = \lambda y.M$, since such reduction gives us the following two reduction sequences that are not confluent:

$$\begin{aligned} (\delta M.(\lambda x.\lambda y.x)M) \odot y &\rightarrow_\beta (\delta M.\lambda y.M) \odot y \Rightarrow_\beta \lambda y.y, \\ (\delta M.(\lambda x.\lambda y.x)M) \odot y &\Rightarrow_\beta (\lambda x.\lambda y.x)y \rightarrow_\alpha (\lambda x.\lambda z.x)y \rightarrow_\beta \lambda z.y. \end{aligned}$$

The second reduction sequence gives us the intended result. Note that we have the following expression with a side condition in the definition of substitution in the lambda calculus:

$$“(\lambda y.N)\{M/x\} = \lambda y.(N\{M/x\}) \quad \text{if } y \notin \text{FV}(M, x).”$$

Hence we cannot actually have $(\lambda y.x)\{M/x\} = \lambda y.(x\{M/x\})$ without satisfying a side condition such as the one above. However, we do not have such side condition in the first place, since we have no information about the free variables occurring in the term for which M stands, namely, the term with which the meta-level variable M is instantiated later.

The above cases indicate that we cannot define easily object-level substitution for terms containing meta-level variables. The heart of the problem is that a meta-level variable is just a placeholder standing for some object-level term *unknown yet*.

1.3 Approaches in Previous Works

Approaches in previous works related to modeling metavariables and the problem discussed above are classified broadly into two categories. Here we explain briefly the essential points of the two categories respectively. Features of individual works are mentioned in Section 5.

(1) Meta-Level Variables Assigned with Interfaces

One approach is to assign each meta-level variable X with a finite list of object-level variables to be bound dynamically, which is called the **interface** of meta-level variable X in this paper. With assigning an *appropriate* interface to each meta-level variable, and with *embedding* such additional information into terms explicitly, instantiation of meta-level variables can be emulated by computation in calculi dealing with only static binding, in particular, the ordinary lambda calculus [12].

For example, consider the following meta-level β -reduction:

$$(\delta M.(\lambda x.(\lambda y.Mx)x)(My)) \odot (yx) \Rightarrow_{\beta} (\lambda x.(\lambda y.(yx)x)x)((yx)y).$$

The meta-level β -reduction can be emulated by the following lambda term, which is obtained from the above term by assigning meta-level variable M with the interface (x, y) :

$$(\lambda m.(\lambda x.(\lambda y.(\underline{mxy})x)((\underline{mxy})y))(\underline{\lambda x.\lambda y.yx}) \rightarrow_{\beta}^* (\lambda x.(\lambda y.(yx)x)x)((yx)y).$$

To take a familiar example, the above fact corresponds to the fact that the C program with macros in Figure 1.1 can be emulated by the C program without macros in Figure 1.2. A meta-level variable assigned with an interface, for example, the list of x and y , is roughly equivalent to an ordinary function of x and y .

Figure1.1. WithMacros.c

```
1: int main() {
2:   int x = 1;
3:   int y = x + 1;
4:   #define M y + x
5:   x = M + y;
6:   y = x;
7:   return M + x;
8: }
```

Figure1.2. VialInterface.c

```
1: int main() {
2:   int x = 1;
3:   int y = x + 1;
4:   int m(int x, int y) {return y + x;}
5:   x = m(x, y) + y;
6:   y = x;
7:   return m(x, y) + x;
8: }
```

Calculus λm in [18] and the calculi in [12, 16, 14, 15, 4] adopt similar approaches to the one discussed above so as to consider metavariables to be assigned with interfaces. These calculi are inherently designed to deal with only static binding as the ordinary lambda calculus, and hence in these calculi, the notion of textual substitution is modeled by a mechanism like the ordinary substitution as illustrated above. As a result, the problem in question disappears from these calculi. In this paper, these calculi are called **lambda calculi with interfaces** distinctively from meta lambda calculi, since there is a difference between the concept modeled in meta lambda calculi and the concept modeled in lambda calculi with interfaces.

In order to clarify the challenge tackled in meta lambda calculi, we explain the essence of the difference between meta lambda calculi and lambda calculi with interfaces. In meta lambda calculi, meta-level variable M must satisfy the following principle for any object-level variables x and y :

$$\lambda x.M \cong \lambda y.M \quad \text{if and only if} \quad x = y,$$

where \cong denotes the equivalence induced from rewriting rules. This principle is a consequence of the requirement that the term $\lambda x.M$ is the counterpart of the expression ' $\lambda x.M$ ' in the metalanguage, and the term $\delta M.\lambda x.M$ is the counterpart of the function ' $M \mapsto \lambda x.M$ ' on the set of *all* lambda terms. The principle corresponds to the fact that two functions ' $M \mapsto \lambda x.M$ ' and ' $M \mapsto \lambda y.M$ ' are equal if and only if $x = y$. Note that if the principle did not hold and we had some distinct object-level variables x and y such that $\lambda x.M \cong \lambda y.M$, then we would have nonsense equivalence such as $\lambda x.y \cong (\delta M.\lambda x.M) \odot y \cong (\delta M.\lambda y.M) \odot y \cong \lambda y.y$.

The goal of meta lambda calculi is to add new syntactic objects called meta-level variables satisfying the above principle into the ordinary lambda calculus. In contrast, lambda calculi with interfaces as well as the ordinary lambda calculus do not contain any term that corresponds to a meta-level variable. Namely, they do not contain a term N satisfying the following property for all variables x and y :

$$\lambda x.N \cong \lambda y.N \quad \text{if and only if} \quad x = y.$$

In the example shown before, the term mxy can be used in the ordinary lambda calculus to emulate the behavior of meta-level variable M . However, the term mxy itself cannot be an alternative to meta-level variable M generally. For instance, we have distinct variables a and b satisfying $\lambda a.\text{mxy} =_\alpha \lambda b.\text{mxy}$ in the ordinary lambda calculus, whereas $\lambda a.M \not\cong \lambda b.M$ in meta lambda calculi.

(2) Level-Controlled Reduction

The other approach to resolve the problem in Section 1.2 is to restrict reduction rules by side conditions taking levels into account. To cite a case of calculus $\lambda\mathcal{M}$ in [18], object-level β -reduction $(\lambda x.M)N \rightarrow_\beta M\{N/x\}$ is defined with the following side condition:

“ M and N contain no meta-level constructs.”

The other previous meta lambda calculi [9, 10, 11] also adopt similar approaches, although there are various technical differences. A reduction defined with such side conditions is called a **level-controlled reduction** in this paper. Level-controlled reductions bring confluence in meta lambda calculi, since wrong reduction sequences mentioned in Section 1.2 are eliminated by such side conditions. However, level-controlled reductions make some β -redexes stuck. For instance, the following object-level β -redex containing free meta-level variables cannot be reduced in the previous meta lambda calculi due to such side conditions shown above:

$$(\lambda y.\lambda z.yMN)(\lambda x.\lambda y.z).$$

In other words, we cannot reduce an object-level β -redex until we instantiate meta-level variables in the β -redex with some object-level terms.

1.4 Our Purpose

In this paper, we propose meta lambda calculus λ^* with a new mechanism to reduce object-level β -redexes containing meta-level variables. Calculus λ^* enables us to advance computation in the presence of meta-level variables, and hence opens up new possibilities for reduction strategies that have been restricted by level-controlled reductions in previous meta lambda calculi. For example, we are able to introduce the notion of lazy evaluation in the presence of meta-level variables.

Another purpose of calculus λ^* is to examine those terms that include a certain type of level mismatch, such as an object-level abstraction applied to a meta-level abstraction like $(\lambda x.(x \odot M))(\delta M.M)$. Such terms are called cross-level terms in this paper. Cross-level terms have been regarded as meaningless terms, and left out of consideration by type systems or by level-controlled reductions in previous meta lambda calculi. One of distinct features of calculus λ^* is that the calculus makes it possible to compute cross-level terms. In Section 4, we show that some cross-level terms behave as quotes and ‘eval’ command in programming languages through an example. These terms provide us new insight into the notions of stores and recursion via meta-level variables.

Chapter 2

Overview of Our Approach

2.1 An Analysis of Calculation in the Metalanguage

In the metalanguage, we can always:

- rewrite a β -redex $(\lambda x.M)N$ to the expression $M\{N/x\}$,
- perform α -renaming with some fresh variable, and
- pass down substitution $\{N/x\}$ on a term to its subterms.

In this way, we can calculate a β -redex to perform substitution of variables even if the β -redex contains metavariables. For example, we can have the following calculation in the metalanguage:

$$\begin{aligned}
 (\lambda x.(\lambda y.Mx)x)(My) &\rightarrow_\alpha (\lambda x.(\lambda v.M\{v/y\}x)x)(My) \quad \text{where } v \text{ is fresh,} \\
 &\rightarrow_\beta (\lambda v.M\{v/y\}\{My/x\})(My) = (\lambda v.M\{v/y, My/x\})(My) \\
 &\rightarrow_\beta M\{v/y, My/x\}\{My/v\}(M\{My/v\}y) \\
 &= M\{My/y, M\{My/v\}y/x, My/v\}(M\{My/v\}y) \\
 &= M\{My/y, My/x\}(My) \quad \text{since } v \text{ is fresh.}
 \end{aligned}$$

Note that x and y are variables as syntactic objects in the lambda calculus, whereas v is a metavariable that stands for some variable in the lambda calculus. If we instantiate metavariables M in the last line with term yx , we get the following result:

$$(yx)\{(yx)y/y, (yx)y/x\}((yx)y) = (yxy)(yxy)(yxy).$$

The key factors that enable us to advance calculation in the presence of metavariables are:

- substitution operators, and
- freshness conditions accompanied with α -renaming.

Note that these two factors interact with each other. In the above example, the freshness condition ' v is fresh' first arises in association with α -renaming of the binder λy to λv . In the process of the calculation, the freshness condition deletes substitution operators $\{My/v\}$ following metavariables M . Furthermore, the last β -reduction that yields substitution operator $\{My/v\}$ actually eliminates the freshness condition ' v is fresh' at the end of the calculation. Consequently, we have

$$(\lambda x.(\lambda y.Mx)x)(My) \rightarrow_\beta^* M\{My/y, My/x\}(My)$$

for any term M .

We attempt to incorporate the two notions of substitution operators and freshness conditions into a meta lambda calculus to achieve our purpose. It is not difficult to embed substitution operators as syntactic objects. A major obstacle is how to deal with freshness conditions and the interaction with substitution operators in a meta lambda calculus. As explained in the subsequent discussion, we overcome the obstacle by adopting indexed variables and skip operators as alternatives to freshness conditions.

2.2 Indexed Variables

Consider the extended syntax of the lambda calculus defined by the following BNF:

$$\text{Terms } M ::= v^d \mid \lambda v.M \mid M M$$

where v ranges over a set of names and d ranges over the set of nonnegative integers. In this syntax, a variable consists of a name v and a nonnegative integer d called the index of the variable. In a term, variable v^d skips d binders of v and hence is bound by the $(d + 1)$ -th binder of v [16]. For instance, in term $\lambda x.\lambda y.\lambda x.y^0x^1$, the variable y^0 is bound by the binder λy as in the ordinary lambda calculus, whereas the variable x^1 skips the rightmost binder λx and is bound by the leftmost binder λx . In a word, we consider an extension of the lambda calculus accompanied with de Bruijn indices [5].

The extended syntax enables us to perform β -reduction without α -renaming, although we can still have the notion of α -renaming. For example, we can reduce a term $(\lambda x.\lambda y.y^0x^0)(\lambda x.y^0)$ in the following two ways:

$$\begin{aligned} (\lambda x.\lambda y.y^0x^0)(\lambda x.y^0) &\rightarrow_{\beta} \lambda y.y^0(\lambda x.y^1) \rightarrow_{\alpha} \lambda z.z^0(\lambda x.y^0), \\ (\lambda x.\lambda y.y^0x^0)(\lambda x.y^0) &\rightarrow_{\alpha} (\lambda x.\lambda z.z^0x^0)(\lambda x.y^0) \rightarrow_{\beta} \lambda z.z^0(\lambda x.y^0). \end{aligned}$$

The reduction sequence in the second line is regarded as the one in the ordinary lambda calculus. Note that if we equate α -equivalent terms as we usually do in the ordinary lambda calculus, then the above two reduction sequences are identical. As a result, the extended syntax provides us a more expressive notation virtually without changing the theory of the lambda calculus.

2.3 Skip Operators

In the extended syntax, we no longer need the notion of freshness of variables. For example, we can calculate with metavariables as follows:

$$(\lambda x.\lambda y.y^0x^0)M \rightarrow_{\beta} (\lambda y.y^0x^0)\{M/x\} = \lambda y.y^0(M * \uparrow_y),$$

where \uparrow_y is a skip operator acting on the term M to increment indices of y in term M appropriately so as to skip the binder λy . For instance, $(\lambda x.y^0) * \uparrow_y$ denotes $\lambda x.y^1$. The above calculation corresponds to the following calculation with a freshness condition in the ordinary lambda calculus:

$$(\lambda x.\lambda y.yx)M \rightarrow_{\alpha} (\lambda x.\lambda v.vx)M \rightarrow_{\beta} \lambda v.vM \quad \text{where } v \text{ is fresh.}$$

Skip operators free us from the need to perform α -renaming and the need to keep freshness conditions outside of terms. Consequently, skip operators can be more easily embedded into a meta lambda calculus than freshness conditions.

2.4 A Sketch of Meta Lambda Calculus λ^*

We construct meta lambda calculus λ^* in Section 3 by adopting indexed variables and by embedding both substitution operators and skip operators as syntactic objects in the calculus. Here we show a sketch to give an intuition about the calculus.

The syntax defined by the following BNF is a part of the syntax of calculus λ^* :

$$\begin{aligned} \text{Terms } M &::= v^d \mid \lambda v.M \mid M M \mid X[\sigma] \mid \delta X.M \mid M \odot M \\ \text{Substitutions } \sigma &::= \text{id} \mid {}^v\downarrow(M) \cdot \sigma \mid \uparrow_v \cdot \sigma \end{aligned}$$

where v ranges over a set of names, d ranges over the set of nonnegative integers, X ranges over the set of meta-level variables, and id signifies the empty sequence. The elements of form ${}^v\downarrow(M)$, called **push-elements**, are counterparts of substitution operators ' $\{M/v\}$ ', and the elements of form \uparrow_v , called **pop-elements**, are counterparts of skip operators. In calculus λ^* , a substitution σ is dealt with as a finite sequence of push-elements and pop-elements in order to calculate the two kinds of elements in an integrated way to model the interaction between substitution operators and freshness conditions in the metalanguage. Note that a substitution σ can occur only with a meta-level variable X in the form $X[\sigma]$. The σ represents a substitution that is suspended to wait for instantiation of the meta-level variable X . For instance, term $M[{}^x\downarrow(z^0)]$ corresponds to the expression ' $M\{z/x\}$ ' that cannot be calculated any more in the metalanguage.

Action of a substitution σ on a term M , denoted by $M * \sigma$, is designed to reflect the behaviors of substitution operators and skip operators as alternatives to freshness conditions. For instance, the expression $(x^0M) * {}^x\downarrow(z^0)$ denotes the term $z^0M[{}^x\downarrow(z^0)]$, as the expression ' $(xM)\{z/x\}$ ' is reduced to ' $z(M\{z/x\})$ ' in the ordinary lambda calculus. Note that the substitution ${}^x\downarrow(z^0)$ is suspended on the meta-level variable M , since the substitution cannot act any more until M is instantiated later. Also, the expression $(\lambda y.y^0x^0) * {}^x\downarrow(M)$ denotes $\lambda y.y^0M[\uparrow_y]$, which corresponds to the expression ' $\lambda y.y^0(M * \uparrow_y)$ ' shown in Section 2.3.

The syntax enables us to advance computation in the presence of meta-level variables in the same way as the metalanguage. For example, the calculation shown in Section 2.1 is represented by the following reduction sequence in calculus λ^* :

$$\begin{aligned} (\lambda x.(\lambda y.Mx)x)(My) &\rightarrow_\beta ((\lambda y.Mx)x) * {}^x\downarrow(My) = (\lambda y.M[{}^y\downarrow(y) \cdot {}^x\downarrow(M[\uparrow_y]y^1) \cdot \uparrow_y](M[\uparrow_y]y^1))(My) \\ &\rightarrow_\beta (M[{}^y\downarrow(y) \cdot {}^x\downarrow(M[\uparrow_y]y^1) \cdot \uparrow_y](M[\uparrow_y]y^1)) * {}^y\downarrow(My) \\ &= M[{}^y\downarrow(My) \cdot {}^x\downarrow(M[\uparrow_y] \cdot {}^y\downarrow(My))y \cdot \uparrow_y \cdot {}^y\downarrow(My)](M[\uparrow_y] \cdot {}^y\downarrow(My))y \rightarrow_\epsilon^* M[{}^y\downarrow(My) \cdot {}^x\downarrow(My)](My), \end{aligned}$$

where indices equal to 0 are omitted. In the last line, ϵ -reduction eliminates substitutions of form $\uparrow_v \cdot {}^v\downarrow(N)$ to simulate the interaction between substitution operators and freshness conditions. If we instantiate the meta-level variables M in the last line with term yx , we get the following intended result:

$$(M[{}^y\downarrow(My) \cdot {}^x\downarrow(My)](My)) * {}^M\downarrow(yx) = ((yx) * ({}^y\downarrow((yx)y) \cdot {}^x\downarrow((yx)y)))(yx)y = (yxy)(yxy)(yxy),$$

where the meta-level push-element ${}^M\downarrow(yx)$ represents the instantiation of M . Note that the suspended substitution for object-level variables y and x following the meta-level variable M in the first line is resumed to obtain the result after the instantiation of M .

In the discussion thus far, we illustrate a sketch of an extension of the lambda calculus added meta-level constructs. In the same way, we can easily add meta-meta-level constructs, meta-meta-meta-level constructs and so on. As a result, we obtain meta lambda calculus λ^* that includes infinitely hierarchical levels as meta lambda calculi in [18, 9, 10].

2.5 Blocks to Variable Binding

Lastly, we make a remark about a connection between object-level variable binding and meta-level application. In calculus λ^* , we have the following equality:

$$(M_1 \odot M_2) * v\downarrow(N) = (M_1 * v\downarrow(N)) \odot M_2.$$

This equality states that object-level variables v^d occurring in the right part M_2 of meta-level application $M_1 \odot M_2$ are prevented from being bound by outer binders of v [13]. As an example, we take a term $(\lambda y.((\delta M.(\lambda x.(\lambda y.Mx)x)(My)) \odot (yx)))(x1)$ that roughly corresponds to a part of the C program with macros in Figure 1.1. We should consider that the variable y in the right part ‘ yx ’ of the meta-level application is not bound by the leftmost binder λy . Otherwise, we could perform substitution for the y as follows:

$$\begin{aligned} & (\lambda y.((\delta M.(\lambda x.(\lambda y.Mx)x)(My)) \odot (yx)))(x1) \rightarrow_{\beta} ((\delta M.(\lambda x.(\lambda y.Mx)x)(My)) \odot (yx)) * v\downarrow(x1) \\ & = ((\delta M.(\lambda x.(\lambda y.Mx)x)(My)) * v\downarrow(x1)) \odot ((yx) * v\downarrow(x1)) \\ & = ((\delta M.(\lambda x.(\lambda y.Mx)x)(My)) * v\downarrow(x1)) \odot ((x1)x). \end{aligned}$$

Such a reduction prevents a calculus from satisfying confluence, since we also have the following reduction sequence that generates new bindings dynamically:

$$\begin{aligned} & (\lambda y.((\delta M.(\lambda x.(\lambda y.Mx)x)(My)) \odot (yx)))(x1) \Rightarrow_{\beta} (\lambda y.(((\lambda x.(\lambda y.Mx)x)(My)) * M\downarrow(yx)))(x1) \\ & = (\lambda y.(\lambda x.(\lambda y.(yx)x)x)((yx)y))(x1). \end{aligned}$$

Note that in the term *after* the meta-level β -reduction, the variables y in the two copies ‘ yx ’ are *newly bound* by one of the two binders λy respectively. This fact indicates that in the term *before* the meta-level β -reduction, we should consider that the variable y in the right part ‘ yx ’ of the meta-level application copied by instantiation is *not bound yet* by any binder. Stated differently, in the C program with macros in Figure 1.1, the variable y in the fifth line signifies the *value* of y determined by assignment in the third line, whereas the variable y on the right of `#define` command in the fourth line signifies the *text* ‘ y ’ itself, which does not refer any value yet.

Chapter 3

Meta Lambda Calculus λ^*

We formally define meta lambda calculus λ^* and show that the calculus satisfies confluence.

3.1 Terms and Substitutions

We consider the set \mathcal{L} of **levels** as the set of positive integers. The object-level is 1, the meta-level is 2, the meta-meta-level is 3, and so on. We assume that we are given the set \mathcal{N} of **names** and function Lv of \mathcal{N} into \mathcal{L} that assigns a level to each name.

Definition 3.1.1. We define inductively the set *Ter* of **terms** and the auxiliary set *Sub_ℓ* for each level ℓ as follows:

$$\begin{aligned}
 v^d[\sigma] &\in \text{Ter} && \text{if } v \in \mathcal{N}, d \in \mathbb{N}, \text{ and } \sigma \in \text{Sub}_{\text{Lv}(v)}, \\
 \lambda v.M &\in \text{Ter} && \text{if } v \in \mathcal{N} \text{ and } M \in \text{Ter}, \\
 M @_\ell N &\in \text{Ter} && \text{if } \ell \in \mathcal{L} \text{ and } M, N \in \text{Ter}, \\
 \text{id} &\in \text{Sub}_\ell, \\
 {}^v\downarrow(M) \cdot \sigma &\in \text{Sub}_\ell && \text{if } v \in \mathcal{N} \text{ with } \text{Lv}(v) < \ell, M \in \text{Ter}, \text{ and } \sigma \in \text{Sub}_\ell, \\
 \uparrow_v \cdot \sigma &\in \text{Sub}_\ell && \text{if } v \in \mathcal{N} \text{ with } \text{Lv}(v) < \ell \text{ and } \sigma \in \text{Sub}_\ell,
 \end{aligned}$$

where \mathbb{N} is the set of nonnegative integers and *id* signifies the empty sequence. We define the set *Sub* of **substitutions** as the union $\bigcup_{\ell \in \mathcal{L}} \text{Sub}_\ell$. Note that a substitution is a finite sequence of push-elements ${}^v\downarrow(M)$ and pop-elements \uparrow_v . Push-elements and pop-elements are collectively called push-pop-elements.

Notation. We also use the dot symbol ‘ \cdot ’ to represent concatenation of substitutions. For instance, if substitution σ equals ${}^v\downarrow(M) \cdot \uparrow_v$, then $\sigma \cdot \sigma$ denotes ${}^v\downarrow(M) \cdot \uparrow_v \cdot {}^v\downarrow(M) \cdot \uparrow_v$. In particular $\text{id} \cdot \sigma = \sigma \cdot \text{id} = \sigma$. We also use exponential notation, namely, $\sigma^0 = \text{id}$ and $\sigma^{n+1} = \sigma \cdot \sigma^n$ for substitutions σ and nonnegative integers n . As shown in the following example, we omit superscripts ‘0’ following names and brackets ‘[id]’ around the empty sequence when no confusion may occur.

Example 3.1.2. The following are terms, where x, y, z are names of level 1, and M, N are names of level 2:

$$(\lambda x.y^0[\text{id}]) @_1 z^0[\text{id}] \quad \text{and} \quad M^0[{}^x\downarrow(N^1[\text{id}]) \cdot \uparrow_x \cdot {}^y\downarrow(\lambda x.M^2[\uparrow_y \cdot \uparrow_z])].$$

We usually represent the first term simply as $(\lambda x.y) @_1 z$.

3.2 Action of Substitutions on Terms

Definition 3.2.1. Let v be a name and d a nonnegative integer. The $\langle v, d \rangle$ -**component** of a substitution σ , denoted by $\sigma\langle v, d \rangle$, is the term defined inductively as follows:

$$\begin{aligned} \text{id}\langle v, d \rangle &:= v^d[\text{id}], \\ ({}^w\downarrow(M) \cdot \sigma)\langle v, d \rangle &:= \begin{cases} M & \text{if } v = w \text{ and } d = 0, \\ \sigma\langle v, d - \delta_{vw} \rangle & \text{otherwise,} \end{cases} \\ (\uparrow_w \cdot \sigma)\langle v, d \rangle &:= \sigma\langle v, d + \delta_{vw} \rangle, \end{aligned}$$

where δ_{vw} is the nonnegative integer defined by

$$\delta_{vw} := \begin{cases} 1 & \text{if } v = w, \\ 0 & \text{otherwise.} \end{cases}$$

We give an intuitive explanation about $\langle v, d \rangle$ -components of a substitution σ . Consider an \mathcal{N} -indexed family of infinite stacks of terms. Such a family of stacks is called an environment here. Push-elements ${}^v\downarrow(M)$ and pop-elements \uparrow_v represent operations on the stack indexed by name v . Hence substitution σ as a composition of push-pop-elements represents an operation on environments. In what follows, the stack indexed by name v in an environment is simply called the v -stack. The environment whose v -stack consists of the infinite sequence v^0, v^1, v^2, \dots for each name v , is called the identity environment. The $\langle v, d \rangle$ -component $\sigma\langle v, d \rangle$ signifies the element at depth d in the v -stack in the environment obtained by operating σ on the identity environment.

For instance, consider a substitution $\sigma = {}^v\downarrow(M_0) \cdot {}^v\downarrow(M_1) \cdot \uparrow_v^n$. The substitution σ means the operation “pop from the v -stack n times, and then push term M_1 into the v -stack, and then push term M_0 into the v -stack.” By operating σ on the identity environment, we obtain the environment whose v -stack consists of the infinite sequence $M_0, M_1, v^n, v^{n+1}, v^{n+2}, \dots$, and thus we have $\sigma\langle v, 0 \rangle = M_0$, $\sigma\langle v, 1 \rangle = M_1$, $\sigma\langle v, 2 \rangle = v^n$, $\sigma\langle v, 3 \rangle = v^{n+1}$, $\sigma\langle v, 4 \rangle = v^{n+2}$, and so on.

Definition 3.2.2. Let S be a subset of the set \mathcal{N} of names. The S -**restriction** of a substitution σ , denoted by $\sigma \upharpoonright S$, is the substitution defined inductively as follows:

$$\begin{aligned} \text{id} \upharpoonright S &:= \text{id}, \\ ({}^v\downarrow(M) \cdot \sigma) \upharpoonright S &:= \begin{cases} {}^v\downarrow(M) \cdot (\sigma \upharpoonright S) & \text{if } v \in S, \\ \sigma \upharpoonright S & \text{otherwise,} \end{cases} \\ (\uparrow_v \cdot \sigma) \upharpoonright S &:= \begin{cases} \uparrow_v \cdot (\sigma \upharpoonright S) & \text{if } v \in S, \\ \sigma \upharpoonright S & \text{otherwise.} \end{cases} \end{aligned}$$

For levels ℓ and substitutions σ , we define $\sigma_{<\ell}$ and $\sigma_{\geq\ell}$ as follows:

$$\begin{aligned} \sigma_{<\ell} &:= \sigma \upharpoonright \{v \in \mathcal{N} \mid \text{Lv}(v) < \ell\}, \\ \sigma_{\geq\ell} &:= \sigma \upharpoonright \{v \in \mathcal{N} \mid \text{Lv}(v) \geq \ell\}. \end{aligned}$$

We abbreviate $\sigma_{<\text{Lv}(v)}$ and $\sigma_{\geq\text{Lv}(v)}$ to $\sigma_{<v}$ and $\sigma_{\geq v}$ respectively for names v .

Definition 3.2.3. We define inductively the term $M * \sigma$ resulting from **action** of a substitution σ on a term M , and the substitution $\tau \circ \sigma$ resulting from **composition** of substitutions σ and τ , as follows:

$$\begin{aligned}
v^d[\tau] * \sigma &:= \begin{cases} v^d[\sigma] & \text{if } \tau = \sigma_{\geq v} = \text{id}, \\ \sigma\langle v, d \rangle * (\tau \circ \sigma)_{<v} & \text{otherwise,} \end{cases} \\
(\lambda v.M) * \sigma &:= \lambda v.(M * \uparrow_v(\sigma)), \\
(M @_{\ell} N) * \sigma &:= (M * \sigma) @_{\ell} (N * \sigma_{\geq \ell}), \\
\text{id} \circ \sigma &:= \sigma, \\
({}^v\downarrow(M) \cdot \tau) \circ \sigma &:= {}^v\downarrow(M * \sigma_{\geq v}) \cdot (\tau \circ \sigma), \\
(\uparrow_v \cdot \tau) \circ \sigma &:= \uparrow_v \cdot (\tau \circ \sigma),
\end{aligned}$$

where $\uparrow_v(\sigma)$ denotes ${}^v\downarrow(v^0[\text{id}]) \cdot (\sigma \circ \uparrow_v)$.

The first branch in the first line for action on a variable of name v states that substitution σ does nothing and stays as the suspended substitution on the variable, if the σ does not contain any push-pop-elements of level $\ell \geq \text{Lv}(v)$. The second branch states that substitution σ replaces the variable $v^d[\tau]$ with the corresponding term $\sigma\langle v, d \rangle$, and the suspended substitution τ composed with σ further acts on the term. The restriction represented by ' $<$ ' indicates that all push-pop-elements of level $\ell \geq \text{Lv}(v)$ in σ are spent for action of substitution on the variable $v^d[\tau]$. The expression $\uparrow_v(\sigma)$ in the second line represents a substitution obtained by adjusting σ to skip the binder ' λv ' appropriately. The restriction represented by ' \geq ' in the third line and the fifth line is due to blocks to variable binding discussed in Section 2.5. Note that we have $v^d[\tau] * \sigma = \sigma\langle v, d \rangle * (\tau \circ \sigma)_{<v}$ even if $\tau = \sigma_{\geq v} = \text{id}$, since then $\sigma\langle v, d \rangle * (\tau \circ \sigma)_{<v} = v^d * \sigma_{<v} = v^d[\sigma_{<v}] = v^d[\sigma]$.

We show that $M * \sigma$ and $\sigma \circ \tau$ are well-defined for all terms M and all substitutions σ and τ by induction on level of substitutions, height of terms, and length of substitutions defined below.

Definition 3.2.4. The level of a substitution σ , denoted by $\text{Lv}(\sigma)$, is the nonnegative integer defined inductively as follows:

$$\begin{aligned}
\text{Lv}(\text{id}) &:= 0, \\
\text{Lv}({}^v\downarrow(M) \cdot \sigma) &:= \begin{cases} \text{Lv}(\sigma) & \text{if } M = v^d[\text{id}] \text{ for a nonnegative integer } d, \\ \max\{\text{Lv}(v), \text{Lv}(\sigma)\} & \text{otherwise,} \end{cases} \\
\text{Lv}(\uparrow_v \cdot \sigma) &:= \text{Lv}(\sigma).
\end{aligned}$$

Definition 3.2.5. The height of a term M , denoted by $\text{Ht}(M)$, and the height of a substitution σ , denoted by $\text{Ht}(\sigma)$, are the nonnegative integers defined inductively as follows:

$$\begin{aligned}
\text{Ht}(v^d[\sigma]) &:= \text{Ht}(\sigma), \\
\text{Ht}(\lambda v.M) &:= \text{Ht}(M) + 1, \\
\text{Ht}(M @_{\ell} N) &:= \max\{\text{Ht}(M), \text{Ht}(N)\} + 1, \\
\text{Ht}(\text{id}) &:= 0, \\
\text{Ht}({}^v\downarrow(M) \cdot \sigma) &:= \max\{\text{Ht}(M) + 1, \text{Ht}(\sigma)\}, \\
\text{Ht}(\uparrow_v \cdot \sigma) &:= \text{Ht}(\sigma).
\end{aligned}$$

Definition 3.2.6. The length of a substitution σ , denoted by $\text{Lh}(\sigma)$, is the nonnegative integer defined inductively as follows:

$$\begin{aligned}\text{Lh}(\text{id}) &:= 0, \\ \text{Lh}(\downarrow(M) \cdot \sigma) &:= 1 + \text{Lh}(\sigma), \\ \text{Lh}(\uparrow_v \cdot \sigma) &:= 1 + \text{Lh}(\sigma).\end{aligned}$$

Remark 3.2.7. Let σ be a substitution, ℓ a level, v a name, d a nonnegative integer, and S a subset of the set \mathcal{N} of names. Then we have the following basic properties, which are used in the subsequent proofs:

- (1) $\text{Lv}(\sigma \upharpoonright S) \leq \text{Lv}(\sigma)$.
- (2) $\text{Lv}(\sigma_{<\ell}) < \ell$.
- (3) If $v \in S$, then $(\sigma \upharpoonright S)\langle v, d \rangle = \sigma\langle v, d \rangle$.
- (4) If $\text{Ht}(\sigma) > 0$, then $\text{Ht}(\sigma\langle v, d \rangle) < \text{Ht}(\sigma)$.

These propositions are all trivial by definition.

Lemma 3.2.8. Let σ be a substitution, v a name, and d a nonnegative integer. If $\text{Lv}(\sigma) < \text{Lv}(v)$, then $\sigma\langle v, d \rangle = v^e$ for a nonnegative integer e .

Proof. We use induction on $\text{Lh}(\sigma)$. Suppose that $\text{Lv}(\sigma) < \text{Lv}(v)$.

- A. Consider the case that $\sigma = \text{id}$. Then $\sigma\langle v, d \rangle = v^d$.
- B. Consider the case that $\sigma = \downarrow(M) \cdot \sigma_1$. Then we have the following two cases.
 1. Consider the case that $M = w^c$ for a nonnegative integer c . If $v = w$ and $d = 0$, then $\sigma\langle v, d \rangle = M = w^c = v^c$. Otherwise, we have $\sigma\langle v, d \rangle = \sigma_1\langle v, d - \delta_{vw} \rangle = v^e$ for a nonnegative integer e by induction hypothesis.
 2. Consider the case that $M \neq w^c$ for any nonnegative integer c . Since $\text{Lv}(v) > \text{Lv}(\sigma) = \max\{\text{Lv}(w), \text{Lv}(\sigma_1)\}$, we have $v \neq w$. Thus $\sigma\langle v, d \rangle = \sigma_1\langle v, d \rangle = v^e$ for a nonnegative integer e by induction hypothesis.
- C. Consider the case that $\sigma = \uparrow_w \cdot \sigma_1$. Then $\sigma\langle v, d \rangle = \sigma_1\langle v, d + \delta_{vw} \rangle = v^e$ for a nonnegative integer e by induction hypothesis. \square

Proposition 3.2.9. Let M be a term and σ a substitution. Then $M * \sigma$ is well-defined.

Proof. We use induction on lexicographic ordering of pairs $\langle \text{Lv}(\sigma), \text{Ht}(M) \rangle$.

- A. Consider the case that $M = v^d[\tau]$. First, we show that $\tau \circ \sigma$ is well-defined. We use induction on $\text{Lh}(\tau)$.
 1. Consider the case that $\tau = \text{id}$. Then $\tau \circ \sigma = \sigma$, and thus $\tau \circ \sigma$ is well-defined.
 2. Consider the case that $\tau = \downarrow(M_1) \cdot \tau_1$. Then $\tau \circ \sigma = \downarrow(M_1 * \sigma_{\geq w}) \cdot (\tau_1 \circ \sigma)$. We see that $M_1 * \sigma_{\geq w}$ is well-defined by induction hypothesis on $\langle \text{Lv}(\sigma), \text{Ht}(M) \rangle$, since $\text{Lv}(\sigma_{\geq w}) \leq \text{Lv}(\sigma)$ and $\text{Ht}(M_1) < \text{Ht}(M)$. We also see that $\tau_1 \circ \sigma$ is well-defined by induction hypothesis on $\text{Lh}(\tau)$. Thus $\tau \circ \sigma$ is well-defined.
 3. Consider the case that $\tau = \uparrow_w \cdot \tau_1$. Then $\tau \circ \sigma = \uparrow_w \cdot (\tau_1 \circ \sigma)$. Since $\tau_1 \circ \sigma$ is well-defined by induction hypothesis on $\text{Lh}(\tau)$, we see that $\tau \circ \sigma$ is well-defined.

By the above, we see that $\tau \circ \sigma$ is well-defined. Next, we show that $\sigma\langle v, d \rangle * (\tau \circ \sigma)_{<v}$ is well-defined.

1. Consider the case that $\text{Lv}(\sigma) < \text{Lv}(v)$. Then $\sigma\langle v, d \rangle = v^e$ for a nonnegative integer e by Lemma 3.2.8. Hence $\sigma\langle v, d \rangle * (\tau \circ \sigma)_{<v} = v^e * (\tau \circ \sigma)_{<v} = v^e[(\tau \circ \sigma)_{<v}]$.
2. Consider the case that $\text{Lv}(\sigma) \geq \text{Lv}(v)$. Then $\sigma\langle v, d \rangle * (\tau \circ \sigma)_{<v}$ is well-defined by induction hypothesis, since $\text{Lv}((\tau \circ \sigma)_{<v}) < \text{Lv}(v) \leq \text{Lv}(\sigma)$.

By the above, we see that $\sigma\langle v, d \rangle * (\tau \circ \sigma)_{<v}$ is well-defined. Hence $M * \sigma = v^d[\tau] * \sigma$ is well-defined.

B. Consider the case that $M = \lambda v.M_1$. First, we show that $\sigma \circ \uparrow_v$ is well-defined and $\text{Lv}(\sigma \circ \uparrow_v) = \text{Lv}(\sigma)$. We use induction on $\text{Lh}(\sigma)$.

1. Consider the case that $\sigma = \text{id}$. Then $\sigma \circ \uparrow_v = \uparrow_v$ and thus $\sigma \circ \uparrow_v$ is well-defined. We also have $\text{Lv}(\sigma \circ \uparrow_v) = \text{Lv}(\uparrow_v) = 0 = \text{Lv}(\sigma)$.
2. Consider the case that $\sigma = {}^w\downarrow(N) \cdot \sigma_1$. Then $\sigma \circ \uparrow_v = {}^w\downarrow(N * (\uparrow_v)_{\geq w}) \cdot (\sigma_1 \circ \uparrow_v)$. First, we show that $\sigma \circ \uparrow_v$ is well-defined.
 - a. If $\text{Lv}(\sigma) = 0$, then $N = w^e$ for a nonnegative integer e . Hence $N * (\uparrow_v)_{\geq w} = (\uparrow_v)_{\geq w} \langle w, e \rangle * (\text{id} \circ (\uparrow_v)_{\geq w})_{< w} = w^{e+\delta_{vw}} * \text{id} = w^{e+\delta_{vw}}$.
 - b. If $\text{Lv}(\sigma) > 0$, then $N * (\uparrow_v)_{\geq w}$ is well-defined by induction hypothesis, since $\text{Lv}((\uparrow_v)_{\geq w}) = 0 < \text{Lv}(\sigma)$.

By the above, we see that $N * (\uparrow_v)_{\geq w}$ is well-defined. We also see that $\sigma_1 \circ \uparrow_v$ is well-defined by induction hypothesis on $\text{Lh}(\sigma)$. Therefore $\sigma \circ \uparrow_v$ is well-defined.

Next, in order to show that $\text{Lv}(\sigma \circ \uparrow_v) = \text{Lv}(\sigma)$, we prove that the following two statements are equivalent:

- (1) $N = w^e$ for a nonnegative integer e ,
- (2) $N * (\uparrow_v)_{\geq w} = w^e$ for a nonnegative integer e .

If $N = w^e$ then $N * (\uparrow_v)_{\geq w} = w^{e+\delta_{vw}}$. Thus (1) implies (2). We show that (2) implies (1). Suppose that (2) holds. Then N cannot be of form $\lambda u.N_1$ or $N_1 @_\ell N_2$, since neither $(\lambda u.N_1) * (\uparrow_v)_{\geq w}$ nor $(N_1 @_\ell N_2) * (\uparrow_v)_{\geq w}$ can equal w^e . Thus $N = u^c[\rho]$ for a name u , a nonnegative integer c , and a substitution ρ . By straightforward calculation, we have $N * (\uparrow_v)_{\geq w} = u^{c'}[\rho']$ for a nonnegative integer c' and a substitution ρ' with $\text{Lh}(\rho) \leq \text{Lh}(\rho')$. By the hypothesis (2), we have $u = w$ and $\rho = \text{id}$. Consequently we see that (2) implies (1).

By the above, the statements (1) and (2) are equivalent. By induction hypothesis on $\text{Lh}(\sigma)$, we have $\text{Lv}(\sigma_1 \circ \uparrow_v) = \text{Lv}(\sigma_1)$. Thus $\text{Lv}(\sigma \circ \uparrow_v) = \text{Lv}(\sigma)$ as follows:

$$\begin{aligned}
 \text{Lv}(\sigma \circ \uparrow_v) &= \text{Lv}({}^w\downarrow(N * (\uparrow_v)_{\geq w}) \cdot (\sigma_1 \circ \uparrow_v)) \\
 &= \begin{cases} \text{Lv}(\sigma_1 \circ \uparrow_v) & \text{if } N * (\uparrow_v)_{\geq w} = w^e \text{ for a nonnegative integer } e, \\ \max\{\text{Lv}(w), \text{Lv}(\sigma_1 \circ \uparrow_v)\} & \text{otherwise,} \end{cases} \\
 &= \begin{cases} \text{Lv}(\sigma_1) & \text{if } N = w^e \text{ for a nonnegative integer } e, \\ \max\{\text{Lv}(w), \text{Lv}(\sigma_1)\} & \text{otherwise,} \end{cases} \\
 &= \text{Lv}({}^w\downarrow(N) \cdot \sigma_1) = \text{Lv}(\sigma).
 \end{aligned}$$

3. Consider the case that $\sigma = \uparrow_w \cdot \sigma_1$. Then $\sigma \circ \uparrow_v = \uparrow_w \cdot (\sigma_1 \circ \uparrow_v)$. We see that $\sigma_1 \circ \uparrow_v$ is well-defined by induction hypothesis on $\text{Lh}(\sigma)$. Thus $\sigma \circ \uparrow_v$ is well-defined. We also have $\text{Lv}(\sigma_1 \circ \uparrow_v) = \text{Lv}(\sigma_1)$ by induction hypothesis on $\text{Lh}(\sigma)$. Hence $\text{Lv}(\sigma \circ \uparrow_v) = \text{Lv}(\uparrow_w \cdot (\sigma_1 \circ \uparrow_v)) = \text{Lv}(\sigma_1 \circ \uparrow_v) = \text{Lv}(\sigma_1) = \text{Lv}(\sigma)$.

By the above, we see that $\sigma \circ \uparrow_v$ is well-defined and $\text{Lv}(\sigma \circ \uparrow_v) = \text{Lv}(\sigma)$. Since $\uparrow_v(\sigma) = {}^v\downarrow(v) \cdot (\sigma \circ \uparrow_v)$, we see that $\uparrow_v(\sigma)$ is well-defined and $\text{Lv}(\uparrow_v(\sigma)) = \text{Lv}(\sigma \circ \uparrow_v) = \text{Lv}(\sigma)$. As a result, we see that $M_1 * \uparrow_v(\sigma)$ is well-defined by induction hypothesis, and thus $M * \sigma = \lambda v.(M_1 * \uparrow_v(\sigma))$ is well-defined.

- C. Consider the case that $M = M_1 @_\ell M_2$. Then $M * \sigma = (M_1 * \sigma) @_\ell (M_2 * \sigma_{\geq \ell})$. Since $\text{Lv}(\sigma_{\geq w}) \leq \text{Lv}(\sigma)$, we see that $M * \sigma$ is well-defined by induction hypothesis. \square

Corollary 3.2.10. Let σ and τ be substitutions. Then $\tau \circ \sigma$ is well-defined.

Proof. This proposition is shown in the proof of Proposition 3.2.9. \square

3.3 Reduction

Definition 3.3.1. Let \rightarrow_t be a binary relation on the set Ter of terms, and \rightarrow_s a binary relation on the set Sub of substitutions. The union of \rightarrow_t and \rightarrow_s , which is a binary relation on the set $Ter \cup Sub$, is a **reduction relation** if \rightarrow_t and \rightarrow_s satisfy the following conditions:

$$\begin{aligned} \sigma \cdot v \downarrow(M) \cdot \tau &\rightarrow_s \sigma \cdot v \downarrow(M') \cdot \tau && \text{if } M \rightarrow_t M', \\ v^d[\sigma] &\rightarrow_t v^d[\sigma'] && \text{if } \sigma \rightarrow_s \sigma', \\ \lambda v.M &\rightarrow_t \lambda v.M' && \text{if } M \rightarrow_t M', \\ M @_\ell N &\rightarrow_t M' @_\ell N && \text{if } M \rightarrow_t M', \\ N @_\ell M &\rightarrow_t N @_\ell M' && \text{if } M \rightarrow_t M', \end{aligned}$$

for all names v , nonnegative integers d , terms M, M', N , and substitutions σ, σ', τ .

Definition 3.3.2. We define **β -reduction** \rightarrow_β as the least reduction relation satisfying the following condition:

$$(\lambda v.M) @_\ell N \rightarrow_\beta M * v \downarrow(N) \quad \text{if } Lv(v) = \ell,$$

for all levels ℓ , names v , and terms M, N .

Notation. Let \rightarrow and \rightarrow' be binary relations on a set S . The composition of binary relations \rightarrow and \rightarrow' is denoted by $\rightarrow \cdot \rightarrow'$. The reflexive transitive closure of \rightarrow is denoted by \rightarrow^* . Namely, the composition $\rightarrow \cdot \rightarrow'$ and the reflexive transitive closure \rightarrow^* are defined as follows:

$$\begin{aligned} s_1 \rightarrow \cdot \rightarrow' s_2 &\text{ if and only if } s_1 \rightarrow s_3 \text{ and } s_3 \rightarrow' s_2 \text{ for some } s_3 \in S, \\ s_1 \rightarrow^* s_2 &\text{ if and only if } s_1 \rightarrow^n s_2 \text{ for some nonnegative integer } n, \end{aligned}$$

where \rightarrow^0 is the equality relation on S , and \rightarrow^{n+1} is defined as $\rightarrow^n \cdot \rightarrow$.

Remark 3.3.3. A term is said to be annotation-free if all indices in the term equal 0 and all substitutions in the term equal id . Consider the set Λ_1 of annotation-free terms consisting only of constructs of level 1. Then the set Λ_1 is just the set of terms in the ordinary lambda calculus under the assumption that we have infinitely many names of level 1 in the set \mathcal{N} of names. Furthermore, if we introduce α -renaming \rightarrow_α and η -reduction \rightarrow_η into λ^* as reduction relations satisfying the following conditions:

$$\lambda v.M \rightarrow_\alpha \lambda w.(M * (v \downarrow(w) \cdot \uparrow_w)) \text{ if } Lv(v) = Lv(w), \quad \lambda v.((M * \uparrow_v) @_{Lv(v)} v) \rightarrow_\eta M,$$

then β -reduction, α -renaming and η -reduction on the set Λ_1 coincide with the counterparts in the ordinary lambda calculus.

3.4 Equivalence

The β -reduction defined in Definition 3.3.2 is not strictly confluent. Consider a term $M = (\lambda x.((\lambda y.N) @_1 z)) @_1 y$ with names x, y, z of level 1 and name N of level 2. We can

obtain two distinct β -normal forms from M as follows:

$$M \rightarrow_{\beta}^* N[\downarrow^y(z) \cdot \downarrow^x(y)] \text{ and } M \rightarrow_{\beta}^* N[\downarrow^y(z) \cdot \downarrow^x(y) \cdot \uparrow_y \cdot \downarrow^y(z)].$$

The two substitutions $\sigma = \downarrow^y(z) \cdot \downarrow^x(y)$ and $\tau = \downarrow^y(z) \cdot \downarrow^x(y) \cdot \uparrow_y \cdot \downarrow^y(z)$ in the above are obviously distinct sequences of push-pop-elements. However, σ and τ have the same $\langle v, d \rangle$ -component for all names v and nonnegative integers d . We introduce an equivalence relation to equate such two substitutions. The equivalence relation leads the β -reduction to be confluent.

Definition 3.4.1. We define binary relation \simeq on the set $Ter \cup Sub$ as the least binary relation satisfying the following conditions:

$$\begin{aligned} \text{id} &\simeq \text{id}, \\ \sigma &\simeq \tau \quad \text{if } \sigma\langle v, d \rangle \simeq \tau\langle v, d \rangle \text{ for any name } v \text{ and nonnegative integer } d, \\ v^d[\sigma] &\simeq v^d[\tau] \quad \text{if } \sigma \simeq \tau, \\ \lambda v.M &\simeq \lambda v.N \quad \text{if } M \simeq N, \\ M_1 @_{\ell} M_2 &\simeq N_1 @_{\ell} N_2 \quad \text{if } M_1 \simeq N_1 \text{ and } M_2 \simeq N_2. \end{aligned}$$

We can easily see that binary relation \simeq is an equivalence relation by induction on height of terms and substitutions, and by the fact that $v^d \simeq v^d$ for all names v and nonnegative integers d .

Remark 3.4.2. Let M and M' be terms, σ, σ' and τ substitutions, v a name, and S a subset of the set \mathcal{N} of names. We have the following basic properties, which are used in the subsequent proofs:

- (1) If $\sigma \simeq \sigma'$, then $\sigma \upharpoonright S \simeq \sigma' \upharpoonright S$.
- (2) If $M \simeq M'$ and $\sigma \simeq \sigma'$, then $\downarrow^v(M) \cdot \sigma \simeq \downarrow^v(M') \cdot \sigma'$.
- (3) If $\sigma \simeq \sigma'$, then $\tau \cdot \sigma \simeq \tau \cdot \sigma'$.

We give another possible formalization of the equivalence to illustrate the connection between equivalent substitutions more concretely by providing canonical form of substitutions.

Definition 3.4.3. We define **ϵ -reduction** as the least reduction relation satisfying the following three conditions:

$$\begin{aligned} \sigma \cdot \uparrow_v \cdot \downarrow^v(M) \cdot \tau &\rightarrow_{\epsilon} \sigma \cdot \tau, \\ \sigma \cdot \downarrow^v(v^d) \cdot \uparrow_v^{d+1} &\rightarrow_{\epsilon} \sigma \cdot \uparrow_v^d, \\ \sigma \cdot \rho_1 \cdot \rho_2 \cdot \tau &\rightarrow_{\epsilon} \sigma \cdot \rho_2 \cdot \rho_1 \cdot \tau, \end{aligned}$$

for all names v, w , nonnegative integers d , terms M, N , substitutions σ, τ , and push-pop-elements $\rho_1 \in \{\downarrow^v(M), \uparrow_v\}$ and $\rho_2 \in \{\downarrow^w(N), \uparrow_w\}$ with $v \neq w$.

Definition 3.4.4. For each term M , we define $\#M$ as the number of push-pop-elements in M , namely, the number of up-arrows and down-arrows occurring in M . In the same way, for each substitution σ we define $\#\sigma$ as the number of up-arrows and down-arrows occurring in σ . A term M is said to be **canonical** if $\#M \leq \#M'$ for any term M' satisfying $M \rightarrow_{\epsilon}^* M'$. Similarly, a substitution σ is said to be canonical if $\#\sigma \leq \#\sigma'$ for any substitution σ' satisfying $\sigma \rightarrow_{\epsilon}^* \sigma'$.

Remark 3.4.5. The first and the second rules in Definition 3.4.3 of ε -reduction eliminate push-pop-elements, while the third just exchanges push-pop-elements of distinct names. Hence $M \rightarrow_\varepsilon M'$ implies $\#M \geq \#M'$. Note that for each term M , we can obtain a canonical term M' such that $M \rightarrow_\varepsilon^* M'$ by eliminating push-pop-elements as far as possible.

Lemma 3.4.6. A substitution σ is canonical if and only if for each name v we have

$$\sigma \upharpoonright \{v\} = v \downarrow (M_0) \cdot v \downarrow (M_1) \cdot \dots \cdot v \downarrow (M_{p-1}) \cdot \uparrow_v^q$$

for some nonnegative integers p, q and for some canonical terms M_0, M_1, \dots, M_{p-1} , where $M_{p-1} \neq v^{q-1}$ if $p > 0$ and $q > 0$.

Proof. Straightforward. \square

Lemma 3.4.7. Let M and N be terms. If $M \rightarrow_\varepsilon N$, then $M \simeq N$.

Proof. We use induction on $\text{Ht}(M)$. Suppose that $M \rightarrow_\varepsilon N$.

- A. Consider the case that $M = v^d[\sigma \cdot \uparrow_w \cdot w \downarrow (M_1) \cdot \tau]$ and $N = v^d[\sigma \cdot \tau]$. For any name u and nonnegative integer e , we have $(\uparrow_w \cdot w \downarrow (M_1) \cdot \tau) \langle u, e \rangle = \tau \langle u, e + \delta_{uw} - \delta_{uw} \rangle = \tau \langle u, e \rangle$. Thus $\uparrow_w \cdot w \downarrow (M_1) \cdot \tau \simeq \tau$, and hence $\sigma \cdot \uparrow_w \cdot w \downarrow (M_1) \cdot \tau \simeq \sigma \cdot \tau$. Therefore $M \simeq N$.
- B. Consider the case that $M = v^d[\sigma \cdot w \downarrow (w^e) \cdot \uparrow_w^{e+1}]$ and $N = v^d[\sigma \cdot \uparrow_w^e]$. Since we have $\uparrow_w^e \langle w, 0 \rangle = w^e$, the following holds for any name u and nonnegative integer c :

$$\begin{aligned} (w \downarrow (w^e) \cdot \uparrow_w^{e+1}) \langle u, c \rangle &= \begin{cases} w^e & \text{if } u = w \text{ and } c = 0, \\ \uparrow_w^{e+1} \langle u, c - \delta_{uw} \rangle & \text{otherwise,} \end{cases} \\ &= \uparrow_w^e \langle u, c \rangle \quad \text{regardless of } u \text{ and } c. \end{aligned}$$

Thus $w \downarrow (w^e) \cdot \uparrow_w^{e+1} \simeq \uparrow_w^e$, and $\sigma \cdot w \downarrow (w^e) \cdot \uparrow_w^{e+1} \simeq \sigma \cdot \uparrow_w^e$. Hence $M \simeq N$.

- C. Consider the case that $M = v^d[\sigma \cdot \rho_1 \cdot \rho_2 \cdot \tau]$ and $N = v^d[\sigma \cdot \rho_2 \cdot \rho_1 \cdot \tau]$ with $\rho_1 \in \{v \downarrow (M_1), \uparrow_{v_1}\}$ and $\rho_2 \in \{v \downarrow (M_2), \uparrow_{v_2}\}$ where $v_1 \neq v_2$. For all names w and nonnegative integers e , we have $(\sigma \cdot \rho_1 \cdot \rho_2 \cdot \tau) \langle w, e \rangle = ((\sigma \cdot \rho_1 \cdot \rho_2 \cdot \tau) \upharpoonright \{w\}) \langle w, e \rangle = ((\sigma \cdot \rho_2 \cdot \rho_1 \cdot \tau) \upharpoonright \{w\}) \langle w, e \rangle = (\sigma \cdot \rho_2 \cdot \rho_1 \cdot \tau) \langle w, e \rangle$. Thus $\sigma \cdot \rho_1 \cdot \rho_2 \cdot \tau \simeq \sigma \cdot \rho_2 \cdot \rho_1 \cdot \tau$, and hence $M \simeq N$.
- D. Consider the following cases: (1) $M = v^d[\sigma \cdot w \downarrow (M_1) \cdot \tau]$ and $N = v^d[\sigma \cdot w \downarrow (N_1) \cdot \tau]$ with $M_1 \rightarrow_\varepsilon N_1$, (2) $M = \lambda v. M_1$ and $N = \lambda v. N_1$ with $M_1 \rightarrow_\varepsilon N_1$, (3) $M = M_1 @_\ell M_2$ and $N = N_1 @_\ell M_2$ with $M_1 \rightarrow_\varepsilon N_1$, (4) $M = M_2 @_\ell M_1$ and $N = M_2 @_\ell N_1$ with $M_1 \rightarrow_\varepsilon N_1$. In each case, we have $M \simeq N$ since $M_1 \simeq N_1$ by induction hypothesis. \square

Corollary 3.4.8. Let σ and σ' be substitutions. If $\sigma \rightarrow_\varepsilon \sigma'$, then $\sigma \simeq \sigma'$.

Proof. This proposition is shown in the proof of Lemma 3.4.7. \square

Lemma 3.4.9. If M and N are canonical terms and $M \simeq N$, then $M \rightarrow_\varepsilon^* N$.

Proof. We use induction on $\text{Ht}(M) + \text{Ht}(N)$. Suppose that M and N be canonical terms such that $M \simeq N$.

- A. Consider the case that $M = v^d[\sigma]$ and $N = v^d[\tau]$ with $\sigma \simeq \tau$. Let w be a name. Since σ and τ are canonical in this case, the following holds for some nonnegative integers p, q, r, s , and canonical terms $M_0, M_1, \dots, M_{p-1}, N_0, N_1, \dots, N_{r-1}$, by Lemma 3.4.6:

$$\begin{aligned} \sigma \upharpoonright \{w\} &= w \downarrow (M_0) \cdot w \downarrow (M_1) \cdot \dots \cdot w \downarrow (M_{p-1}) \cdot \uparrow_w^q, \\ \tau \upharpoonright \{w\} &= w \downarrow (N_0) \cdot w \downarrow (N_1) \cdot \dots \cdot w \downarrow (N_{r-1}) \cdot \uparrow_w^s, \end{aligned}$$

where $M_{p-1} \neq w^{q-1}$ if $p > 0$ and $q > 0$, and $N_{r-1} \neq w^{s-1}$ if $r > 0$ and $s > 0$.

We prove that $p = r$ and $q = s$ by contradiction. Suppose that $p > r$. Then $w^q = \sigma\langle w, p \rangle \simeq \tau\langle w, p \rangle = w^{p-r+s}$. Hence $q = p - r + s$. We also have $M_{p-1} = \sigma\langle w, p-1 \rangle \simeq \tau\langle w, p-1 \rangle = w^{p-1-r+s} = w^{q-1}$. Thus $M_{p-1} \simeq w^{q-1}$. Since $\text{Ht}(w^{q-1}) = 0$ and $\text{Ht}(M_{p-1}) < \text{Ht}(M)$, we have $\text{Ht}(w^{q-1}) + \text{Ht}(M_{p-1}) < \text{Ht}(M) + \text{Ht}(N)$. Consequently $w^{q-1} \rightarrow_{\epsilon}^* M_{p-1}$ by induction hypothesis, since w^{q-1} and M_{p-1} are canonical terms. However, $w^{q-1} \rightarrow_{\epsilon}^* M_{p-1}$ implies $M_{p-1} = w^{q-1}$, which contradicts the hypothesis that $M_{p-1} \neq w^{q-1}$. As a result, we cannot have $p > r$, and in the same way we cannot have $r > p$. Therefore $p = r$. Since $w^q = \sigma\langle w, p \rangle \simeq \tau\langle w, r \rangle = w^s$, we also have $q = s$. For each $i = 0, 1, \dots, p-1$, we have $M_i \rightarrow_{\epsilon}^* N_i$ by induction hypothesis since we have $M_i = \sigma\langle w, i \rangle \simeq \tau\langle w, i \rangle = N_i$.

By the above, we see that $\sigma \upharpoonright \{w\} \rightarrow_{\epsilon}^* \tau \upharpoonright \{w\}$ for any name w . Lastly, we show that $\sigma \rightarrow_{\epsilon}^* \tau$. Let w_1, w_2, \dots, w_n be distinct names such that $\{w \in \mathcal{N} \mid \sigma \upharpoonright \{w\} \neq \text{id}\} = \{w_1, w_2, \dots, w_n\}$. Then we have the following:

$$\begin{aligned}
& \sigma \rightarrow_{\epsilon}^* \sigma \upharpoonright \{w_n\} \cdot \dots \cdot \sigma \upharpoonright \{w_2\} \cdot \sigma \upharpoonright \{w_1\} && \text{by exchanging push-pop-elements,} \\
& \rightarrow_{\epsilon}^* \sigma \upharpoonright \{w_n\} \cdot \dots \cdot \sigma \upharpoonright \{w_2\} \cdot \tau \upharpoonright \{w_1\} && \text{since } \sigma \upharpoonright \{w_1\} \rightarrow_{\epsilon}^* \tau \upharpoonright \{w_1\}, \\
& \rightarrow_{\epsilon}^* \tau \upharpoonright \{w_1\} \cdot \sigma \upharpoonright \{w_n\} \cdot \dots \cdot \sigma \upharpoonright \{w_2\} && \text{by exchanging push-pop-elements,} \\
& \rightarrow_{\epsilon}^* \tau \upharpoonright \{w_1\} \cdot \sigma \upharpoonright \{w_n\} \cdot \dots \cdot \tau \upharpoonright \{w_2\} && \text{since } \sigma \upharpoonright \{w_2\} \rightarrow_{\epsilon}^* \tau \upharpoonright \{w_2\}, \\
& \rightarrow_{\epsilon}^* \tau \upharpoonright \{w_2\} \cdot \tau \upharpoonright \{w_1\} \cdot \sigma \upharpoonright \{w_n\} \cdot \dots \cdot \sigma \upharpoonright \{w_3\} && \text{by exchanging push-pop-elements,} \\
& \rightarrow_{\epsilon}^* \tau \upharpoonright \{w_n\} \cdot \dots \cdot \tau \upharpoonright \{w_2\} \cdot \tau \upharpoonright \{w_1\} && \text{by iterating the above procedure,} \\
& \rightarrow_{\epsilon}^* \tau && \text{by exchanging push-pop-elements.}
\end{aligned}$$

As a result, we have $M \rightarrow_{\epsilon}^* N$.

- B. Consider the case that $M = \lambda v. M_1$ and $N = \lambda v. N_1$ with $M_1 \simeq N_1$. Then $M_1 \rightarrow_{\epsilon}^* N_1$ by induction hypothesis, and hence $M \rightarrow_{\epsilon}^* N$.
- C. Consider the case that $M = M_1 @_{\ell} M_2$ and $N = N_1 @_{\ell} N_2$ with $M_i \simeq N_i$ for $i = 1, 2$. Then $M_i \rightarrow_{\epsilon}^* N_i$ by induction hypothesis for $i = 1, 2$. Thus $M \rightarrow_{\epsilon}^* N$. \square

Corollary 3.4.10. If σ and τ are canonical substitutions and $\sigma \simeq \tau$, then $\sigma \rightarrow_{\epsilon}^* \tau$.

Proof. This proposition is shown in the proof of Lemma 3.4.9.

Proposition 3.4.11. Let M and N be terms. Then the following holds:

$$M \simeq N \quad \text{if and only if} \quad M \rightarrow_{\epsilon}^* \cdot \leftarrow_{\epsilon}^* N.$$

Proof for ‘if’. If $M \rightarrow_{\epsilon}^* \cdot \leftarrow_{\epsilon}^* N$, then $M \simeq N$ by Lemma 3.4.7.

Proof for ‘only if’. Suppose that $M \simeq N$. We have some canonical terms M' and N' such that $M \rightarrow_{\epsilon}^* M'$ and $N \rightarrow_{\epsilon}^* N'$. By Lemma 3.4.7, we have $M \simeq M'$ and $N \simeq N'$, and hence $M' \simeq N'$. By Lemma 3.4.9, we have $M' \rightarrow_{\epsilon}^* N'$. Consequently, we have $M \rightarrow_{\epsilon}^* M' \rightarrow_{\epsilon}^* N' \leftarrow_{\epsilon}^* N$. \square

Corollary 3.4.12. Let σ and τ be substitutions. Then the following holds:

$$\sigma \simeq \tau \quad \text{if and only if} \quad \sigma \rightarrow_{\epsilon}^* \cdot \leftarrow_{\epsilon}^* \tau.$$

Proof. This proposition is shown in the same way as Proposition 3.4.11. \square

3.5 Algebraic Properties

As stated at the end of this section, we demonstrate that action of substitutions on terms and composition of substitutions preserve equivalence, and the set Sub of substitutions amounts to a monoid acting on the set Ter of terms up to equivalence.

Lemma 3.5.1. Let σ and τ be substitutions, and S a subset of the set \mathcal{N} of names. We have $(\sigma \circ \tau) \upharpoonright S = ((\sigma \upharpoonright S) \circ \tau) \upharpoonright S$.

Proof. Straightforward by induction on $Lh(\sigma)$.

Lemma 3.5.2. Let σ and τ be substitutions, ℓ a level. Then $(\sigma \circ \tau)_{\geq \ell} = \sigma_{\geq \ell} \circ \tau_{\geq \ell}$.

Proof. We use induction on $Lh(\sigma)$.

- A. Consider the case that $\sigma = \text{id}$. Then $(\sigma \circ \tau)_{\geq \ell} = \tau_{\geq \ell} = \sigma_{\geq \ell} \circ \tau_{\geq \ell}$.
- B. Consider the case that $\sigma = v \downarrow (M) \cdot \sigma_1$. By induction hypothesis, we have $(\sigma_1 \circ \tau)_{\geq \ell} = (\sigma_1)_{\geq \ell} \circ \tau_{\geq \ell}$. If $Lv(v) \geq \ell$, then $\tau_{\geq v} = (\tau_{\geq \ell})_{\geq v}$ and hence the following holds:

$$\begin{aligned} (\sigma \circ \tau)_{\geq \ell} &= (v \downarrow (M * \tau_{\geq v}) \cdot (\sigma_1 \circ \tau))_{\geq \ell} = v \downarrow (M * \tau_{\geq v}) \cdot (\sigma_1 \circ \tau)_{\geq \ell} \\ &= v \downarrow (M * (\tau_{\geq \ell})_{\geq v}) \cdot ((\sigma_1)_{\geq \ell} \circ \tau_{\geq \ell}) \\ &= (v \downarrow (M) \cdot (\sigma_1)_{\geq \ell}) \circ \tau_{\geq \ell} = (v \downarrow (M) \cdot \sigma_1)_{\geq \ell} \circ \tau_{\geq \ell} = \sigma_{\geq \ell} \circ \tau_{\geq \ell}. \end{aligned}$$

If $Lv(v) < \ell$, then we have $(\sigma \circ \tau)_{\geq \ell} = (v \downarrow (M * \tau_{\geq v}) \cdot (\sigma_1 \circ \tau))_{\geq \ell} = (\sigma_1 \circ \tau)_{\geq \ell} = (\sigma_1)_{\geq \ell} \circ \tau_{\geq \ell} = (v \downarrow (M) \cdot \sigma_1)_{\geq \ell} \circ \tau_{\geq \ell} = \sigma_{\geq \ell} \circ \tau_{\geq \ell}$.

- C. Consider the case that $\sigma = \uparrow_v \cdot \sigma_1$. By induction hypothesis, we have $(\sigma_1 \circ \tau)_{\geq \ell} = (\sigma_1)_{\geq \ell} \circ \tau_{\geq \ell}$. Let $\delta = 1$ if $Lv(v) \geq \ell$, or otherwise let $\delta = 0$. Then $(\sigma \circ \tau)_{\geq \ell} = (\uparrow_v \cdot (\sigma_1 \circ \tau))_{\geq \ell} = \uparrow_v^\delta \cdot (\sigma_1 \circ \tau)_{\geq \ell} = \uparrow_v^\delta \cdot ((\sigma_1)_{\geq \ell} \circ \tau_{\geq \ell}) = (\uparrow_v^\delta \cdot (\sigma_1)_{\geq \ell}) \circ \tau_{\geq \ell} = (\uparrow_v \cdot \sigma_1)_{\geq \ell} \circ \tau_{\geq \ell} = \sigma_{\geq \ell} \circ \tau_{\geq \ell}$. \square

Lemma 3.5.3. Let $P(\sigma)$ be the following proposition for substitutions σ :

$P(\sigma)$: For each name v , there exists a nonnegative integer p such that $\sigma \upharpoonright \{v\} = \pi_{v,p}$,

where $\pi_{v,p}$ denotes $v \downarrow (v^0) \cdot v \downarrow (v^1) \cdot \dots \cdot v \downarrow (v^{p-1}) \cdot \uparrow_v^p$. Suppose that a substitution σ satisfy $P(\sigma)$. Then the following three properties hold:

- (1) We have $\sigma \langle v, d \rangle = v^d$ for all names v and nonnegative integers d , and thus $\sigma \simeq \text{id}$.
- (2) Proposition $P(\uparrow_v(\sigma))$ holds for each name v .
- (3) Proposition $P(\sigma \upharpoonright S)$ holds for any subset S of the set \mathcal{N} of names.

Proof for (1). Let v be a name, and p a nonnegative integer such that $\sigma \upharpoonright \{v\} = \pi_{v,p}$. By straightforward calculation, we see that $\pi_{v,p} \langle v, d \rangle = v^d$ for each nonnegative integer d . Thus $\sigma \langle v, d \rangle = (\sigma \upharpoonright \{v\}) \langle v, d \rangle = \pi_{v,p} \langle v, d \rangle = v^d$.

Proof for (2). Let w be a name, and p a nonnegative integer such that $\sigma \upharpoonright \{w\} = \pi_{w,p}$. Then the following holds:

$$\begin{aligned} (\uparrow_v(\sigma)) \upharpoonright \{w\} &= (v \downarrow (v) \cdot (\sigma \circ \uparrow_v)) \upharpoonright \{w\} = v \downarrow (v)^{\delta_{vw}} \cdot (\sigma \circ \uparrow_v) \upharpoonright \{w\} \\ &= v \downarrow (v)^{\delta_{vw}} \cdot (\sigma \upharpoonright \{w\} \circ \uparrow_v) \upharpoonright \{w\} \quad \text{by Lemma 3.5.1,} \\ &= v \downarrow (v)^{\delta_{vw}} \cdot ((w \downarrow (w^0) \cdot w \downarrow (w^1) \cdot \dots \cdot w \downarrow (w^{p-1}) \cdot \uparrow_w^p) \circ \uparrow_v) \upharpoonright \{w\} \\ &= v \downarrow (v)^{\delta_{vw}} \cdot (w \downarrow (w^0 * (\uparrow_v)_{\geq w}) \cdot w \downarrow (w^1 * (\uparrow_v)_{\geq w}) \cdot \dots \cdot w \downarrow (w^{p-1} * (\uparrow_v)_{\geq w}) \cdot \uparrow_w^p \cdot \uparrow_v) \upharpoonright \{w\} \\ &= v \downarrow (v)^{\delta_{vw}} \cdot w \downarrow (w^{0+\delta_{vw}}) \cdot w \downarrow (w^{1+\delta_{vw}}) \cdot \dots \cdot w \downarrow (w^{p-1+\delta_{vw}}) \cdot \uparrow_w^p \cdot \uparrow_v^{\delta_{vw}} = \pi_{w,p+\delta_{vw}}. \end{aligned}$$

where note that $w^i * (\uparrow_v)_{\geq w} = \uparrow_v \langle w, i \rangle * \text{id} = w^{i+\delta_{vw}}$ for each nonnegative integer i .

Proof for (3). Let v be a name, and p a nonnegative integer such that $\sigma \upharpoonright \{v\} = \pi_{v,p}$. Then $(\sigma \upharpoonright S) \upharpoonright \{v\} = \pi_{v,p}$ if $v \in S$, or otherwise $(\sigma \upharpoonright S) \upharpoonright \{v\} = \text{id} = \pi_{v,0}$. \square

Lemma 3.5.4. Let M be a term and σ a substitution. If for each name v there exists a nonnegative integer p such that $\sigma \upharpoonright \{v\} = v \downarrow (v^0) \cdot v \downarrow (v^1) \cdot \dots \cdot v \downarrow (v^{p-1}) \cdot \uparrow_v^p$, then we have $M * \sigma \simeq M$.

Proof. We prove this proposition by induction on $\text{Ht}(M)$. Suppose that σ satisfy the hypothesis.

- A. Consider the case that $M = v^d[\tau]$. We show that $\tau \circ \sigma \simeq \tau$ by induction on $\text{Lh}(\tau)$.
 - 1. Consider the case that $\tau = \text{id}$. Then $\tau \circ \sigma = \sigma \simeq \text{id} = \tau$ by Lemma 3.5.3(1).
 - 2. Consider the case that $\tau = w \downarrow (M_1) \cdot \tau_1$. By induction hypothesis on $\text{Ht}(M)$ and by Lemma 3.5.3(3), we have $M_1 * \sigma_{\geq w} \simeq M_1$. By induction hypothesis on $\text{Lh}(\sigma)$, we have $\tau_1 \circ \sigma \simeq \tau_1$. Thus $\tau \circ \sigma = w \downarrow (M_1 * \sigma_{\geq w}) \cdot (\tau_1 \circ \sigma) \simeq w \downarrow (M_1) \cdot \tau_1 = \tau$.
 - 3. Consider the case that $\tau = \uparrow_w \cdot \tau_1$. By induction hypothesis on $\text{Lh}(\sigma)$, we have $\tau_1 \circ \sigma \simeq \tau_1$. Hence $\tau \circ \sigma = \uparrow_w \cdot (\tau_1 \circ \sigma) \simeq \uparrow_w \cdot \tau_1 = \tau$.
 By the above, we have $\tau \circ \sigma \simeq \tau$. We also have $\sigma \langle v, d \rangle = v^d$ by Lemma 3.5.3(1). Thus $M * \sigma = \sigma \langle v, d \rangle * (\tau \circ \sigma)_{<v} = v^d[(\tau \circ \sigma)_{<v}] \simeq v^d[\tau_{<v}] = v^d[\tau] = M$, where note that $\tau_{<v} = \tau$ since $v^d[\tau]$ is a term and hence $\tau \in \text{Sub}_{\text{Lv}(v)}$ by definition.
- B. Consider the case that $M = \lambda v. M_1$. By induction hypothesis and Lemma 3.5.3(2), we have $M_1 * \uparrow_v(\sigma) \simeq M_1$. Thus $M * \sigma = \lambda v. (M_1 * \uparrow_v(\sigma)) \simeq \lambda v. M_1 = M$.
- C. Consider the case that $M = M_1 @_{\ell} M_2$. By induction hypothesis and Lemma 3.5.3(3), we have $M_1 * \sigma \simeq M_1$ and $M_2 * \sigma_{\geq \ell} \simeq M_2$. Hence $M * \sigma = (M_1 * \sigma) @_{\ell} (M_2 * \sigma_{\geq \ell}) \simeq M_1 @_{\ell} M_2 = M$. \square

Proposition 3.5.5. Let M be a term. Then the following holds:

$$M * \text{id} \simeq M.$$

Proof. This statement is a special case of Lemma 3.5.4. \square

Lemma 3.5.6. Let σ and τ be substitutions, v a name, and d a nonnegative integer. Then the following holds:

$$(\tau \circ \sigma) \langle v, d \rangle \simeq \tau \langle v, d \rangle * \sigma_{\geq v}.$$

Proof. We use induction on $\text{Lh}(\tau)$.

- A. Consider the case that $\tau = \text{id}$. Then $(\tau \circ \sigma) \langle v, d \rangle = \sigma \langle v, d \rangle \simeq v^d * \sigma_{\geq v} = \tau \langle v, d \rangle * \sigma_{\geq v}$.
- B. Consider the case that $\tau = w \downarrow (M) \cdot \tau_1$. Then, by induction hypothesis, we have $(\tau_1 \circ \sigma) \langle v, d - \delta_{vw} \rangle \simeq \tau_1 \langle v, d - \delta_{vw} \rangle * \sigma_{\geq v}$. Thus the following holds:

$$\begin{aligned} (\tau \circ \sigma) \langle v, d \rangle &= (w \downarrow (M * \sigma_{\geq w}) \cdot (\tau_1 \circ \sigma)) \langle v, d \rangle \\ &= \begin{cases} M * \sigma_{\geq w} & \text{if } v = w \text{ and } d = 0, \\ (\tau_1 \circ \sigma) \langle v, d - \delta_{vw} \rangle \simeq \tau_1 \langle v, d - \delta_{vw} \rangle * \sigma_{\geq v} & \text{otherwise,} \end{cases} \\ &= \tau \langle v, d \rangle * \sigma_{\geq v} \quad \text{regardless of whether } v = w \text{ and } d = 0, \end{aligned}$$

since $\tau \langle v, d \rangle = M$ if $v = w$ and $d = 0$, or otherwise $\tau \langle v, d \rangle = \tau_1 \langle v, d - \delta_{vw} \rangle$.

- C. Consider the case that $\tau = \uparrow_w \cdot \tau_1$. Then $(\tau_1 \circ \sigma) \langle v, d + \delta_{vw} \rangle \simeq \tau_1 \langle v, d + \delta_{vw} \rangle * \sigma_{\geq v}$ by induction hypothesis. Thus $(\tau \circ \sigma) \langle v, d \rangle = (\tau_1 \circ \sigma) \langle v, d + \delta_{vw} \rangle \simeq \tau_1 \langle v, d + \delta_{vw} \rangle * \sigma_{\geq v} = \tau \langle v, d \rangle * \sigma_{\geq v}$. \square

Proposition 3.5.7. Let M and M' be terms, σ and σ' substitutions. The following holds:

$$\text{If } M \simeq M' \text{ and } \sigma \simeq \sigma', \text{ then } M * \sigma \simeq M' * \sigma'.$$

Proof. We show this proposition by induction on lexicographic ordering of pairs $\langle \max\{\text{Lv}(\sigma), \text{Lv}(\sigma')\}, \text{Ht}(M) + \text{Ht}(M') \rangle$. Suppose that $M \simeq M'$ and $\sigma \simeq \sigma'$.

A. Consider the case that $M = v^d[\tau]$. Since $M \simeq M'$, we have $M' = v^d[\tau']$ for a substitution τ' with $\tau \simeq \tau'$. First, we show that $\tau \circ \sigma \simeq \tau' \circ \sigma'$. Let w be a name and e a nonnegative integer.

1. Consider the case that $\max\{\text{Ht}(M), \text{Ht}(M')\} = 0$. Then both τ and τ' consist only of pop-elements, and hence $\tau\langle w, e \rangle = w^c$ and $\tau'\langle w, e \rangle = w^{c'}$ for some nonnegative integers c and c' . Since $\tau \simeq \tau'$, we have $w^c = \tau\langle w, e \rangle \simeq \tau'\langle w, e \rangle = w^{c'}$ and hence $c = c'$. Thus $(\tau \circ \sigma)\langle w, e \rangle \simeq \tau\langle w, e \rangle * \sigma_{\geq w} = w^c * \sigma_{\geq w} \simeq \sigma\langle w, c \rangle \simeq \sigma'\langle w, c \rangle$. We also have $(\tau' \circ \sigma')\langle w, e \rangle \simeq \tau'\langle w, e \rangle * \sigma'_{\geq w} = w^{c'} * \sigma'_{\geq w} \simeq \sigma'\langle w, c' \rangle$. Therefore $(\tau \circ \sigma)\langle w, e \rangle \simeq (\tau' \circ \sigma')\langle w, e \rangle$.
2. Consider the case that $\max\{\text{Ht}(M), \text{Ht}(M')\} > 0$. Then $\text{Ht}(\tau\langle w, e \rangle) < \text{Ht}(M)$ or $\text{Ht}(\tau'\langle w, e \rangle) < \text{Ht}(M')$. Hence $\tau\langle w, e \rangle * \sigma_{\geq w} \simeq \tau'\langle w, e \rangle * \sigma'_{\geq w}$ by induction hypothesis. Thus $(\tau \circ \sigma)\langle w, e \rangle \simeq \tau\langle w, e \rangle * \sigma_{\geq w} \simeq \tau'\langle w, e \rangle * \sigma'_{\geq w} \simeq (\tau' \circ \sigma')\langle w, e \rangle$. By the above, we have $(\tau \circ \sigma)\langle w, e \rangle \simeq (\tau' \circ \sigma')\langle w, e \rangle$ for any name w and nonnegative integer e . Thus $\tau \circ \sigma \simeq \tau' \circ \sigma'$.

Next, we show that $M * \sigma \simeq M' * \sigma'$.

1. Consider the case that $\max\{\text{Lv}(\sigma), \text{Lv}(\sigma')\} < \text{Lv}(v)$. Then $\sigma\langle v, d \rangle = v^c$ and $\sigma'\langle v, d \rangle = v^{c'}$ for some nonnegative integers c and c' . We also have $v^c = \sigma\langle v, d \rangle \simeq \sigma'\langle v, d \rangle = v^{c'}$, and thus $c = c'$. Hence $M * \sigma = \sigma\langle v, d \rangle * (\tau \circ \sigma)_{<v} = v^c[(\tau \circ \sigma)_{<v}] \simeq v^c[(\tau' \circ \sigma')_{<v}] = \sigma'\langle v, d \rangle * (\tau' \circ \sigma')_{<v} = M' * \sigma'$.
2. Consider the case that $\max\{\text{Lv}(\sigma), \text{Lv}(\sigma')\} \geq \text{Lv}(v)$. Then $\sigma\langle v, d \rangle * (\tau \circ \sigma)_{<v} \simeq \sigma'\langle v, d \rangle * (\tau' \circ \sigma')_{<v}$ by induction hypothesis. Thus $M * \sigma = \sigma\langle v, d \rangle * (\tau \circ \sigma)_{<v} \simeq \sigma'\langle v, d \rangle * (\tau' \circ \sigma')_{<v} = M' * \sigma'$.

By the above, we have $M * \sigma \simeq M' * \sigma'$.

B. Consider the case that $M = \lambda v.M_1$. Since $M \simeq M'$, we have $M' = \lambda v.M'_1$ for a term M'_1 with $M_1 \simeq M'_1$. We show below that $\sigma \circ \uparrow_v \simeq \sigma' \circ \uparrow_v$. Let w be a name and d a nonnegative integer.

1. Consider the case that $\max\{\text{Lv}(\sigma), \text{Lv}(\sigma')\} = 0$. Then we have $\sigma\langle w, d \rangle = w^e$ and $\sigma'\langle w, d \rangle = w^{e'}$ for some nonnegative integers e and e' . We also have $w^e = \sigma\langle w, d \rangle \simeq \sigma'\langle w, d \rangle = w^{e'}$. Hence $e = e'$ and $\sigma\langle w, d \rangle = \sigma'\langle w, d \rangle$. Thus we have $(\sigma \circ \uparrow_v)\langle w, d \rangle \simeq \sigma\langle w, d \rangle * (\uparrow_v)_{\geq w} = \sigma'\langle w, d \rangle * (\uparrow_v)_{\geq w} \simeq (\sigma' \circ \uparrow_v)\langle w, d \rangle$.
2. Consider the case that $\max\{\text{Lv}(\sigma), \text{Lv}(\sigma')\} > 0$. Since $\text{Lv}((\uparrow_v)_{\geq w}) = 0$, we have $\sigma\langle w, d \rangle * (\uparrow_v)_{\geq w} \simeq \sigma'\langle w, d \rangle * (\uparrow_v)_{\geq w}$ by induction hypothesis. Thus $(\sigma \circ \uparrow_v)\langle w, d \rangle \simeq \sigma\langle w, d \rangle * (\uparrow_v)_{\geq w} \simeq \sigma'\langle w, d \rangle * (\uparrow_v)_{\geq w} \simeq (\sigma' \circ \uparrow_v)\langle w, d \rangle$.

By the above, we have $(\sigma \circ \uparrow_v)\langle w, d \rangle \simeq (\sigma' \circ \uparrow_v)\langle w, d \rangle$ for any name w and nonnegative integer d . Thus $\sigma \circ \uparrow_v \simeq \sigma' \circ \uparrow_v$. Consequently, we have $\uparrow_v(\sigma) = v\downarrow(v) \cdot (\sigma \circ \uparrow_v) \simeq v\downarrow(v) \cdot (\sigma' \circ \uparrow_v) = \uparrow_v(\sigma')$. Since $\text{Lv}(\uparrow_v(\sigma)) = \text{Lv}(\sigma)$ and $\text{Lv}(\uparrow_v(\sigma')) = \text{Lv}(\sigma')$, we have $M_1 * \uparrow_v(\sigma) \simeq M'_1 * \uparrow_v(\sigma')$ by induction hypothesis. Thus $M * \sigma = \lambda v.(M_1 * \uparrow_v(\sigma)) \simeq \lambda v.(M'_1 * \uparrow_v(\sigma')) = M' * \sigma'$.

C. Consider the case that $M = M_1 @_{\ell} M_2$. Since $M \simeq M'$, we have $M' = M'_1 @_{\ell} M'_2$ for some terms M'_1 and M'_2 with $M_i \simeq M'_i$ for $i = 1, 2$. By induction hypothesis, we have $M_1 * \sigma \simeq M'_1 * \sigma'$ and $M_2 * \sigma_{\geq \ell} \simeq M'_2 * \sigma'_{\geq \ell}$. Thus $M * \sigma = (M_1 * \sigma) @_{\ell} (M_2 * \sigma_{\geq \ell}) \simeq (M'_1 * \sigma') @_{\ell} (M'_2 * \sigma'_{\geq \ell}) = M' * \sigma'$. \square

Corollary 3.5.8. Let σ, σ', τ and τ' be substitutions. Then the following holds:

$$\text{If } \sigma \simeq \sigma' \text{ and } \tau \simeq \tau', \text{ then } \tau \circ \sigma \simeq \tau' \circ \sigma'.$$

Proof. This proposition is shown in the proof of Proposition 3.5.7. \square

Lemma 3.5.9. We have $\sigma_{<\ell} \circ \tau \simeq \tau_{\geq\ell} \circ (\sigma \circ \tau)_{<\ell}$ for all substitutions σ, τ , and levels ℓ .

Proof. Let v be a name, and d a nonnegative integer.

- A. Consider the case that $\text{Lv}(v) \geq \ell$. Then $(\sigma_{<\ell} \circ \tau)\langle v, d \rangle \simeq \sigma_{<\ell}\langle v, d \rangle * \tau_{\geq v} = v^d * \tau_{\geq v} \simeq \tau\langle v, d \rangle$. We also have $(\tau_{\geq\ell} \circ (\sigma \circ \tau)_{<\ell})\langle v, d \rangle \simeq \tau_{\geq\ell}\langle v, d \rangle * ((\sigma \circ \tau)_{<\ell})_{\geq v} = \tau\langle v, d \rangle * \text{id} \simeq \tau\langle v, d \rangle$. Hence $(\sigma_{<\ell} \circ \tau)\langle v, d \rangle \simeq (\tau_{\geq\ell} \circ (\sigma \circ \tau)_{<\ell})\langle v, d \rangle$.
- B. Consider the case that $\text{Lv}(v) < \ell$. Then we have $(\sigma_{<\ell} \circ \tau)\langle v, d \rangle \simeq \sigma_{<\ell}\langle v, d \rangle * \tau_{\geq v} = \sigma\langle v, d \rangle * \tau_{\geq v} \simeq (\sigma \circ \tau)\langle v, d \rangle$. Also $(\tau_{\geq\ell} \circ (\sigma \circ \tau)_{<\ell})\langle v, d \rangle \simeq \tau_{\geq\ell}\langle v, d \rangle * ((\sigma \circ \tau)_{<\ell})_{\geq v} = v^d * ((\sigma \circ \tau)_{<\ell})_{\geq v} \simeq ((\sigma \circ \tau)_{<\ell})\langle v, d \rangle = (\sigma \circ \tau)\langle v, d \rangle$. Therefore $(\sigma_{<\ell} \circ \tau)\langle v, d \rangle \simeq (\tau_{\geq\ell} \circ (\sigma \circ \tau)_{<\ell})\langle v, d \rangle$.

By the above, we see that $(\sigma_{<\ell} \circ \tau)\langle v, d \rangle \simeq (\tau_{\geq\ell} \circ (\sigma \circ \tau)_{<\ell})\langle v, d \rangle$ for any name v and nonnegative integer d . Consequently $\sigma_{<\ell} \circ \tau \simeq \tau_{\geq\ell} \circ (\sigma \circ \tau)_{<\ell}$. \square

Proposition 3.5.10. Let M be a term, σ and τ substitutions. We have the following:

$$(M * \sigma) * \tau \simeq M * (\sigma \circ \tau).$$

Proof. We use induction on lexicographic ordering of pairs $\langle \text{Lv}(\sigma) + \text{Lv}(\tau), \text{Ht}(M) \rangle$.

- A. Consider the case that $M = v^d[\rho]$. First we show that $(\rho \circ \sigma) \circ \tau \simeq \rho \circ (\sigma \circ \tau)$.
1. Consider the case that $\text{Ht}(M) = 0$. Then ρ consists only of pop-elements. Thus $(\rho \circ \sigma) \circ \tau = (\rho \cdot \sigma) \circ \tau = \rho \cdot (\sigma \circ \tau) = \rho \circ (\sigma \circ \tau)$.
 2. Consider the case that $\text{Ht}(M) > 0$. Suppose that w be a name and e a nonnegative integer. Then $((\rho \circ \sigma) \circ \tau)\langle w, e \rangle \simeq (\rho \circ \sigma)\langle w, e \rangle * \tau_{\geq w} \simeq (\rho\langle w, e \rangle * \sigma_{\geq w}) * \tau_{\geq w} \simeq \rho\langle w, e \rangle * (\sigma_{\geq w} \circ \tau_{\geq w})$ by induction hypothesis. We also have $(\rho \circ (\sigma \circ \tau))\langle w, e \rangle \simeq \rho\langle w, e \rangle * (\sigma \circ \tau)_{\geq w} = \rho\langle w, e \rangle * (\sigma_{\geq w} \circ \tau_{\geq w})$ by Lemma 3.5.2. Thus $(\rho \circ \sigma) \circ \tau \simeq \rho \circ (\sigma \circ \tau)$.

Next, we show that $(M * \sigma) * \tau \simeq M * (\sigma \circ \tau)$.

1. Consider the case that $\text{Lv}(\sigma) < \text{Lv}(v)$. Then $\sigma\langle v, d \rangle = v^e$ for some nonnegative integer e . We have $(M * \sigma) * \tau = (\sigma\langle v, d \rangle * (\rho \circ \sigma)_{<v}) * \tau = v^e[(\rho \circ \sigma)_{<v}] * \tau = \tau\langle v, e \rangle * ((\rho \circ \sigma)_{<v} \circ \tau)_{<v} = \tau\langle v, e \rangle * ((\rho \circ \sigma) \circ \tau)_{<v}$ by Lemma 3.5.1. We also have $M * (\sigma \circ \tau) = (\sigma \circ \tau)\langle v, d \rangle * (\rho \circ (\sigma \circ \tau))_{<v} \simeq (\sigma\langle v, d \rangle * \tau_{\geq v}) * (\rho \circ (\sigma \circ \tau))_{<v} = (v^e * \tau_{\geq v}) * (\rho \circ (\sigma \circ \tau))_{<v} \simeq \tau\langle v, e \rangle * (\rho \circ (\sigma \circ \tau))_{<v}$. Since $(\rho \circ \sigma) \circ \tau \simeq \rho \circ (\sigma \circ \tau)$, we have $(M * \sigma) * \tau \simeq M * (\sigma \circ \tau)$.
2. Consider the case that $\text{Lv}(\sigma) \geq \text{Lv}(v)$. Then the following holds:

$$\begin{aligned} (M * \sigma) * \tau &= (v^d[\rho] * \sigma) * \tau = (\sigma\langle v, d \rangle * (\rho \circ \sigma)_{<v}) * \tau \\ &\simeq \sigma\langle v, d \rangle * ((\rho \circ \sigma)_{<v} \circ \tau) \quad \text{by induction hypothesis,} \\ &\simeq \sigma\langle v, d \rangle * (\tau_{\geq v} \circ ((\rho \circ \sigma) \circ \tau)_{<v}) \quad \text{by Lemma 3.5.9,} \\ &\simeq (\sigma\langle v, d \rangle * \tau_{\geq v}) * ((\rho \circ \sigma) \circ \tau)_{<v} \quad \text{by induction hypothesis,} \\ &\simeq (\sigma\langle v, d \rangle * \tau_{\geq v}) * (\rho \circ (\sigma \circ \tau))_{<v} \quad \text{since } (\rho \circ \sigma) \circ \tau \simeq \rho \circ (\sigma \circ \tau). \end{aligned}$$

Also $M * (\sigma \circ \tau) \simeq (\sigma \circ \tau)\langle v, d \rangle * (\rho \circ (\sigma \circ \tau))_{<v} \simeq (\sigma\langle v, d \rangle * \tau_{\geq v}) * (\rho \circ (\sigma \circ \tau))_{<v}$.

Hence $(M * \sigma) * \tau \simeq M * (\sigma \circ \tau)$.

- B. Consider the case that $M = \lambda v.M_1$. First, we show that $(\sigma \circ \uparrow_v) \circ \uparrow_v(\tau) \simeq (\sigma \circ \tau) \circ \uparrow_v$. Let w be a name and d a nonnegative integer.

1. Consider the case that $\text{Lv}(\sigma) = 0$. Then $\sigma\langle w, d \rangle = w^e$ for a nonnegative integer e . Thus we have $((\sigma \circ \uparrow_v) \circ \uparrow_v(\tau))\langle w, d \rangle \simeq (\sigma \circ \uparrow_v)\langle w, d \rangle * (\uparrow_v(\tau))_{\geq w} \simeq (\sigma\langle w, d \rangle * (\uparrow_v)_{\geq w}) * (\uparrow_v(\tau))_{\geq w} = (w^e * (\uparrow_v)_{\geq w}) * (\uparrow_v(\tau))_{\geq w} = w^{e+\delta_{vw}} * (\uparrow_v(\tau))_{\geq w} \simeq \uparrow_v(\tau)\langle w, e + \delta_{vw} \rangle = (v\downarrow(v) \cdot (\tau \circ \uparrow_v))\langle w, e + \delta_{vw} \rangle = (\tau \circ \uparrow_v)\langle w, e \rangle \simeq \tau\langle w, e \rangle * (\uparrow_v)_{\geq w}$. We also have $((\sigma \circ \tau) \circ \uparrow_v)\langle w, d \rangle \simeq (\sigma \circ \tau)\langle w, d \rangle * (\uparrow_v)_{\geq w} \simeq (\sigma\langle w, d \rangle * \tau_{\geq w}) * (\uparrow_v)_{\geq w} \simeq (w^e * \tau_{\geq w}) * (\uparrow_v)_{\geq w} \simeq \tau\langle w, e \rangle * (\uparrow_v)_{\geq w}$. Accordingly $((\sigma \circ \uparrow_v) \circ \uparrow_v(\tau))\langle w, d \rangle \simeq ((\sigma \circ \tau) \circ \uparrow_v)\langle w, d \rangle$.
2. Consider the case that $\text{Lv}(\sigma) > 0$. Then $\text{Lv}((\uparrow_v)_{\geq w}) = 0 < \text{Lv}(\sigma)$. Thus the following holds:

$$\begin{aligned}
& ((\sigma \circ \uparrow_v) \circ \uparrow_v(\tau))\langle w, d \rangle \simeq (\sigma\langle w, d \rangle * (\uparrow_v)_{\geq w}) * (\uparrow_v(\tau))_{\geq w} \quad \text{by Lemma 3.5.6,} \\
& \simeq \sigma\langle w, d \rangle * ((\uparrow_v)_{\geq w} \circ (\uparrow_v(\tau))_{\geq w}) \quad \text{by induction hypothesis,} \\
& = \sigma\langle w, d \rangle * (\uparrow_v \circ \uparrow_v(\tau))_{\geq w} \quad \text{by Lemma 3.5.2,} \\
& = \sigma\langle w, d \rangle * (\uparrow_v \cdot v\downarrow(v) \cdot (\tau \circ \uparrow_v))_{\geq w} \simeq \sigma\langle w, d \rangle * (\tau \circ \uparrow_v)_{\geq w}.
\end{aligned}$$

Also, by induction hypothesis $((\sigma \circ \tau) \circ \uparrow_v)\langle w, d \rangle \simeq (\sigma\langle w, d \rangle * \tau_{\geq w}) * (\uparrow_v)_{\geq w} \simeq \sigma\langle w, d \rangle * (\tau_{\geq w} \circ (\uparrow_v)_{\geq w})$, which equals $\sigma\langle w, d \rangle * (\tau \circ \uparrow_v)_{\geq w}$ by Lemma 3.5.2. Thus $((\sigma \circ \uparrow_v) \circ \uparrow_v(\tau))\langle w, d \rangle \simeq ((\sigma \circ \tau) \circ \uparrow_v)\langle w, d \rangle$.

By the above, we have $(\sigma \circ \uparrow_v) \circ \uparrow_v(\tau) \simeq (\sigma \circ \tau) \circ \uparrow_v$. Accordingly $\uparrow_v(\sigma) \circ \uparrow_v(\tau) = v\downarrow(v * (\uparrow_v(\tau))_{\geq w}) \cdot ((\sigma \circ \uparrow_v) \circ \uparrow_v(\tau)) \simeq v\downarrow(v) \cdot ((\sigma \circ \tau) \circ \uparrow_v) = \uparrow_v(\sigma \circ \tau)$. Thus we have $M * (\sigma \circ \tau) = \lambda v.(M_1 * \uparrow_v(\sigma \circ \tau)) \simeq \lambda v.(M_1 * (\uparrow_v(\sigma) \circ \uparrow_v(\tau)))$, and by induction hypothesis the last term $\lambda v.(M_1 * (\uparrow_v(\sigma) \circ \uparrow_v(\tau))) \simeq \lambda v.((M_1 * \uparrow_v(\sigma)) * \uparrow_v(\tau)) = (M * \sigma) * \tau$.

- C. Consider the case that $M = M_1 @_{\ell} M_2$. Then by induction hypothesis $(M * \sigma) * \tau = ((M_1 * \sigma) * \tau) @_{\ell} ((M_2 * \sigma_{\geq \ell}) * \tau_{\geq \ell}) \simeq (M_1 * (\sigma \circ \tau)) @_{\ell} (M_2 * (\sigma_{\geq \ell} \circ \tau_{\geq \ell}))$, which equals $(M_1 * (\sigma \circ \tau)) @_{\ell} (M_2 * (\sigma \circ \tau)_{\geq \ell}) = M * (\sigma \circ \tau)$ by Lemma 3.5.2. \square

Corollary 3.5.11. Let ρ, σ and τ be substitutions. Then $(\rho \circ \sigma) \circ \tau \simeq \rho \circ (\sigma \circ \tau)$.

Proof. This proposition is shown in the proof of Proposition 3.5.10. \square

Summing up, we have the following basic algebraic properties for all terms M and M' , and for all substitutions $\rho, \sigma, \sigma', \tau$ and τ' :

- (1) If $M \simeq M'$ and $\sigma \simeq \sigma'$, then $M * \sigma \simeq M' * \sigma'$.
- (2) If $\sigma \simeq \sigma'$ and $\tau \simeq \tau'$, then $\sigma \circ \tau \simeq \sigma' \circ \tau'$.
- (3) $\text{id} \circ \sigma \simeq \sigma \simeq \sigma \circ \text{id}$.
- (4) $(\rho \circ \sigma) \circ \tau \simeq \rho \circ (\sigma \circ \tau)$.
- (5) $M * \text{id} \simeq M$.
- (6) $(M * \sigma) * \tau \simeq M * (\sigma \circ \tau)$.

3.6 Confluence

We prove that β -reduction is confluent up to equivalence by the technique of parallel reduction [19]. In other words, we prove confluence of $\beta\varepsilon$ -reduction.

Definition 3.6.1. We define binary relation \Rightarrow on the union $\text{Ter} \cup \text{Sub}$ of the set Ter of

terms and the set Sub of substitutions inductively as follows:

$$\begin{aligned}
 (\lambda v.M) @_\ell N &\Rightarrow M' * {}^v\downarrow(N') && \text{if } \ell = \text{Lv}(v), M \Rightarrow M', \text{ and } N \Rightarrow N', \\
 v^d[\sigma] &\Rightarrow v^d[\sigma'] && \text{if } \sigma \Rightarrow \sigma', \\
 \lambda v.M &\Rightarrow \lambda v.M' && \text{if } M \Rightarrow M', \\
 M_1 @_\ell M_2 &\Rightarrow M'_1 @_\ell M'_2 && \text{if } M_1 \Rightarrow M'_1 \text{ and } M_2 \Rightarrow M'_2, \\
 \text{id} &\Rightarrow \text{id}, \\
 {}^v\downarrow(M) \cdot \sigma &\Rightarrow {}^v\downarrow(M') \cdot \sigma' && \text{if } M \Rightarrow M' \text{ and } \sigma \Rightarrow \sigma', \\
 \uparrow_v \cdot \sigma &\Rightarrow \uparrow_v \cdot \sigma' && \text{if } \sigma \Rightarrow \sigma'.
 \end{aligned}$$

We can easily see that binary relation \Rightarrow is reflexive.

Remark 3.6.2. Let σ, σ', τ and τ' be substitutions. In the subsequent proofs, we use the following basic properties that are trivial by definition.

- (1) If $\sigma \Rightarrow \sigma'$, then $\sigma \upharpoonright S \Rightarrow \sigma' \upharpoonright S$ for any subset S of the set \mathcal{N} of names.
- (2) If $\sigma \Rightarrow \sigma'$, then $\sigma \langle v, d \rangle \Rightarrow \sigma' \langle v, d \rangle$ for all names v and nonnegative integers d .
- (3) Suppose that $\text{Lh}(\sigma) = \text{Lh}(\sigma')$. Then $\sigma \cdot \tau \Rightarrow \sigma' \cdot \tau'$ if and only if $\sigma \Rightarrow \sigma'$ and $\tau \Rightarrow \tau'$.

Lemma 3.6.3. We have $\rightarrow_\beta \subseteq \Rightarrow \subseteq \rightarrow_\beta^*$.

Proof. First, we show that $\rightarrow_\beta \subseteq \Rightarrow$. Let ℓ be a level, v a name, d a nonnegative integer, M, M', N terms, and σ, σ', τ substitutions. Then the following hold by reflexivity of \Rightarrow :

$$\begin{aligned}
 \sigma \cdot {}^v\downarrow(M) \cdot \tau &\Rightarrow \sigma \cdot {}^v\downarrow(M') \cdot \tau && \text{if } M \Rightarrow M', \\
 v^d[\sigma] &\Rightarrow v^d[\sigma'] && \text{if } \sigma \Rightarrow \sigma', \\
 \lambda v.M &\Rightarrow \lambda v.M' && \text{if } M \Rightarrow M', \\
 M @_\ell N &\Rightarrow M' @_\ell N && \text{if } M \Rightarrow M', \\
 N @_\ell M &\Rightarrow N @_\ell M' && \text{if } M \Rightarrow M'.
 \end{aligned}$$

Hence \Rightarrow is a reduction relation, which is defined in Definition 3.3.1. Furthermore, the following expression holds:

$$(\lambda v.M) @_\ell N \Rightarrow M * {}^v\downarrow(N) \quad \text{if } \text{Lv}(v) = \ell,$$

by reflexivity of \Rightarrow . Since \rightarrow_β is the least reduction relation satisfying the counterpart of the above expression as defined in Definition 3.3.2, we have $\rightarrow_\beta \subseteq \Rightarrow$.

Next, we show that $\Rightarrow \subseteq \rightarrow_\beta^*$. Suppose that $M \Rightarrow M'$ for terms M and M' . We prove that $M \rightarrow_\beta^* M'$ by induction on $\text{Ht}(M)$.

A. Consider the case that $M = v^d[\sigma]$. Since $M \Rightarrow M'$, we have $M' = v^d[\sigma']$ for a substitution σ' with $\sigma \Rightarrow \sigma'$. We show that $\sigma \rightarrow_\beta^* \sigma'$ by induction on $\text{Lh}(\sigma)$.

1. Consider the case that $\sigma = \text{id}$. Since $\sigma \Rightarrow \sigma'$, we have $\sigma' = \text{id}$ and thus $\sigma \rightarrow_\beta^* \sigma'$.
2. Consider the case that $\sigma = {}^w\downarrow(N) \cdot \sigma_1$. Since $\sigma \Rightarrow \sigma'$, we have $\sigma' = {}^w\downarrow(N') \cdot \sigma'_1$ for a term N' and a substitution σ'_1 with $N \Rightarrow N'$ and $\sigma_1 \Rightarrow \sigma'_1$. By induction hypothesis on $\text{Ht}(M)$, we have $N \rightarrow_\beta^* N'$. We also have $\sigma_1 \rightarrow_\beta^* \sigma'_1$ by induction hypothesis on $\text{Lh}(\sigma)$. Thus $\sigma = {}^w\downarrow(N) \cdot \sigma_1 \rightarrow_\beta^* {}^w\downarrow(N') \cdot \sigma'_1 = \sigma'$.
3. Consider the case that $\sigma = \uparrow_w \cdot \sigma_1$. Since $\sigma \Rightarrow \sigma'$, we have $\sigma' = \uparrow_w \cdot \sigma'_1$ for a substitution σ'_1 with $\sigma_1 \Rightarrow \sigma'_1$. By induction hypothesis on $\text{Lh}(\sigma)$, we have $\sigma_1 \rightarrow_\beta^* \sigma'_1$. Thus $\sigma = \uparrow_w \cdot \sigma_1 \rightarrow_\beta^* \uparrow_w \cdot \sigma'_1 = \sigma'$.

By the above, we have $\sigma \rightarrow_{\beta}^* \sigma'$ and thus $M = v^d[\sigma] \rightarrow_{\beta}^* v^d[\sigma'] = M'$.

- B. Consider the case that $M = \lambda v.M_1$. Since $M \Rightarrow M'$, we have $M' = \lambda v.M'_1$ for a term M'_1 with $M_1 \Rightarrow M'_1$. By induction hypothesis, we have $M_1 \rightarrow_{\beta}^* M'_1$. Thus $M = \lambda v.M_1 \rightarrow_{\beta}^* \lambda v.M'_1 = M'$.
 - C. Consider the case that $M = M_1 @_{\ell} M_2$ and $M' = M'_1 @_{\ell} M'_2$ with $M_i \Rightarrow M'_i$ for $i = 1, 2$. By induction hypothesis, we have $M_i \rightarrow_{\beta}^* M'_i$ for $i = 1, 2$. Thus $M = M_1 @_{\ell} M_2 \rightarrow_{\beta}^* M'_1 @_{\ell} M'_2 = M'$.
 - D. Consider the case that $M = (\lambda v.M_1) @_{\ell} M_2$ and $M' = M'_1 * v\downarrow(M'_2)$ with $\text{Lv}(v) = \ell$ and $M_i \Rightarrow M'_i$ for $i = 1, 2$. By induction hypothesis, we have $M_i \rightarrow_{\beta}^* M'_i$ for $i = 1, 2$. Thus $M = (\lambda v.M_1) @_{\ell} M_2 \rightarrow_{\beta}^* (\lambda v.M'_1) @_{\ell} M'_2 \rightarrow_{\beta}^* M'_1 * v\downarrow(M'_2) = M'$.
- By the above, we have $M \rightarrow_{\beta}^* M'$. We also see that $\sigma \Rightarrow \sigma'$ implies $\sigma \rightarrow_{\beta}^* \sigma'$ for all substitutions σ and σ' by the discussion for case A. Therefore $\Rightarrow \subseteq \rightarrow_{\beta}^*$. \square

Corollary 3.6.4. We have $\Rightarrow^* = \rightarrow_{\beta}^*$.

Proof. By Lemma 3.6.3, we have $\rightarrow_{\beta}^* \subseteq \Rightarrow^* \subseteq (\rightarrow_{\beta}^*)^* = \rightarrow_{\beta}^*$. \square

Lemma 3.6.5. Let M and M' be terms, σ and σ' substitutions. If $M \Rightarrow \cdot \simeq M'$ and $\sigma \Rightarrow \cdot \simeq \sigma'$, then $M * \sigma \Rightarrow \cdot \simeq M' * \sigma'$.

Proof. We use induction on lexicographic ordering of pairs $\langle \text{Lv}(\sigma), \text{Ht}(M) \rangle$. Suppose that $M \Rightarrow M'' \simeq M'$ and $\sigma \Rightarrow \sigma'' \simeq \sigma'$ for a term M'' and a substitution σ'' .

- A. Consider the case that $M = v^d[\tau]$. Since $M \Rightarrow M'' \simeq M'$, we have $M'' = v^d[\tau'']$ and $M' = v^d[\tau']$ for some substitutions τ'' and τ' with $\tau \Rightarrow \tau'' \simeq \tau'$. First, we show that $\tau \circ \sigma \Rightarrow \cdot \simeq \tau' \circ \sigma'$ by induction on $\text{Lh}(\tau)$.
 1. Consider the case that $\tau = \text{id}$. Since $\tau \Rightarrow \tau''$, we have $\tau'' = \text{id}$. Thus $\tau \circ \sigma = \sigma \Rightarrow \sigma'' = \tau'' \circ \sigma'' \simeq \tau' \circ \sigma'$.
 2. Consider the case that $\tau = w\downarrow(N) \cdot \tau_1$. Since $\tau \Rightarrow \tau''$, we have $\tau'' = w\downarrow(N'') \cdot \tau_1''$ for some term N'' and some substitution τ_1'' with $N \Rightarrow N''$ and $\tau_1 \Rightarrow \tau_1''$. By induction hypothesis on $\langle \text{Lv}(\sigma), \text{Ht}(M) \rangle$, we have $N * \sigma_{\geq w} \Rightarrow \cdot \simeq N'' * \sigma''_{\geq w}$. By induction hypothesis on $\text{Lh}(\tau)$, we also have $\tau_1 \circ \sigma \Rightarrow \cdot \simeq \tau_1'' \circ \sigma''$. Thus $\tau \circ \sigma = w\downarrow(N * \sigma_{\geq w}) \cdot (\tau_1 \circ \sigma) \Rightarrow \cdot \simeq w\downarrow(N'' * \sigma''_{\geq w}) \cdot (\tau_1'' \circ \sigma'') = \tau'' \circ \sigma'' \simeq \tau' \circ \sigma'$.
 3. Consider the case that $\tau = \uparrow_w \cdot \tau_1$. Since $\tau \Rightarrow \tau''$, we have $\tau'' = \uparrow_w \cdot \tau_1''$ for some substitution τ_1'' with $\tau_1 \Rightarrow \tau_1''$. By induction hypothesis on $\text{Lh}(\tau)$, we have $\tau_1 \circ \sigma \Rightarrow \cdot \simeq \tau_1'' \circ \sigma''$. Therefore $\tau \circ \sigma = \uparrow_w \cdot (\tau_1 \circ \sigma) \Rightarrow \cdot \simeq \uparrow_w \cdot (\tau_1'' \circ \sigma'') = \tau'' \circ \sigma'' \simeq \tau' \circ \sigma'$.

Next, we show that $M * \sigma \Rightarrow \cdot \simeq M' * \sigma'$.

1. Consider the case that $\text{Lv}(\sigma) < \text{Lv}(v)$. Then $\sigma \langle v, d \rangle = v^e$ for a nonnegative integer e . Since $\sigma \langle v, d \rangle \Rightarrow \sigma'' \langle v, d \rangle$, we have $\sigma'' \langle v, d \rangle = v^e$ and thus $v^e \simeq \sigma' \langle v, d \rangle$. Hence $M * \sigma = \sigma \langle v, d \rangle * (\tau \circ \sigma)_{<v} = v^e * (\tau \circ \sigma)_{<v} = v^e[(\tau \circ \sigma)_{<v}] \Rightarrow \cdot \simeq v^e[(\tau' \circ \sigma')_{<v}] = v^e * (\tau' \circ \sigma')_{<v} \simeq \sigma' \langle v, d \rangle * (\tau' \circ \sigma')_{<v} = M' * \sigma'$.
 2. Consider the case that $\text{Lv}(\sigma) \geq \text{Lv}(v)$. Then we have $\sigma \langle v, d \rangle * (\tau \circ \sigma)_{<v} \Rightarrow \cdot \simeq \sigma' \langle v, d \rangle * (\tau' \circ \sigma')_{<v}$ by induction hypothesis. Accordingly, we have $M * \sigma = \sigma \langle v, d \rangle * (\tau \circ \sigma)_{<v} \Rightarrow \cdot \simeq \sigma' \langle v, d \rangle * (\tau' \circ \sigma')_{<v} = M' * \sigma'$.
- B. Consider the case that $M = \lambda v.M_1$. Since $M \Rightarrow M'' \simeq M'$, we have $M'' = \lambda v.M'_1$ and $M' = \lambda v.M'_1$ for some terms M''_1 and M'_1 with $M_1 \Rightarrow M''_1 \simeq M'_1$. First, we show that $\sigma \circ \uparrow_v \Rightarrow \cdot \simeq \sigma' \circ \uparrow_v$ by induction on $\text{Lh}(\sigma)$.
 1. Consider the case that $\sigma = \text{id}$. Since $\sigma \Rightarrow \sigma''$, we have $\sigma'' = \text{id}$. Thus $\sigma \circ \uparrow_v = \sigma'' \circ \uparrow_v \simeq \sigma' \circ \uparrow_v$.
 2. Consider the case that $\sigma = w\downarrow(N) \cdot \sigma_1$. Since $\sigma \Rightarrow \sigma''$, we have $\sigma'' = w\downarrow(N'') \cdot \sigma_1''$.

for a term N'' and a substitution σ_1'' with $N \Rightarrow N''$ and $\sigma_1 \Rightarrow \sigma_1''$. By induction hypothesis on $\text{Lh}(\sigma)$, we have $\sigma_1 \circ \uparrow_v \Rightarrow \cdot \simeq \sigma_1'' \circ \uparrow_v$. There are the following two possible cases.

- a. Consider the case that $N = w^d$ for a nonnegative integer d . Since $N \Rightarrow N''$, we have $N'' = w^d = N$. Hence $\sigma \circ \uparrow_v = w\downarrow(N * (\uparrow_v)_{\geq w}) \cdot (\sigma_1 \circ \uparrow_v) \Rightarrow \cdot \simeq w\downarrow(N'' * (\uparrow_v)_{\geq w}) \cdot (\sigma_1'' \circ \uparrow_v) = (w\downarrow(N'') \cdot \sigma_1'') \circ \uparrow_v = \sigma'' \circ \uparrow_v \simeq \sigma' \circ \uparrow_v$.
- b. Consider the case that $N \neq w^d$ for any nonnegative integer d . Then $\text{Lv}(\sigma) \geq \text{Lv}(w) > 0$. Since $\text{Lv}((\uparrow_v)_{\geq w}) = 0$, we have $N * (\uparrow_v)_{\geq w} \Rightarrow \cdot \simeq N'' * (\uparrow_v)_{\geq w}$ by induction hypothesis on $\langle \text{Lv}(\sigma), \text{Ht}(M) \rangle$. Therefore, we have $\sigma \circ \uparrow_v = w\downarrow(N * (\uparrow_v)_{\geq w}) \cdot (\sigma_1 \circ \uparrow_v) \Rightarrow \cdot \simeq w\downarrow(N'' * (\uparrow_v)_{\geq w}) \cdot (\sigma_1'' \circ \uparrow_v) = \sigma'' \circ \uparrow_v \simeq \sigma' \circ \uparrow_v$.
3. Consider the case that $\sigma = \uparrow_w \cdot \sigma_1$. Since $\sigma \Rightarrow \sigma''$, we have $\sigma'' = \uparrow_w \cdot \sigma_1''$ for a substitution σ_1'' with $\sigma_1 \Rightarrow \sigma_1''$. By induction hypothesis on $\text{Lh}(\sigma)$, we have $\sigma_1 \circ \uparrow_v \Rightarrow \cdot \simeq \sigma_1'' \circ \uparrow_v$. Thus $\sigma \circ \uparrow_v = \uparrow_w \cdot (\sigma_1 \circ \uparrow_v) \Rightarrow \cdot \simeq \uparrow_w \cdot (\sigma_1'' \circ \uparrow_v) = \sigma'' \circ \uparrow_v \simeq \sigma' \circ \uparrow_v$. By the above, we have $\sigma \circ \uparrow_v \Rightarrow \cdot \simeq \sigma' \circ \uparrow_v$ and thus $\uparrow_v(\sigma) = v\downarrow(v) \cdot (\sigma \circ \uparrow_v) \Rightarrow \cdot \simeq v\downarrow(v) \cdot (\sigma' \circ \uparrow_v) = \uparrow_v(\sigma')$. Consequently $M_1 * \uparrow_v(\sigma) \Rightarrow \cdot \simeq M_1' * \uparrow_v(\sigma')$ by induction hypothesis. Hence $M * \sigma = \lambda v. (M_1 * \uparrow_v(\sigma)) \Rightarrow \cdot \simeq \lambda v. (M_1' * \uparrow_v(\sigma')) = M' * \sigma'$.
- C. Consider the case that $M = M_1 @_\ell M_2$, $M'' = M_1'' @_\ell M_2''$ and $M' = M_1' @_\ell M_2'$ with $M_i \Rightarrow M_i'' \simeq M_i'$ for $i = 1, 2$. By induction hypothesis, we have $M_1 * \sigma \Rightarrow \cdot \simeq M_1' * \sigma'$ and $M_2 * \sigma_{\geq \ell} \Rightarrow \cdot \simeq M_2' * \sigma'_{\geq \ell}$. Therefore $M * \sigma = (M_1 * \sigma) @_\ell (M_2 * \sigma_{\geq \ell}) \Rightarrow \cdot \simeq (M_1' * \sigma') @_\ell (M_2' * \sigma'_{\geq \ell}) = M' * \sigma'$.
- D. Consider the case that $M = (\lambda v. M_1) @_\ell M_2$ and $M'' = M_1'' * v\downarrow(M_2'')$ with $\ell = \text{Lv}(v)$ and $M_i \Rightarrow M_i''$ for $i = 1, 2$. As shown in the discussion for case B, we have $\uparrow_v(\sigma) \Rightarrow \cdot \simeq \uparrow_v(\sigma') \simeq \uparrow_v(\sigma'')$. By induction hypothesis, we have $M_1 * \uparrow_v(\sigma) \Rightarrow L_1 \simeq M_1'' * \uparrow_v(\sigma'')$ for a term L_1 , and $M_2 * \sigma_{\geq v} \Rightarrow L_2 \simeq M_2'' * \sigma''_{\geq v}$ for a term L_2 . Hence the following holds:

$$\begin{aligned}
M * \sigma &= (\lambda v. (M_1 * \uparrow_v(\sigma))) @_\ell (M_2 * \sigma_{\geq v}) \\
&\Rightarrow L_1 * v\downarrow(L_2) \simeq (M_1'' * \uparrow_v(\sigma'')) * v\downarrow(M_2'' * \sigma''_{\geq v}) \\
&\simeq M_1'' * ((\uparrow_v(\sigma'') \circ v\downarrow(M_2'' * \sigma''_{\geq v})) = M_1'' * ((v\downarrow(v) \cdot (\sigma'' \circ \uparrow_v)) \circ v\downarrow(M_2'' * \sigma''_{\geq v})) \\
&= M_1'' * (v\downarrow((M_2'' * \sigma''_{\geq v}) * \text{id}) \cdot ((\sigma'' \circ \uparrow_v) \circ v\downarrow(M_2'' * \sigma''_{\geq v}))) \\
&\simeq M_1'' * (v\downarrow(M_2'' * \sigma''_{\geq v}) \cdot (\sigma'' \circ (\uparrow_v \circ v\downarrow(M_2'' * \sigma''_{\geq v})))) \simeq M_1'' * (v\downarrow(M_2'' * \sigma''_{\geq v}) \cdot \sigma'') \\
&= M_1'' * (v\downarrow(M_2'') \circ \sigma'') \simeq (M_1'' * v\downarrow(M_2'')) * \sigma'' = M'' * \sigma'' \simeq M' * \sigma'. \quad \square
\end{aligned}$$

Proposition 3.6.6. We have $(\simeq \cdot \Rightarrow) \subseteq (\Rightarrow \cdot \simeq)$.

Proof. Let M , M' and N be terms. By induction on $\text{Ht}(M')$, we prove that if $M \simeq M' \Rightarrow N$ then $M \Rightarrow \cdot \simeq N$.

- A. Consider the case that $M = v^d[\sigma]$. By $M \simeq M' \Rightarrow N$, we have $M' = v^d[\sigma']$ and $N = v^d[\tau]$ for some substitutions σ' and τ with $\sigma \simeq \sigma' \Rightarrow \tau$. We show that $\sigma \Rightarrow \cdot \simeq \tau$.
 1. Consider the case that $\text{Ht}(M') = 0$. Then σ' consists only of pop-elements. Since $\sigma' \Rightarrow \tau$, we have $\tau = \sigma'$. Thus $\sigma \Rightarrow \cdot \simeq \sigma' = \tau$.
 2. Consider the case that $\text{Ht}(M') > 0$. Let $\alpha_0, \alpha_1, \dots, \alpha_{n-1}$ be the push-pop-elements satisfying $\sigma = \alpha_0 \cdot \alpha_1 \cdot \dots \cdot \alpha_{n-1}$. We define τ' as $\beta_0 \cdot \beta_1 \cdot \dots \cdot \beta_{n-1}$, where β_i is the push-pop-element defined as follows for each $i = 0, 1, \dots, n-1$:
 - a. Consider the case that σ contains the $\langle w, e \rangle$ -component in the i -th push-pop-element, or more specifically, the case satisfying the following two conditions for some name w and nonnegative integer e : (1) $(\alpha_0 \cdot \alpha_1 \cdot \dots \cdot \alpha_{i-1} \cdot \rho) \langle w, e \rangle = \rho \langle w, 0 \rangle$ for any substitution ρ , and (2) $\alpha_i = w\downarrow(\sigma \langle w, e \rangle)$. Since $\sigma \langle w, e \rangle \simeq$

$\sigma'\langle w, e \rangle \Rightarrow \tau\langle w, e \rangle$ and $\text{Ht}(\sigma'\langle w, e \rangle) < \text{Ht}(M')$, there exists a term L such that $\sigma\langle w, e \rangle \Rightarrow L \simeq \tau\langle w, e \rangle$ by induction hypothesis. We define β_i as $w\downarrow(L)$.

b. If it is not the above case, then we define β_i as α_i .

Since $\alpha_i \Rightarrow \beta_i$ for each $i = 0, 1, \dots, n-1$, we have $\sigma \Rightarrow \tau'$. Lastly, we show that $\tau' \simeq \tau$. Let w be a name and e a nonnegative integer.

- a. Consider the case that $\sigma\langle w, e \rangle \neq w^c$ for any nonnegative integer c . Then there exists a nonnegative integer i such that σ contains the $\langle w, e \rangle$ -component in the i -th push-pop-element. Hence $\beta_i = w\downarrow(L)$ for a term L satisfying $L \simeq \tau\langle w, e \rangle$. By definition, we have the following fact for each $j = 0, 1, \dots, n-1$: if α_j is a push-element (resp. pop-element) indexed by w , then β_j is also a push-element (resp. pop-element) indexed by w . By this fact, we see that τ' also has the $\langle w, e \rangle$ -component in the i -th push-pop-element. Thus $\tau'\langle w, e \rangle = L \simeq \tau\langle w, e \rangle$.
- b. Consider the case that $\sigma\langle w, e \rangle = w^c$. Since $\sigma \simeq \sigma' \Rightarrow \tau$, we have $\sigma'\langle w, e \rangle = w^c[\sigma'_1]$ and $\tau\langle w, e \rangle = w^c[\tau_1]$ for some substitutions σ'_1 and τ_1 with $\text{id} \simeq \sigma'_1 \Rightarrow \tau_1$. We show that $\text{id} \simeq \tau_1$. Let u be a name and b a nonnegative integer. Then $u^b = \text{id}\langle u, b \rangle \simeq \sigma'_1\langle u, b \rangle \Rightarrow \tau_1\langle u, b \rangle$. Since $\text{Ht}(\sigma'_1\langle u, b \rangle) \leq \text{Ht}(\sigma'\langle w, e \rangle) < \text{Ht}(M')$, we have $u^b \Rightarrow \cdot \simeq \tau_1\langle u, b \rangle$ by induction hypothesis, and thus $u^b \simeq \tau_1\langle u, b \rangle$. Since u and b are arbitrary, it follows that $\text{id} \simeq \tau_1$. Since $w^c = \sigma\langle w, e \rangle \Rightarrow \tau'\langle w, e \rangle$, we also have $\tau'\langle w, e \rangle = w^c$. Thus $\tau'\langle w, e \rangle = w^c \simeq w^c[\tau_1] = \tau\langle w, e \rangle$.

By the above, we have $\tau' \simeq \tau$. Therefore $M = v^d[\sigma] \Rightarrow v^d[\tau'] \simeq v^d[\tau] = N$.

- B. Consider the case that $M = \lambda v.M_1$. Since $M \simeq M' \Rightarrow N$, we have $M' = \lambda v.M'_1$ and $N = \lambda v.N_1$ for some terms M'_1 and N_1 with $M_1 \simeq M'_1 \Rightarrow N_1$. By induction hypothesis, we have $M_1 \Rightarrow \cdot \simeq N_1$. Thus $M = \lambda v.M_1 \Rightarrow \cdot \simeq \lambda v.N_1 = N$.
- C. Consider the case that $M = M_1 @_\ell M_2$, $M' = M'_1 @_\ell M'_2$ and $N = N_1 @_\ell N_2$ with $M_i \simeq M'_i \Rightarrow N_i$ for $i = 1, 2$. By induction hypothesis, we have $M_i \Rightarrow \cdot \simeq N_i$ for $i = 1, 2$. Thus $M = M_1 @_\ell M_2 \Rightarrow \cdot \simeq N_1 @_\ell N_2 = N$.
- D. Consider the case that $M = (\lambda v.M_1) @_\ell M_2$, $M' = (\lambda v.M'_1) @_\ell M'_2$ and $N = N_1 * v\downarrow(N_2)$ with $\ell = \text{Lv}(v)$ and $M_i \simeq M'_i \Rightarrow N_i$ for $i = 1, 2$. By induction hypothesis, there exist terms N'_1 and N'_2 such that $M_i \Rightarrow N'_i \simeq N_i$ for $i = 1, 2$. Thus $M = (\lambda v.M_1) @_\ell M_2 \Rightarrow N'_1 * v\downarrow(N'_2) \simeq N_1 * v\downarrow(N_2) = N$.

We have seen that if $M \simeq \cdot \Rightarrow N$ for terms M and N , then $M \Rightarrow \cdot \simeq N$. By the discussion in case A, we have also proved that if $\sigma \simeq \cdot \Rightarrow \tau$ for substitutions σ and τ , then $\sigma \Rightarrow \cdot \simeq \tau$. Therefore $(\simeq \cdot \Rightarrow) \subseteq (\Rightarrow \cdot \simeq)$. \square

Definition 3.6.7. For each term M and each substitution σ , we define term M^\wedge and substitution σ^\wedge inductively as follows:

$$\begin{aligned}
(v^d[\sigma])^\wedge &:= v^d[\sigma^\wedge], \\
(\lambda v.M)^\wedge &:= \lambda v.M^\wedge, \\
(M @_\ell N)^\wedge &:= \begin{cases} L^\wedge * v\downarrow(N^\wedge) & \text{if } M = \lambda v.L \text{ and } \text{Lv}(v) = \ell, \\ M^\wedge @_\ell N^\wedge & \text{otherwise,} \end{cases} \\
\text{id}^\wedge &:= \text{id}, \\
(v\downarrow(M) \cdot \sigma)^\wedge &:= v\downarrow(M^\wedge) \cdot \sigma^\wedge, \\
(\uparrow_v \cdot \sigma)^\wedge &:= \uparrow_v \cdot \sigma^\wedge.
\end{aligned}$$

Lemma 3.6.8. Let M and N be terms. If $M \Rightarrow N$, then $N \Rightarrow \cdot \simeq M^\wedge$.

Proof. We use induction on $\text{Ht}(M)$. Suppose that $M \Rightarrow N$.

- A. Consider the case that $M = v^d[\sigma]$. Since $M \Rightarrow N$, we have $N = v^d[\tau]$ for a substitution

τ with $\sigma \Rightarrow \tau$. We show that $\tau \Rightarrow \cdot \simeq \sigma^\wedge$ by induction on $\text{Lh}(\sigma)$.

1. Consider the case that $\sigma = \text{id}$. Since $\sigma \Rightarrow \tau$, we have $\tau = \text{id}$ and thus $\tau \Rightarrow \cdot \simeq \sigma^\wedge$.
 2. Consider the case that $\sigma = w\downarrow(M_1) \cdot \sigma_1$. Since $\sigma \Rightarrow \tau$, we have $\tau = w\downarrow(N_1) \cdot \tau_1$ for some term N_1 and substitution τ_1 with $M_1 \Rightarrow N_1$ and $\sigma_1 \Rightarrow \tau_1$. By induction hypotheses on $\text{Ht}(M)$ and $\text{Lh}(\sigma)$, we have $N_1 \Rightarrow \cdot \simeq M_1^\wedge$ and $\tau_1 \Rightarrow \cdot \simeq \sigma_1^\wedge$. Hence $\tau = w\downarrow(N_1) \cdot \tau_1 \Rightarrow \cdot \simeq w\downarrow(M_1^\wedge) \cdot \sigma_1^\wedge = \sigma^\wedge$.
 3. Consider the case that $\sigma = \uparrow_w \cdot \sigma_1$. Since $\sigma \Rightarrow \tau$, we have $\tau = \uparrow_w \cdot \tau_1$ for some substitution τ_1 with $\sigma_1 \Rightarrow \tau_1$. By induction hypothesis on $\text{Lh}(\sigma)$, we have $\tau_1 \Rightarrow \cdot \simeq \sigma_1^\wedge$. Hence $\tau = \uparrow_w \cdot \tau_1 \Rightarrow \cdot \simeq \uparrow_w \cdot \sigma_1^\wedge = \sigma^\wedge$.
- By the above, we have $\tau \Rightarrow \cdot \simeq \sigma^\wedge$. Thus $N = v^d[\tau] \Rightarrow \cdot \simeq v^d[\sigma^\wedge] = M^\wedge$.
- B. Consider the case that $M = \lambda v.M_1$. Since $M \Rightarrow N$, we have $N = \lambda v.N_1$ for some term N_1 with $M_1 \Rightarrow N_1$. By induction hypothesis, we have $N_1 \Rightarrow \cdot \simeq M_1^\wedge$. Thus $N = \lambda v.N_1 \Rightarrow \cdot \simeq \lambda v.(M_1^\wedge) = M^\wedge$.
 - C. Consider the case that $M = M_1 @_\ell M_2$ and $N = N_1 @_\ell N_2$ with $M_i \Rightarrow N_i$ for $i = 1, 2$.
 1. Suppose that $M_1 = \lambda v.K_1$ for a name v and a term K_1 with $\text{Lv}(v) = \ell$. Then $N_1 = \lambda v.L_1$ for a term L_1 with $K_1 \Rightarrow L_1$. By induction hypothesis, we have $L_1 \Rightarrow L'_1 \simeq K_1^\wedge$ and $N_2 \Rightarrow N'_2 \simeq M_2^\wedge$ for some terms L'_1 and N'_2 . Thus $N \Rightarrow L'_1 * v\downarrow(N'_2) \simeq K_1^\wedge * v\downarrow(M_2^\wedge) = M^\wedge$.
 2. If it is not the above case, then $N_i \Rightarrow \cdot \simeq M_i^\wedge$ for $i = 1, 2$ by induction hypothesis, and hence $N = N_1 @_\ell N_2 \Rightarrow \cdot \simeq M_1^\wedge @_\ell M_2^\wedge = M^\wedge$.
 - D. Consider the case that $M = (\lambda v.M_1) @_\ell M_2$ and $N = N_1 * v\downarrow(N_2)$ with $\ell = \text{Lv}(v)$ and $M_i \Rightarrow N_i$ for $i = 1, 2$. By induction hypothesis, we have $N_i \Rightarrow \cdot \simeq M_i^\wedge$ for $i = 1, 2$. By Lemma 3.6.5, we have $N = N_1 * v\downarrow(N_2) \Rightarrow \cdot \simeq M_1^\wedge * v\downarrow(M_2^\wedge) = M^\wedge$. \square

Corollary 3.6.9. Let σ and τ be substitutions. If $\sigma \Rightarrow \tau$, then $\tau \Rightarrow \cdot \simeq \sigma^\wedge$.

Proof. This proposition is shown in the proof of Lemma 3.6.8. \square

Lemma 3.6.10. We have $(\Leftarrow \cdot \simeq \cdot \Rightarrow) \subseteq (\Rightarrow \cdot \simeq \cdot \Leftarrow)$.

Proof. Suppose that $M \Leftarrow M_1 \simeq N_1 \Rightarrow N$ for terms M, M_1, N and N_1 . By Proposition 3.6.6, there exists a term L such that $M_1 \Rightarrow L \simeq N$. By Lemma 3.6.8, there exist terms M' and L' such that $M \Rightarrow M' \simeq M_1^\wedge$ and $L \Rightarrow L' \simeq M_1^\wedge$. By Proposition 3.6.6, there exists a term N' such that $N \Rightarrow N' \simeq L'$. Consequently $M \Rightarrow M' \simeq M_1^\wedge \simeq L' \simeq N' \Leftarrow N$. In the same way, $\sigma \Leftarrow \cdot \simeq \cdot \Rightarrow \tau$ implies $\sigma \Rightarrow \cdot \simeq \cdot \Leftarrow \tau$ for all substitutions σ and τ . Therefore $(\Leftarrow \cdot \simeq \cdot \Rightarrow) \subseteq (\Rightarrow \cdot \simeq \cdot \Leftarrow)$. \square

Theorem 3.6.11. β -reduction \rightarrow_β is confluent up to equivalence. In other words, we have $(\leftarrow_\beta^* \cdot \simeq \cdot \rightarrow_\beta^*) \subseteq (\rightarrow_\beta^* \cdot \simeq \cdot \leftarrow_\beta^*)$.

Proof. Let M and N be terms. By Lemma 3.6.10, we see that $M (\Leftarrow \cdot \simeq)^* \cdot (\simeq \cdot \Rightarrow)^* N$ implies $M (\Rightarrow \cdot \simeq)^* \cdot (\simeq \cdot \Leftarrow)^* N$. In the same manner, we see that $\sigma (\Leftarrow \cdot \simeq)^* \cdot (\simeq \cdot \Rightarrow)^* \tau$ implies $\sigma (\Rightarrow \cdot \simeq)^* \cdot (\simeq \cdot \Leftarrow)^* \tau$ for all substitutions σ and τ . Hence $(\Leftarrow \cdot \simeq)^* \cdot (\simeq \cdot \Rightarrow)^* \subseteq (\Rightarrow \cdot \simeq)^* \cdot (\simeq \cdot \Leftarrow)^*$. Consequently, the following holds:

$$\begin{aligned} (\Leftarrow^* \cdot \simeq \cdot \Rightarrow^*) &\subseteq (\Leftarrow \cdot \simeq)^* \cdot (\simeq \cdot \Rightarrow)^* \subseteq (\Rightarrow \cdot \simeq)^* \cdot (\simeq \cdot \Leftarrow)^* && \text{as stated above,} \\ &\subseteq (\Rightarrow^* \cdot \simeq) \cdot (\simeq \cdot \Leftarrow^*) && \text{since } (\Rightarrow \cdot \simeq)^* \subseteq (\Rightarrow^* \cdot \simeq) \text{ by Lemma 3.6.6,} \\ &= (\Rightarrow^* \cdot \simeq \cdot \Leftarrow^*). \end{aligned}$$

Since \Rightarrow^* equals \rightarrow_β^* by Corollary 3.6.4, we have $(\leftarrow_\beta^* \cdot \simeq \cdot \rightarrow_\beta^*) \subseteq (\rightarrow_\beta^* \cdot \simeq \cdot \leftarrow_\beta^*)$. \square

Corollary 3.6.12. Let $\rightarrow_{\beta\varepsilon}$ be the union of β -reduction \rightarrow_β and ε -reduction \rightarrow_ε . Then $\rightarrow_{\beta\varepsilon}$ is confluent. In other words, we have $(\leftarrow_{\beta\varepsilon}^* \cdot \rightarrow_{\beta\varepsilon}^*) \subseteq (\rightarrow_{\beta\varepsilon}^* \cdot \leftarrow_{\beta\varepsilon}^*)$.

Proof. The following holds:

$$\begin{aligned}
 (\leftarrow_{\beta\epsilon}^* \cdot \rightarrow_{\beta\epsilon}^*) &\subseteq (\Leftarrow \cdot \simeq)^* \cdot (\simeq \cdot \Rightarrow)^* && \text{since } \rightarrow_{\beta\epsilon} \subseteq (\simeq \cdot \Rightarrow), \\
 &\subseteq (\Rightarrow^* \cdot \simeq \cdot \Leftarrow^*) && \text{as shown in the proof of Theorem 3.6.11,} \\
 &= (\rightarrow_{\beta}^* \cdot \simeq \cdot \leftarrow_{\beta}^*) && \text{since } \Rightarrow^* \text{ equals } \rightarrow_{\beta}^* \text{ by Corollary 3.6.4,} \\
 &= (\rightarrow_{\beta}^* \cdot \rightarrow_{\epsilon}^* \cdot \leftarrow_{\epsilon}^* \cdot \leftarrow_{\beta}^*) && \text{since } \simeq \text{ equals } (\rightarrow_{\epsilon}^* \cdot \leftarrow_{\epsilon}^*) \text{ by Proposition 3.4.11,} \\
 &\subseteq (\rightarrow_{\beta\epsilon}^* \cdot \leftarrow_{\beta\epsilon}^*).
 \end{aligned}$$

□

Chapter 4

An Application

Calculus λ^* provides us a way to manipulate binding structure flexibly with dynamic binding via meta-level variables. We illustrate the feature through an application of the calculus to a procedural language.

4.1 Procedural Language PROC

We introduce a simple procedural language PROC as an extension of imperative language IMP in [20]. PROC permits us to store and retrieve not only numbers but also procedures. We implement PROC in λ^* by exploiting the feature to manipulate binding structure dynamically. The implementation of PROC demonstrates that some notions related to names in procedural languages, such as stores, recursion, and localization of names, can be realized by dynamic binding via meta-level variables.

Definition 4.1.1. We define the syntax of PROC by the following BNF:

$$\begin{aligned}
 \text{N-expressions } E &::= x \mid n \mid \text{plus } E \ E \mid \text{minus } E \ E \\
 \text{P-expressions } F &::= z \mid \text{proc } P \\
 \text{Commands } C &::= x = E \mid z = F \mid \text{exec } F \mid \text{if } E \ P \ P \mid \text{while } E \ P \mid \\
 &\quad \text{local } P \ \text{export } \begin{smallmatrix} x_1 x_2 \dots x_h \\ z_1 z_2 \dots z_k \end{smallmatrix} \\
 \text{Procedures } P &::= \text{id} \mid C ; P
 \end{aligned}$$

where n, h, k range over the set \mathbb{N} of nonnegative integers, x, x_1, x_2, \dots, x_h range over the set \mathcal{N}_N of names for numbers, z, z_1, z_2, \dots, z_k range over the set \mathcal{N}_P of names for procedures, and id signifies the empty sequence. The sets \mathcal{N}_N and \mathcal{N}_P are disjoint. We define \mathbb{P} as the set of all procedures. Note that a procedure P is a finite sequence of commands. We represent concatenation of procedures P_1 and P_2 as $P_1 ; P_2$.

Definition 4.1.2. A pair $\langle \varphi_N, \varphi_P \rangle$ of a function φ_N of \mathcal{N}_N into \mathbb{N} and a function φ_P of \mathcal{N}_P into \mathbb{P} , is called a **state** of stores. We define \mathbb{S} as the set of all states of stores, and call an element of the set \mathbb{S} simply a state. Let $\varphi = \langle \varphi_N, \varphi_P \rangle$ be a state. Then the value E^φ of N-expression E in state φ is the nonnegative integer defined inductively as follows:

$$\begin{aligned}
 x^\varphi &:= \varphi_N(x), & n^\varphi &:= n, \\
 (\text{plus } E_1 \ E_2)^\varphi &:= E_1^\varphi + E_2^\varphi, & (\text{minus } E_1 \ E_2)^\varphi &:= \max\{E_1^\varphi - E_2^\varphi, 0\}.
 \end{aligned}$$

Similarly, the value F^φ of P-expression F in state φ is the procedure defined as follows:

$$z^\varphi := \varphi_P(z), \quad (\text{proc } P)^\varphi := P.$$

Definition 4.1.3. Let $\varphi = \langle \varphi_N, \varphi_P \rangle$ be a state. For nonnegative integers n and names x in \mathcal{N}_N , state $\varphi\{n/x\}$ is defined as follows:

$$\varphi\{n/x\} := \langle \varphi'_N, \varphi_P \rangle \quad \text{where } \varphi'_N(x') = \begin{cases} n & \text{if } x' = x, \\ \varphi_N(x') & \text{otherwise,} \end{cases}$$

for names x' in \mathcal{N}_N . Similarly, state $\varphi\{P/z\}$ is defined for procedures P and names z in \mathcal{N}_P as follows:

$$\varphi\{P/z\} := \langle \varphi_N, \varphi'_P \rangle \quad \text{where } \varphi'_P(z') = \begin{cases} P & \text{if } z' = z, \\ \varphi_P(z') & \text{otherwise,} \end{cases}$$

for names z' in \mathcal{N}_P .

Definition 4.1.4. For procedures P, P' , and states φ, φ' , we define transition relation $\langle P \mid \varphi \rangle \rightarrow \langle P' \mid \varphi' \rangle$ as follows:

$$\begin{aligned} \langle x = E ; P \mid \varphi \rangle &\rightarrow \langle P \mid \varphi\{E^\varphi/x\} \rangle, \\ \langle z = F ; P \mid \varphi \rangle &\rightarrow \langle P \mid \varphi\{F^\varphi/z\} \rangle, \\ \langle \text{exec } F ; P \mid \varphi \rangle &\rightarrow \langle F^\varphi ; P \mid \varphi \rangle, \\ \langle \text{if } E \ P_1 \ P_2 ; P \mid \varphi \rangle &\rightarrow \langle P_1 ; P \mid \varphi \rangle \quad \text{if } E^\varphi > 0, \\ \langle \text{if } E \ P_1 \ P_2 ; P \mid \varphi \rangle &\rightarrow \langle P_2 ; P \mid \varphi \rangle \quad \text{if } E^\varphi = 0, \\ \langle \text{while } E \ P_1 ; P \mid \varphi \rangle &\rightarrow \langle \text{if } E \ \{P_1 ; \text{while } E \ P_1\} \text{id} ; P \mid \varphi \rangle, \end{aligned}$$

and, if $\langle P_1 \mid \varphi \rangle \rightarrow^* \langle \text{id} \mid \varphi' \rangle$ for a state φ' then

$$\langle \text{local } P_1 \ \text{export } \overset{x_1 \dots x_h}{z_1 \dots z_k} ; P \mid \varphi \rangle \rightarrow \langle P \mid \psi \rangle$$

where $\psi = \varphi\{x_1^{\varphi'}/x_1\} \dots \{x_h^{\varphi'}/x_h\} \{z_1^{\varphi'}/z_1\} \dots \{z_k^{\varphi'}/z_k\}$.

4.2 Implementation of PROC in λ^*

In the subsequent discussion, we assume that the set \mathcal{N}_N of names for numbers and the set \mathcal{N}_P of names for procedures are disjoint finite sets $\{\bar{x}_1, \bar{x}_2, \dots, \bar{x}_a\}$ and $\{\bar{z}_1, \bar{z}_2, \dots, \bar{z}_b\}$ respectively for some nonnegative integers a and b . Note that we can adopt such assumption when computing a procedure in PROC, since every procedure contains only finitely many names. Furthermore, we assume that the set of names of level 1 in calculus λ^* contains all names in PROC.

Notation. In this section, applications of each level are left-associative, and the body of an abstraction extends as far right as possible, in a customary way. Upper-case letters are names of level 2, and lower-case letters are names of level 1. We omit marks '@₁' for applications of level 1. For instance, $(\lambda M. \text{Mxy}) @_2 M @_2 N$ signifies $((\lambda M. ((M @_1 x) @_1 y)) @_2 M) @_2 N$.

Definition 4.2.1. To N-expressions E , P-expressions F , commands C and procedures P

in PROC, we assign terms $\llbracket E \rrbracket$, $\llbracket F \rrbracket$, $\llbracket C \rrbracket$ and $\llbracket P \rrbracket$ in calculus λ^* respectively, as follows:

$$\begin{aligned} \llbracket x \rrbracket &:= x, & \llbracket 0 \rrbracket &:= \text{zero}, & \llbracket n+1 \rrbracket &:= \text{succ}[\llbracket n \rrbracket], & \llbracket \text{plus } E_1 \ E_2 \rrbracket &:= \text{plus}[\llbracket E_1 \rrbracket][\llbracket E_2 \rrbracket], \\ \llbracket \text{minus } E_1 \ E_2 \rrbracket &:= \text{minus}[\llbracket E_1 \rrbracket][\llbracket E_2 \rrbracket], & \llbracket z \rrbracket &:= z, & \llbracket \text{proc } P \rrbracket &:= \text{block} @_2 [\llbracket P \rrbracket], \\ \llbracket (x = E) \rrbracket &:= \text{set}_x @_2 [\llbracket E \rrbracket], & \llbracket (z = F) \rrbracket &:= \text{set}_z @_2 [\llbracket F \rrbracket], & \llbracket (\text{exec } F) \rrbracket &:= \text{unblock} @_2 [\llbracket F \rrbracket], \\ \llbracket (\text{if } E \ P_1 \ P_2) \rrbracket &:= \text{if} @_2 [\llbracket E \rrbracket] @_2 [\llbracket P_1 \rrbracket] @_2 [\llbracket P_2 \rrbracket], & \llbracket (\text{while } E \ P) \rrbracket &:= \text{while} @_2 [\llbracket E \rrbracket] @_2 [\llbracket P \rrbracket], \\ \llbracket (\text{local } P \ \text{export } \overset{x_1 \dots x_k}{z_1 \dots z_k}) \rrbracket &:= \text{local} \overset{x_1 \dots x_k}{z_1 \dots z_k} @_2 [\llbracket P \rrbracket], \\ \llbracket \text{id} \rrbracket &:= \text{id}, & \llbracket C ; P \rrbracket &:= \text{comp} @_2 (\llbracket C \rrbracket) @_2 [\llbracket P \rrbracket], \end{aligned}$$

where

$$\begin{aligned} \text{zero} &= \lambda s. \lambda z. z, & \text{fst} &= \lambda a. \lambda b. a, & \text{snd} &= \lambda a. \lambda b. b, & \text{succ} &= \lambda n. \lambda s. \lambda z. s(\text{nsz}), \\ \text{pred} &= \lambda n. n(\lambda p. \lambda c. c(p \ \text{snd})(\text{succ}(p \ \text{snd}))) (\lambda c. c(\text{zero}) \ \text{zero}) \ \text{fst}, & \text{plus} &= \lambda m. \lambda n. n(\text{succ})m, \\ \text{minus} &= \lambda m. \lambda n. n(\text{pred})m, & \text{block} &= \lambda Y. \lambda x. (x @_2 Y), & \text{unblock} &= \lambda Z. Z(\lambda Y. Y), \\ \text{set}_v &= \lambda E. \lambda K. (\lambda v. K)E, & \text{if} &= \lambda E. \lambda P. \lambda Q. \lambda K. E(\lambda z. \text{fst}) \ \text{snd}(P @_2 K)(Q @_2 K), \\ \text{while} &= \text{fix} @_2 (\lambda W. \lambda E. \lambda P. (\text{if} @_2 E @_2 (\text{comp} @_2 P @_2 (W @_2 E @_2 P)) @_2 \text{id})), \\ \text{fix} &= \lambda F. ((\lambda X. (F @_2 (X @_2 X))) @_2 (\lambda X. (F @_2 (X @_2 X)))), \\ \text{local} \overset{x_1 \dots x_k}{z_1 \dots z_k} &= \lambda P. \lambda K. (\lambda e. \lambda r. (e @_2 K)(r @_2 x_1) \dots (r @_2 z_k)) (\lambda K. \lambda x_1. \dots \lambda z_k. K)P, \\ \text{id} &= \lambda K. K, & \text{comp} &= \lambda P. \lambda Q. \lambda K. (P @_2 (Q @_2 K)). \end{aligned}$$

Definition 4.2.2. For each state φ in PROC, we define $\llbracket \varphi \rrbracket$ as the substitution $\sigma_N \cdot \sigma_P$ in calculus λ^* , where σ_N and σ_P are the substitutions defined as follows:

$$\sigma_N := \bar{x}_1 \downarrow (\llbracket \bar{x}_1^\varphi \rrbracket) \cdot \dots \cdot \bar{x}_a \downarrow (\llbracket \bar{x}_a^\varphi \rrbracket), \quad \sigma_P := \bar{z}_1 \downarrow (\text{block} @_2 [\llbracket \bar{z}_1^\varphi \rrbracket]) \cdot \dots \cdot \bar{z}_b \downarrow (\text{block} @_2 [\llbracket \bar{z}_b^\varphi \rrbracket]).$$

We also define a relation $\sigma \approx_0 \tau$ for substitutions σ and τ as follows:

$$\sigma \approx_0 \tau \quad \text{if and only if} \quad \sigma \langle v, 0 \rangle = \tau \langle v, 0 \rangle \text{ for each name } v.$$

Notation. In what follows, the relation $\rightarrow_\beta^* \cdot \simeq$ is denoted by \rightsquigarrow_β , and the relation $\rightarrow_\beta^* \cdot \simeq \cdot \leftarrow_\beta^*$ is denoted by \cong_β .

With the above setting, a transition sequence in PROC is simulated by the corresponding reduction sequence in calculus λ^* , as stated in the following propositions.

Proposition 4.2.3. Let P be a procedure, φ and φ' states, and σ a substitution. If $\langle P \mid \varphi \rangle \rightarrow^* \langle \text{id} \mid \varphi' \rangle$ and $\sigma \approx_0 \llbracket \varphi \rrbracket$, then we have $\llbracket P \rrbracket * \sigma \rightsquigarrow_\beta \llbracket \text{id} \rrbracket * \sigma'$ for a substitution σ' such that $\sigma' \approx_0 \llbracket \varphi' \rrbracket$.

Proof. Straightforward by induction on size of transition sequence $\langle P \mid \varphi \rangle \rightarrow^* \langle \text{id} \mid \varphi' \rangle$. \square

Remark 4.2.4. Stated differently, we have the following as an alternative to Proposition 4.2.3. Let P be a procedure, φ and φ' states, and σ a substitution. If $\langle P \mid \varphi \rangle \rightarrow^* \langle \text{id} \mid \varphi' \rangle$ and $\sigma \approx_0 \llbracket \varphi \rrbracket$, then we have

$$\text{comp} @_2 (\lambda K. K[\sigma]) @_2 [\llbracket P \rrbracket] \rightsquigarrow_\beta \lambda K. K[\sigma']$$

for a substitution σ' such that $\sigma' \approx_0 \llbracket \varphi' \rrbracket$.

Example 4.2.5. Let P be the following procedure in PROC that corresponds to the Ruby program in Figure 4.1:

```

sum = proc { n = ar;
  ct = plus ct 1;
  if n {
    local {ar = minus n 1; exec sum} export rv, ct;
    rv = n + rv
  } { rv = 0 }
}; ct = 0; local {ar = 3; exec sum} export rv, ct

```

Then $\llbracket P \rrbracket$ is reduced to the following term in normal form:

$$\llbracket P \rrbracket \rightsquigarrow_{\beta} \lambda K.K[\text{ct}\downarrow(\hat{4}) \cdot \text{ct}\downarrow(\hat{0}) \cdot \text{rv}\downarrow(\hat{6}) \cdot \text{sum}\downarrow(M)]$$

where $\hat{0}$, $\hat{4}$ and $\hat{6}$ are the Church numerals of 0, 4 and 6 respectively, and M is the term that corresponds to the procedure assigned to name `sum`.

Figure 4.1. Sum.rb

```

1: def sum(n)
2:   $ct = $ct + 1
3:   if n > 0
4:     n + sum(n - 1)
5:   else 0 end
6: end
7: $ct = 0
8: sum(3)

```

Figure 4.2. DynamicLiar.rb

```

1: p = "not(eval(p))"
2: print eval(p)

```

Figure 4.3. StaticLiar.rb

```

1: p = not(p)
2: print p

```

4.3 Recursion via Names and Textual Substitution

In PROC, we can write recursive procedures in the usual manner. This feature stems from the fact that we can store and retrieve procedures textually via names, and the feature is implemented with terms *block* and *unblock* in calculus λ^* . Actually, the terms *block* and *unblock* behave as quotes and ‘eval’ command in programming languages. For example, consider the following term L that corresponds to the Ruby program in Figure 4.2:

$$L = (\lambda p.(unblock @_2 p))(block @_2 (n(unblock @_2 p))).$$

Then we have the following infinite reduction sequence that corresponds to the infinite loop caused by the program in Figure 4.2:

$$L \rightsquigarrow_{\beta} M \rightsquigarrow_{\beta} nM \rightsquigarrow_{\beta} n(nM) \rightsquigarrow_{\beta} n(n(nM)) \rightsquigarrow_{\beta} \dots$$

where $M = (\lambda Z.Z[\text{P}\downarrow(N)](\lambda Y.Y[\text{P}\downarrow(N)])) @_2 p$ with $N = block @_2 (n(unblock @_2 p))$. Note that the terms *block* and *unblock* play fundamental roles to generate the infinite reduction sequence. In fact, if we remove the applications of the terms *block* and *unblock* in term L , then we get the term $(\lambda p.p)(np)$, which corresponds to the Ruby program in Figure 4.3 that causes no infinite loops. From viewpoint of variable binding, quotes are considered as blocks to variable binding discussed in Section 2.5, and ‘eval’ command destroys blocks to variable binding. The terms *block* and *unblock* behave in the same way.

4.4 A Comment from Viewpoint of Type Systems

To make a brief comment about properties of calculus λ^* , we introduce a rough type system similar to the simple type system for the lambda calculus.

Definition 4.4.1. We assume that we are given a set of atomic types. The set Typ of types is defined inductively as follows:

$$\begin{aligned} T &\in Typ && \text{if } T \text{ is an atomic type,} \\ T_1 \rightarrow_\ell T_2 &\in Typ && \text{if } \ell \text{ is a level and } T_1, T_2 \in Typ. \end{aligned}$$

Definition 4.4.2. A function of \mathcal{N} into Typ is called a **type assignment**. For type assignments ξ , terms M , substitutions σ , and types T , we define typing relations $\xi \vdash M : T$ and $\xi \vdash \sigma$ inductively as follows:

$$\begin{aligned} \xi \vdash v^d[\sigma] : \xi(v) &&& \text{if } \xi \vdash \sigma, \\ \xi \vdash \lambda v.M : \xi(v) \rightarrow_\ell T &&& \text{if } \ell = \text{Lv}(v) \text{ and } \xi \vdash M : T, \\ \xi \vdash M_1 @_\ell M_2 : T' &&& \text{if } \xi \vdash M_1 : T \rightarrow_\ell T' \text{ and } \xi \vdash M_2 : T, \\ \xi \vdash \text{id}, &&& \\ \xi \vdash v \downarrow(M) \cdot \sigma &&& \text{if } \xi \vdash M : \xi(v) \text{ and } \xi \vdash \sigma, \\ \xi \vdash \uparrow_v \cdot \sigma &&& \text{if } \xi \vdash \sigma. \end{aligned}$$

With the rough type system defined above, we can have subject reduction property stated by the following proposition.

Proposition 4.4.3. Let ξ be a type assignment, T a type, M and M' terms. If $\xi \vdash M : T$ and $M \rightarrow_\beta M'$, then $\xi \vdash M' : T$.

However, the rough type system does not provide us strong normalization property, in contrast to the type system for calculus $\lambda\mathcal{M}$ [18]. For example, consider a type assignment ξ_x satisfying the following conditions:

$$\xi_x(\mathbf{n}) = B \rightarrow_1 B, \quad \xi_x(\mathbf{p}) = \xi_x(\mathbf{z}) = \text{Block}(B), \quad \xi_x(\mathbf{x}) = B \rightarrow_2 B, \quad \xi_x(\mathbf{y}) = B,$$

where B is a type, and $\text{Block}(B)$ signifies $(B \rightarrow_2 B) \rightarrow_1 B$. Then we have the following:

$$\xi_x \vdash \text{block} : B \rightarrow_2 \text{Block}(B), \quad \xi_x \vdash \text{unblock} : \text{Block}(B) \rightarrow_2 B, \quad \xi_x \vdash L : B,$$

where L is the term defined in Section 4.3. The term L has type B , and thus we may expect that the term L signifies a value of type B . However, the term L actually has no normal form and hence signifies nothing.

In order to eliminate such unwanted situations and achieve strong normalization property, we need a more elaborate type system. In fact, the rough type system permits term $L' = \lambda \mathbf{n}.L$ to have type $(B \rightarrow_1 B) \rightarrow_1 B$ in the type assignment ξ_x , whereas L' is a fixed-point operator of level 1 in a sense that we have $L'M \cong_\beta M(L'M)$ for any term M . A key difference from calculus $\lambda\mathcal{M}$ providing strong normalization property is caused by the existence of cross-level terms that bring on level-increasing reduction, such as object-level β -reduction generating new meta-level β -redexes. A term is said to be **cross-level** if the term can be typed only in cross-level type assignments. A type assignment ξ is said to

be cross-level if there exists a name v such that the type $\xi(v)$ of v in the type assignment ξ contains an arrow ' \rightarrow_ℓ ' of level ℓ greater than $\text{Lv}(v)$. For instance, the above ξ_x is a cross-level type assignment, and the terms *block* and $\text{local}_{z_1 \dots z_k}^{x_1 \dots x_h}$ are cross-level terms as well as the term L mentioned above. Cross-level type assignments and cross-level terms are seemingly meaningless. Hence, cross-level terms have been left out of consideration by type systems or by level-controlled reductions in previous meta lambda calculi. However, we consider that cross-level terms may be worth investigating, since cross-level terms seem to have connections with notions related to names and bindings in programming languages, such as stores, quotes, recursion, and localization of names, as demonstrated in this section. We leave further research about this topic as a future work.

Chapter 5

Related Works

5.1 Meta Lambda Calculi

We discuss connections with previous meta lambda calculi that include inherently textual substitution via meta-level variables.

Calculus $\lambda\mathcal{M}$ proposed by Sato, Sakurai, Kameyama and Igarashi [18] is a meta lambda calculus with infinitely hierarchical levels. $\lambda\mathcal{M}$ adopts level-controlled reductions and a type system to achieve preferable properties in coexistence with textual substitution. $\lambda\mathcal{M}$ is actually a subsystem of λ^* consisting only of annotation-free terms. The level-controlled reduction in $\lambda\mathcal{M}$ is viewed as a restriction of β -reduction in λ^* on a set of annotation-free terms.

Calculus LamCC in Gabbay and Lengrand [10] also includes infinitely hierarchical levels. LamCC adopts level-controlled reductions and explicit substitutions. Unlike other meta lambda calculi, LamCC does not have the notion of level of application. In other words, the level of each application occurring in a term is determined dynamically in the process of computation. One of the features of λ^* different from LamCC is that substitutions in λ^* have canonical representation. By this feature, terms signifying the same substitution, for example, $(\lambda x.\lambda y.M)Nz$ and $(\lambda y.\lambda x.M)zN$ are reduced to the same term $M[x\downarrow(N) \cdot y\downarrow(z)]$ by β -reduction and ε -reduction in λ^* . In LamCC, the above two terms are reduced to distinct two terms in normal form.

Gabbay's NEW calculus of contexts [9] and two-level lambda calculus [11] adopt level-controlled reductions and freshness contexts. A freshness context is regarded as an assumption about freshness conditions for meta-level variables. Reductions are controlled by freshness contexts. In a word, a term is reduced under some assumption about freshness conditions for meta-level variables.

The NEW calculus and LamCC mentioned above include NEW binders ' \mathcal{U} ' separately from ordinary lambda binders ' λ '. A NEW binder indicates that an object-level variable is fresh for meta-level variables. For instance, $\mathcal{U}x.\lambda x.Mx \rightarrow_\alpha \mathcal{U}y.\lambda y.My$ holds, since the object-level variables ' x ' and ' y ' in the above terms are considered to be fresh for meta-level variable M . In calculus λ^* , such information is represented by pop-elements. The above two terms and the α -renaming are represented in λ^* as $\lambda x.M[\uparrow_x]x \rightarrow_{\alpha \rightarrow \varepsilon} \lambda y.M[\uparrow_y]y$.

Bekki's meta-lambda calculus [2] is a study of categorical semantics for metavariables. The formalization shown in Bekki and Asai [3] and in Masuko and Bekki [13] adopts assumption about free variables for meta-level variables in order to perform α -renaming in the presence of meta-level variables. The notion of blocks to variable binding discussed in Section 2.5 is pointed out in [13].

5.2 Other Works

Attempts to model textual substitution via metavariables in a calculus are originated from Hashimoto and Ohori’s typed context calculus [12], which is designed to internalize the notion of lambda contexts. Their calculus adopts level-controlled reductions and a mechanism, called renamers, to manage binding structure consistently with the notion of holes to represent lambda contexts. Holes are regarded as meta-level variables, and lambda contexts are regarded as meta-level abstractions from viewpoint of meta lambda calculi. The technique of renamers is considered a kind of approaches by interfaces assigned to meta-level variables mentioned in Section 1.3. Sato, Sakurai and Kameyama’s simply typed context calculus [16], calculus λm in Sato et al. [18], Nanevski, Pientka and Pfenning’s calculi [14, 15] with modal types, and Boespflug and Pientka [4] with multi-level modal types are also designed by approaches of interfaces assigned to metavariables. These calculi are called lambda calculi with interfaces distinctively from meta lambda calculi in this paper. The goal of these calculi to provide type systems for meta-level variables seems to lead the design to use information about interfaces assigned to meta-level variables not only in type systems but also in rewriting systems. This feature of the design makes the difference between these calculi and meta lambda calculi as illustrated in Section 1.3. The technique of indexed variables is adopted in [16].

The syntax of calculus λ^m in Davies [7] is similar to the syntax of λ^* in a sense that variables and applications are assigned with numbers. The numbers in calculus λ^* signify levels of variables, which determine *strength* of substitution so that substitution of level ℓ is performed by regarding variables of level less than ℓ as mere texts. In contrast, the numbers in calculus λ^m signify stages of computation, which determine *time* of substitution. In λ^m , all variables occurring in a term are bound statically as usual, unlike meta lambda calculi.

Methods to deal with substitutions as syntactic objects date back to Abadi, Cardelli, Curien and Lévy’s explicit substitutions [1]. Dowek, Hardin and Kirchner [8] applies the method to the nameless lambda calculus with meta-level variables, which includes neither meta-level abstraction nor meta-level application. The formalization of substitutions with push-elements and pop-elements in λ^* is almost the same as their formalization, although substitutions in λ^* are sorted by names and defined to occur only as suspended substitutions on meta-level variables. The slight difference from their formalization stems from the purpose to make calculus λ^* become a supersystem of $\lambda\mathcal{M}$ [18] as well as the ordinary lambda calculus. In other words, this paper does not concern itself about modeling concrete way and cost of performing substitution, which is the original purpose of explicit substitutions in Abadi et al.

Calculus λN proposed by Dami [6] is an extension of the lambda calculus to model dynamic binding. The syntax of λN includes additional constructs called labels separately from ordinary variables. A main difference between λN and λ^* is their style of dynamic binding. Calculus λ^* deals with dynamic binding by lambda binders via meta-level variables, whereas λN deals with dynamic binding by labels, not by lambda binders.

Chapter 6

Conclusion

We have proposed meta lambda calculus λ^* , in which any β -redex of any level can be reduced to perform substitution for variables. This feature makes it possible to advance computation even in the presence of meta-level variables, and hence provides us new possibilities for reduction strategies that have been restricted by level-controlled reductions in previous meta lambda calculi.

Also, we have shown a procedural language as an application of calculus λ^* . Through the implementation of the procedural language, we have observed the connections between dynamic binding via meta-level variables and the notions of stores and recursion in procedural languages. We hope that calculus λ^* contributes toward understanding metavariables and the association with notions related to names and bindings in programming languages.

Acknowledgements

I wish to thank my supervisor, Ryu Hasegawa, sincerely for the fruitful discussions. I also thank my colleagues for helpful comments.

Bibliography

- [1] M. Abadi, L. Cardelli, P.-L. Curien and J.-J. Lévy. 1991. Explicit substitutions. *Journal of Functional Programming* 1, 4, 375–416.
- [2] Daisuke Bekki. 2009. Monads and meta-lambda calculus. In *New Frontiers in Artificial Intelligence (JSAI 2008)*, LNAI 5447, 193–208.
- [3] Daisuke Bekki and Kenichi Asai. 2010. Representing covert movements by delimited continuations. In *New Frontiers in Artificial Intelligence (JSAI-isAI 2009)*, LNAI 6284, 161–180.
- [4] Mathieu Boespflug and Brigitte Pientka. 2011. Multi-level contextual type theory. In *Proceedings of the 6th International Workshop on Logical Frameworks and Meta-languages: Theory and Practice (LFMTP 2011)*, EPTCS 71, 29–43.
- [5] N. G. de Bruijn. 1972. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae* 75, 5, 381–392.
- [6] Laurent Dami. 1998. A lambda-calculus for dynamic binding. *Theoretical Computer Science* 192, 2, 201–231.
- [7] Rowan Davies. 1996. A temporal-logic approach to binding-time analysis. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS '96)*, 184–195.
- [8] Gilles Dowek, Thérèse Hardin and Claude Kirchner. 2000. Higher order unification via explicit substitutions. *Information and Computation* 157, 183–235.
- [9] Murdoch J. Gabbay. 2005. A NEW calculus of contexts. In *Proceedings of the 7th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP '05)*, 94–105.
- [10] Murdoch J. Gabbay and Stéphane Lengrand. 2009. The lambda-context calculus (extended version). *Information and Computation* 207, 12, 1369–1400.
- [11] Murdoch J. Gabbay and Dominic P. Mulligan. 2009. Two-level lambda-calculus. *Electronic Notes in Theoretical Computer Science* 246, 107–129.
- [12] Masatomo Hashimoto and Atsushi Ohori. 2001. A typed context calculus. *Theoretical Computer Science* 266, 249–272.
- [13] Moe Masuko and Daisuke Bekki. 2011. Categorical semantics of meta-lambda calculus (in Japanese). In *Informal Proceedings of the 13th JSSST Workshop on Programming and Programming Languages*, 60–74.
- [14] Aleksandar Nanevski, Brigitte Pientka and Frank Pfenning. 2003. A modal foundation for meta-variables. In *Proceedings of the 2003 ACM SIGPLAN Workshop on Mechanized Reasoning about Languages with Variable Binding (MERLIN '03)*, 1–6.
- [15] Aleksandar Nanevski, Frank Pfenning and Brigitte Pientka. 2008. Contextual modal type theory. *ACM Transactions on Computational Logic* 9, 3, Article 23, 49 pages.
- [16] Masahiko Sato, Takafumi Sakurai and Yuki Yoshi Kameyama. 2002. A simply typed context calculus with first-class environments. *Journal of Functional and Logic Programming* 2002, 1–41.

- [17] Masahiko Sato, Takafumi Sakurai, Yuki Yoshi Kameyama and Atsushi Igarashi. 2003. Calculi of meta-variables. In *Computer Science Logic (CSL 2003)*, LNCS 2803, 484–497.
- [18] Masahiko Sato, Takafumi Sakurai, Yuki Yoshi Kameyama and Atsushi Igarashi. 2008. Calculi of meta-variables. *Frontiers of Computer Science in China* 2, 1, 12–21.
- [19] Masako Takahashi. 1995. Parallel reductions in λ -calculus. *Information and Computation* 118, 1, 120–127.
- [20] Glynn Winskel. 1993. *The Formal Semantics of Programming Languages*. The MIT Press.