A Curry-Howard Isomorphism for Compilation and Program Execution (Extended Abstract)*

Atsushi Ohori**

Research Institute for Mathematical Sciences
Kyoto University, Kyoto 606-8502, Japan
ohori@kurims.kyoto-u.ac.jp

Abstract. This paper establishes a Curry-Howard isomorphism for compilation and program execution by showing the following facts. (1) The set of *A-normal forms*, which is often used as an intermediate language for compilation, corresponds to a subsystem of Kleene's contraction-free variant of Gentzen's intuitionistic sequent calculus. (2) Compiling the lambda terms to the set of A-normal forms corresponds to proof transformation from the natural deduction to the sequent calculus followed by proof normalization. (3) Execution of an A-normal form corresponds to a special proof reduction in the sequent calculus. Different from cut elimination, this process eliminates left rules by converting them to cuts of proofs corresponding to closed values. The evaluation of an entire program is the process of inductively applying this process followed by constructing data structures.

1 Introduction

Curry-Howard isomorphism [3, 11] is one of most influential concepts in design and analysis of programming languages. It reveals the exact correspondence between the typed lambda calculus and the natural deduction proof system: typing derivations correspond to proofs and β reduction corresponds to proof normalization. This notion is, however, not entirely appropriate for an actual programming language because of the apparent mismatch between β reduction and language implementation. In actual programming languages (except for some interpreted languages) a program is not β reduced but instead is compiled to a low-level code and then executed by an (abstract) machine. Because of this mismatch, the profound correspondence between β reduction and proof normalization does not have much significance in language implementation. If Curry-Howard isomorphism is extended to implementation process, then research on compilation

* This is an author's version of the paper publised in **Proceedings of Typed Lambda Calculi and Applications, Springer LNCS 1581, 258-179, 1999.**

** This work was partly supported by the Japanese Ministry of Education Grant-in-Aid for Scientific Research on Priority Area no. 275 "Advanced databases," and by the Parallel and Distributed Processing Research Consortium, Japan.

and implementation would be greatly benefited through high-level logical analysis made available by the extended isomorphism. This would be particularly useful for recent active researches on types in compilation where compilation is directed by typing derivation. The goal of this paper is to establish a Curry-Howard isomorphism for compilation and program execution.

There are several formalisms for compilation and program execution. Here we base our development on the work by Flanagan et. al. [6] for a call-by-value functional language using an intermediate language called *A-normal forms*, which is equivalent to the language obtained from CPS terms by "un-CPS" transformation [5, 19] that eliminates continuation. In this formalism, compilation is modeled by transformation from lambda terms into A-normal forms, and program execution is defined by an abstract machine for A-normal forms. As forcefully argued by Flanagan et al, compiling into A-normal forms can be regarded as "the essence" of compiling a functional language, and the execution model for A-normal forms closely reflects an actual implementation of a functional language using environments. They give a simple linear time compilation algorithm and demonstrate that it can be used as a basis for an efficient practical compiler through their experimentation. Because of these facts, we also believe that compiling with A-normal forms can serve as an realistic model for efficient implementation of functional languages.

Our specific goal is therefore to develop logical foundations for compiling the lambda terms to the set of A-normal forms and for evaluation of A-normal forms. We achieve this goal by establishing the following facts.

1. A logic that corresponds to a language for mechanical execution in a conventional computer system is a Gentzen-style sequent calculus, which represents finer notion of computation than the natural deduction system. Instead of performing general substitution, it decomposes a computation on a data type into smaller structures by the corresponding left rule. In particular, Kleene's [13] contraction-free sequent calculus, denoted here by $\mathcal{GK}$, serves as a logic for an implementation language. The set of A-normal forms is identified with a subsystem $\mathcal{GKA}$ whose proofs are those of $\mathcal{GK}$ in a certain normal form.
2. A compilation algorithm from lambda terms to A-normal forms in the style of [6] corresponds to the composition of a proof transformation from the natural deduction system (denoted here by $\mathcal{N}$) to $\mathcal{GK}$ and a proof normalization from $\mathcal{GK}$ to $\mathcal{GKA}$.
3. Execution of an A-normal form corresponds to a special proof reduction process in $\mathcal{GKA}$. Different from cut elimination, this process eliminates left rules by converting them to cuts of proofs corresponding to closed values. The evaluation of an entire program is the process of inductively applying this process followed by constructing data structures. This process exactly corresponds to execution of a program using environments.

These results establish a *Curry-Howard isomorphism* for compilation and program execution. The summary of the correspondence is shown in Fig. 1.

Intuitively, an A-normal compiler performs two types of transformations: (1) it identifies all the redexes by naming the intermediate results of reductions,

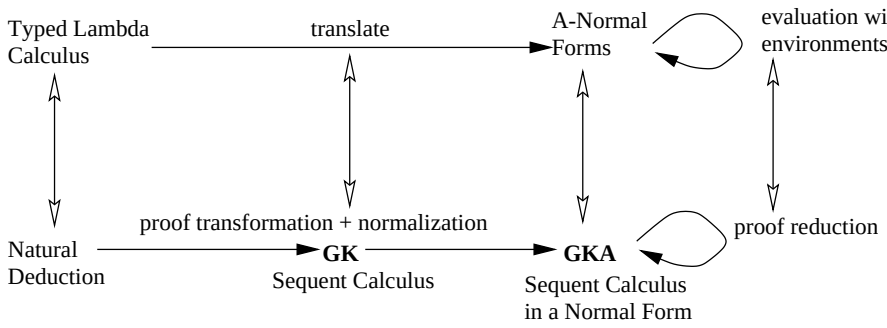

Fig. 1. Curry-Howard isomorphism for compilation and program execution

and (2) it flattens and linearizes redexes by extending the scope of intermediate bindings. As a simple example, consider the source term $(f\ (g\ x))$. The first type of transformation converts this term into the following:

$$\mathbf{app}\ (f\ (\mathbf{app}\ (g\ x)\ \mathbf{is}\ z\ \mathbf{in}\ z))\ \mathbf{is}\ w\ \mathbf{in}\ w$$

where $\mathbf{app}\ (x\ M)\ \mathbf{is}\ y\ \mathbf{in}\ N$ is our syntax in $\mathcal{GK}$ for applying function x to M and naming the result as y in N. This is exactly the transformation of a natural deduction proof to a sequent calculus proof. The second type of transformation converts this into the following A-normal form:

$$\mathbf{app}\ (g\ x)\ \mathbf{is}\ z\ \mathbf{in}\ (\mathbf{app}\ (f\ z)\ \mathbf{is}\ w\ \mathbf{in}\ w)$$

This process is the proof normalization from $\mathcal{GK}$ to $\mathcal{GKA}$. Execution of an A-normal form is also tightly modeled in $\mathcal{GKA}$: an operational semantics of A-normal forms exactly corresponds to a proof reduction in $\mathcal{GKA}$.

We believe that those logical correspondences worked out in this paper will contribute to design, analysis and optimization of compilation in a higher-order functional language. As an example of one of such benefits, A-normal compilation of [6] is immediately extended to products and sums by using the corresponding logical principles, as seen in this paper.

Related work. Before giving the technical development, we compare the results presented in this paper with related works. The use of a Gentzen-style sequent calculus as a model of computation is not new. Abramsky [1] has given a term calculus for linear logic. Breaze-Tannen et. al. [2] have given a typed pattern calculus where the underlying logic is a sequent calculus. In [4, 10] a sequent calculus is regarded as a model of computation. In particular, Herbelin [10] has argued that a sequent calculus can be a basis for computation and presented a term calculus. Based on a similar observation, Ogata [16] has shown that the term calculus presented in [4] corresponds to CPS terms under Griffin's [9] interpretation of CPS terms. In a tutorial article, Gallier [7] has given a term calculus for a Gentzen sequent calculus and suggested that a Gentzen-style

sequent calculus represents finer notion of computation than β reduction. In general perspective, all those term calculi have the similarity to ours in the sense that they represent refined notion of computation, and they have been source of inspiration of the present paper. However, to the author's knowledgement, the connection to compilation and execution of compiled code has not been investigated.

In establishing this connection in the present paper, we use a well known result stating that any natural deduction proof can be transformed into a proof in the sequent calculus. Zucker [21] and Pottinger [17] conducted extensive studies on the relationship between the two proof systems. As we will show, however, this relationship alone does not provide the desired interpretation for compilation and program execution, and significant new results are needed to extend Curry-Howard isomorphism to them. In the existing works on Gentzen's sequent calculus and computation, the advocated thesis is that "cut elimination corresponds to computation." Our analysis shows, however, that this commonly believed thesis does not apply to actual implementation of a (call-by-value) functional language. In the usual cut elimination, cut rule is inductively moved upward to smaller proofs. A somewhat surprising result of our work is that program evaluation in conventional implementation pushes cut downwards, and corresponds a quite difference proof normalization process.

Paper Organization. Section 2 defines the typed lambda calculus. Section 3 defines $\mathcal{GK}$, $\mathcal{GKA}$, and a proof normalization from $\mathcal{GK}$ to $\mathcal{GKA}$. Section 4 shows that a compilation algorithm from lambda terms to A-normal forms is the combination of a proof transformation from the natural deduction to $\mathcal{GK}$ and a proof normalization from $\mathcal{GK}$ to $\mathcal{GKA}$. Section 5 shows that the operational semantics of A-normal forms is a proof reduction in $\mathcal{GKA}$. Section 6 concludes the paper.

Limitations of space make it difficult to cover the technical development fully; the author intends to present a more detailed description elsewhere.

Acknowledgments. The author would like to thank Yasuhiko Minamide and Ichiro Ogata for useful discussions on A-normal forms and sequent calculi. He also thanks Susumu Nishimura for helpful comments on a draft of this paper.

2 Typed Lambda Calculus

To make the relationship to logic explicit, we use the following logical notations for types (ranger over by τ):

$$\tau ::= b \mid \tau \supset \tau \mid \tau \wedge \tau \mid \tau \vee \tau$$

where b stands for a given set of atomic types. A *type assignment* Γ is a function from a finite set of variables to types. We write $\{x_1 : \tau_1, \ldots, x_n : \tau_n\}$ for the function that maps each x_i to τ_i $(1 \leq i \leq n)$. If f is a function, we write $f, x : \tau$ for the function f' such that $dom(f') = dom(f) \cup \{x\}$ and $f'(x) = \tau$,

$f'(y) = f(y)$ if $y \neq x$. The set of terms is given by the following syntax:

$$M ::= c^b \mid x \mid \lambda x : \tau.M \mid M\ M \mid (M, M) \mid M.1 \mid M.2 \mid \\ \mathbf{in1}(M : \tau) \mid \mathbf{in2}(M : \tau) \mid \mathbf{case}\ M\ \mathbf{of}\ x.M,\ x.M$$

c^b stands for atomic constants of type b. x stands for a given set of variables. $M.1$ and $M.2$ are first and second projection, respectively. $\mathbf{in1}(M : \tau), \mathbf{in2}(M : \tau)$ are left and right injection to a variant, respectively. The type annotations in $\lambda x : \tau.M$, $\mathbf{in1}(M : \tau)$, and $\mathbf{in2}(M : \tau)$ are necessary to achieve the uniqueness of typing derivation (uniqueness of a term representation of a proof). In what follows, however, we make those type annotations implicit.

$$\text{(axiom)}\ \ \Gamma \rhd c^b : b \qquad \text{(taut)}\ \ \Gamma, x : \tau \rhd x : \tau \qquad (\supset\text{:I})\ \ \frac{\Gamma, x : \tau_1 \rhd M : \tau_1}{\Gamma \rhd \lambda x : \tau.M : \tau_1 \supset \tau_2}$$

$$(\supset\text{:E})\ \ \frac{\Gamma \rhd M_1 : \tau_1 \supset \tau_2 \quad \Gamma \rhd M_2 : \tau_1}{\Gamma \rhd M_1\ M_2 : \tau_2} \qquad (\wedge\text{:I})\ \ \frac{\Gamma \rhd M_1 : \tau_1 \quad \Gamma \rhd M_2 : \tau_2}{\Gamma \rhd (M_1, M_2) : \tau_1 \wedge \tau_2}$$

$$(\wedge\text{:Ei})\ \ \frac{\Gamma \rhd M : \tau_1 \wedge \tau_2}{\Gamma \rhd M.i : \tau_i}\ \ i \in \{1, 2\} \qquad (\vee\text{:Ii})\ \ \frac{\Gamma \rhd M : \tau_i}{\Gamma \rhd \mathbf{in}i(M : \tau_1 \vee \tau_2) : \tau_1 \vee \tau_2}\ i \in \{1, 2\}$$

$$(\vee\text{:E})\ \ \frac{\Gamma \rhd M_1 : \tau_1 \vee \tau_2 \quad \Gamma, x : \tau_1 \rhd M_2 : \tau_3 \quad \Gamma, y : \tau_2 \rhd M_3 : \tau_3}{\Gamma \rhd \mathbf{case}\ M_1\ \mathbf{of}\ x.M_2, y.M_3 : \tau_3}$$

Fig. 2. Typed Lambda Calculus with Products and Sums

The proof system for the typed lambda terms is given in Fig. 2. The following properties are well known as *Curry-Howard isomorphism*.

- If we erase M from $\Gamma \rhd M : \tau$ and replace Γ with the multi-set obtained by erasing the variables, then we obtain the natural deduction system [18] (with additional axioms for atomic propositions), which is denoted here by $\mathcal{N}$.
- If $\vdash \Gamma \rhd M : \tau$ then the term M uniquely represents a proof of $\vdash \Gamma \rhd M : \tau$ in $\mathcal{N}$.
- The β reduction on lambda terms corresponds to proof normalization in $\mathcal{N}$.

We write $\mathcal{N} \vdash \Gamma \rhd M : \tau$ if $\Gamma \rhd M : \tau$ is provable in this proof system. Our aim is to extend this logical correspondence to compilation and program execution using a Gentzen-style sequent calculus.

3 Intuitionistic Sequent Calculus : $\mathcal{GK}$

We choose a contraction-free variant of the Gentzen's intuitionistic sequent calculus due to Kleene [13, Ch.XV,§80], which is particularly suitable for establishing the exact correspondence between program execution and proof reduction.

The set of types is the same as that of $\mathcal{N}$. The set of terms is given by the following syntax.

$$M ::= c^b \mid x \mid \lambda x.M \mid \mathbf{app}\ (x\ M)\ \mathbf{is}\ y\ \mathbf{in}\ M \mid (M, M) \mid \mathbf{proj}\ x\ \mathbf{on}\ (y, z)\ \mathbf{in}\ M \\ \mid\ \mathbf{in1}(M) \mid \mathbf{in2}(M) \mid \mathbf{case}\ x\ \mathbf{of}\ y.M, z.M \mid \mathbf{let}\ x = M\ \mathbf{in}\ M$$

We have explained $\mathbf{app}\ (x\ M_1)\ \mathbf{is}\ y\ \mathbf{in}\ M_2$. $\mathbf{proj}\ x\ \mathbf{on}\ (y, z)\ \mathbf{in}\ M$ binds y to the first component of x, and binds z to the second component of x in M. $\mathbf{case}\ x\ \mathbf{of}\ y.M_1, z.M_2$ performs case analysis on x and if it is of the form $\mathbf{in1}(v)$ then binds x to v in M_1 otherwise x is of the form $\mathbf{in2}(v)$ and it binds z to v in M_2. The proof system is given in Fig. 3.

$$\text{(axiom)}\ \ \Gamma \rhd c^b\ :\ b \qquad \text{(taut)}\ \ \Gamma, x : \tau \rhd x\ :\ \tau \qquad (\supset\text{:R})\ \ \frac{\Gamma, x : \tau_1 \rhd M\ :\ \tau_2}{\Gamma \rhd \lambda x.M\ :\ \tau_1 \supset \tau_2}$$

$$(\supset\text{:L})\ \ \frac{\Gamma, x : \tau_1 \supset \tau_2 \rhd M_1\ :\ \tau_1 \quad \Gamma, x : \tau_1 \supset \tau_2, y : \tau_2 \rhd M_2\ :\ \tau_3}{\Gamma, x : \tau_1 \supset \tau_2 \rhd \mathbf{app}\ (x\ M_1)\ \mathbf{is}\ y\ \mathbf{in}\ M_2\ :\ \tau_3}$$

$$(\wedge\text{:R})\ \ \frac{\Gamma \rhd M_1\ :\ \tau_1 \quad \Gamma \rhd M_2\ :\ \tau_2}{\Gamma \rhd (M_1, M_2)\ :\ \tau_1 \wedge \tau_2} \qquad (\vee\text{:Ri})\ \ \frac{\Gamma \rhd M\ :\ \tau_2}{\Gamma \rhd \mathbf{in}i(M)\ :\ \tau_1 \vee \tau_2}\ \ (i \in \{1, 2\})$$

$$(\wedge\text{:L})\ \ \frac{\Gamma, x : \tau_1 \wedge \tau_2, y : \tau_1, z : \tau_2 \rhd M\ :\ \tau_3}{\Gamma, x : \tau_1 \wedge \tau_2 \rhd \mathbf{proj}\ x\ \mathbf{on}\ (y, z)\ \mathbf{in}\ M\ :\ \tau_3}$$

$$(\vee\text{:L})\ \ \frac{\Gamma, x : \tau_1 \vee \tau_2, y : \tau_1 \rhd M_1\ :\ \tau_3 \quad \Gamma, x : \tau_1 \vee \tau_2, z : \tau_2 \rhd M_2\ :\ \tau_3}{\Gamma, x : \tau_1 \vee \tau_2 \rhd \mathbf{case}\ x\ \mathbf{of}\ y.M_1, z.M_2\ :\ \tau_3}$$

$$\text{(cut)}\ \ \frac{\Gamma \rhd M_1\ :\ \tau_1 \quad \Gamma, x : \tau_1 \rhd M_2\ :\ \tau_2}{\Gamma \rhd \mathbf{let}\ x = M_1\ \mathbf{in}\ M_2\ :\ \tau_2}$$

Fig. 3. Gentzen-style Intuitionistic Sequent Calculus $\mathcal{GK}$

For our calculus, the notion of bound and free variables are defined on both terms and proofs, and we can show that α-equivalence hold in this calculus. In the following development, we assume the "bound variable convention", i.e. all bound variables are distinct and are different from any free variables. It should be noted, however, that α equivalence is not entirely obvious for sequent calculi. For example, if we adopt the Gentzen's original proof system where each left rule introduces a new assumption, then some extra machinery will be needed to obtain α equivalence.

3.1 A-Normal Forms and Proof Normalization

We define a subsystem $\mathcal{GKA}$ of $\mathcal{GK}$ whose proofs correspond to the set of A-normal forms. We say that a premise is an *argument premise* if it is a premise of one of right rules except ($\supset$:R), or it is the left premise of ($\supset$:L) or (cut). $\mathcal{GKA}$ is obtained from $\mathcal{GK}$ by distinguishing those proofs that correspond to "values", and restricting argument premises to be value proofs. The set of values (ranged

over by V) and the set of A-normal forms are given as follows.

$$\begin{aligned} V &::= c^b \mid x \mid \lambda x.M \mid (V,V) \mid \mathbf{in1}(V) \mid \mathbf{in2}(V) \\ M &::= V \mid \mathbf{app}\ (x\ V)\ \mathbf{is}\ y\ \mathbf{in}\ M \mid \mathbf{proj}\ x\ \mathbf{on}\ (y,z)\ \mathbf{in}\ M \mid \\ &\quad \mathbf{case}\ x\ \mathbf{of}\ y.M, z.M \mid \mathbf{let}\ x = V\ \mathbf{in}\ M \end{aligned}$$

The proof system $\mathcal{GKA}$ is given in Fig. 4, where the use of a meta variable V indicates that it must be a value.

Values.

(axiom) $\Gamma \rhd c^b : b$ (taut) $\Gamma, x:\tau \rhd x : \tau$ ($\wedge$:R) $\dfrac{\Gamma \rhd V_1 : \tau_1 \quad \Gamma \rhd V_2 : \tau_2}{\Gamma \rhd (V_1, V_2) : \tau_1 \wedge \tau_2}$

($\vee$:Ri) $\dfrac{\Gamma \rhd V : \tau_i}{\Gamma \rhd \mathbf{in}i(V) : \tau_1 \vee \tau_2}$ $(i \in \{1,2\})$ ($\supset$:R) $\dfrac{\Gamma, x:\tau_1 \rhd M : \tau_2}{\Gamma \rhd \lambda x.M : \tau_1 \supset \tau_2}$

General A-normal forms

($\supset$:L) $\dfrac{\Gamma, x:\tau_1 \supset \tau_2 \rhd V : \tau_1 \quad \Gamma, x:\tau_1 \supset \tau_2, y:\tau_2 \rhd M : \tau_3}{\Gamma, x:\tau_1 \supset \tau_2 \rhd \mathbf{app}\ (x\ V)\ \mathbf{is}\ y\ \mathbf{in}\ M : \tau_3}$

($\wedge$:L) $\dfrac{\Gamma, x:\tau_1 \wedge \tau_2, y:\tau_1, z:\tau_2 \rhd M : \tau_3}{\Gamma, x:\tau_1 \wedge \tau_2 \rhd \mathbf{proj}\ x\ \mathbf{on}\ (y,z)\ \mathbf{in}\ M : \tau_3}$

($\vee$:L) $\dfrac{\Gamma, x:\tau_1 \vee \tau_2, y:\tau_1 \rhd M_1 : \tau_3 \quad \Gamma, x:\tau_1 \vee \tau_2, z:\tau_2 \rhd M_2 : \tau_3}{\Gamma, x:\tau_1 \vee \tau_2 \rhd \mathbf{case}\ x\ \mathbf{of}\ y.M_1, z.M_2 : \tau_3}$

(cut) $\dfrac{\Gamma \rhd V : \tau_1 \quad \Gamma, x:\tau_1 \rhd M : \tau_2}{\Gamma \rhd \mathbf{let}\ x = V\ \mathbf{in}\ M : \tau_2}$

Fig. 4. Proof system $\mathcal{GKA}$ for A-normal forms

We define the set $\mathcal{S}$ of proof transformations from $\mathcal{GK}$ to $\mathcal{GKA}$. Each transformation pushes a cut rule or a left rule appearing in an argument premise downward. For each of {(cut), ($\supset$:L), ($\wedge$:L), ($\vee$:L)} there are 6 transformation rules corresponding to the 6 different argument premises.

The sets of transformations for {($\supset$:L), ($\wedge$:L), (cut)} are similar to one another. Here we only show the two cases where (cut) appears in an argument premise as follows:

$$\dfrac{\dfrac{\begin{matrix}\Delta_1 \\ (\Gamma \rhd M_1 : \tau_1)\end{matrix} \quad \dfrac{\begin{matrix}\Delta_2 \\ (\Gamma, x:\tau_1 \rhd M_2 : \tau_2)\end{matrix}}{\Gamma \rhd \mathbf{let}\ x = M_1\ \mathbf{in}\ M_2 : \tau_2}\ (\text{cut}) \quad \dfrac{\begin{matrix}\Delta_3 \\ \Gamma \rhd M_3 : \tau_3\end{matrix}}{\Gamma \rhd (\mathbf{let}\ x = M_1\ \mathbf{in}\ M_2, M_3) : \tau_2 \wedge \tau_3}\ (\wedge\text{:R})}{}$$

$$\Longrightarrow \dfrac{\begin{matrix}\Delta_1 \\ (\Gamma \rhd M_1 : \tau_1)\end{matrix} \quad \dfrac{\begin{matrix}\Delta_2 \\ (\Gamma, x:\tau_1 \rhd M_2 : \tau_2)\end{matrix} \quad \dfrac{\begin{matrix}\Delta_3 + \{x:\tau_1\} \\ (\Gamma, x:\tau_1 \rhd M_3 : \tau_3)\end{matrix}}{\Gamma, x:\tau_1 \rhd (M_2, M_3) : \tau_2 \wedge \tau_3}\ (\wedge\text{:R})}{\Gamma \rhd \mathbf{let}\ x = M_1\ \mathbf{in}\ (M_2, M_3) : \tau_2 \wedge \tau_3}\ (\text{cut})$$

$$\dfrac{\dfrac{\overset{\Delta_1}{(\Gamma \rhd M_1 \,:\, \tau_1)} \quad \overset{\Delta_2}{(\Gamma, x:\tau_1 \rhd M_2 \,:\, \tau_2)}}{\Gamma \rhd \mathbf{let}\ x = M_1\ \mathbf{in}\ M_2 \,:\, \tau_2}\ (\mathrm{cut}) \quad \overset{\Delta_3}{(\Gamma, y:\tau_2 \rhd M_3 \,:\, \tau_3)}}{\Gamma \rhd \mathbf{let}\ y = (\mathbf{let}\ x = M_1\ \mathbf{in}\ M_2)\ \mathbf{in}\ M_3 \,:\, \tau_3}\ (\mathrm{cut})$$

$$\Longrightarrow \dfrac{\overset{\Delta_1}{(\Gamma \rhd M_1 \,:\, \tau_1)} \quad \dfrac{\overset{\Delta_2}{(\Gamma, x:\tau_1 \rhd M_2 \,:\, \tau_2)} \quad \overset{\Delta_3 + \{x:\tau_1\}}{(\Gamma, x:\tau_1, y:\tau_2 \rhd M_3 \,:\, \tau_3)}}{\Gamma, x:\tau_1 \rhd \mathbf{let}\ y = M_2\ \mathbf{in}\ M_3 \,:\, \tau_3}\ (\mathrm{cut})}{\Gamma \rhd \mathbf{let}\ x = M_1\ \mathbf{in\ let}\ y = M_2\ \mathbf{in}\ M_3 \,:\, \tau_3}\ (\mathrm{cut})$$

where $\overset{\Delta}{(\Gamma \rhd M \,:\, \tau)}$ is a proof of the sequent $\Gamma \rhd M \,:\, \tau$, and $\Delta + \{x:\tau\}$ is the proof obtained from the proof Δ by adding $\{x:\tau\}$ to the assumption of each sequent in Δ.

If we project the set $\mathcal{S}$ of proof transformations on untyped term structures, then they become the following set of reduction rules.

$$C[\mathbf{app}\ (x\ M_1)\ \mathbf{is}\ y\ \mathbf{in}\ M_2] \Longrightarrow \mathbf{app}\ (x\ M_1)\ \mathbf{is}\ y\ \mathbf{in}\ C[M_2]$$
$$C[\mathbf{proj}\ z\ \mathbf{on}\ (w,v)\ \mathbf{in}\ M_1] \Longrightarrow \mathbf{proj}\ z\ \mathbf{on}\ (w,v)\ \mathbf{in}\ C[M_1]$$
$$C[\mathbf{let}\ x = M_1\ \mathbf{in}\ M_2] \Longrightarrow \mathbf{let}\ x = M_1\ \mathbf{in}\ C[M_2]$$
$$C[\mathbf{case}\ x\ \mathbf{of}\ y.M_1, z.M_2] \Longrightarrow \mathbf{case}\ x\ \mathbf{of}\ y.C[M_1], z.C[M_2]$$

where $C[\ ]$ denotes any one of the following contexts:

$$C[\ ] ::= ([\ ], M) \mid (M, [\ ]) \mid \mathbf{in1}([\ ]) \mid \mathbf{in2}([\ ]) \mid \mathbf{app}\ (x\ [\ ])\ \mathbf{is}\ y\ \mathbf{in}\ M \mid \mathbf{let}\ x = [\ ]\ \mathbf{in}\ M$$

This set of rules can be regarded as a "one-step version" of some of A-reductions defined in [6]. (The other A-reduction rules corresponds to proof transformation from $\mathcal{N}$ to $\mathcal{GK}$ for function application.)

Next we consider transformations for ($\vee$:L). The structures of the transformations are similar to the previous cases except that part of derivation is copied. Suppose ($\vee$:L) appears in an argument premise of a rule R. There are two types of transformations depending on whether R is a right rule or not. Here we only show the case where where ($\vee$:L) appears in the left argument premise of ($\wedge$:R).

$$\dfrac{\dfrac{\Delta_1 \quad \Delta_2}{\Gamma, x:\tau_1 \vee \tau_2 \rhd \mathbf{case}\ x\ \mathbf{of}\ y.M_1, z.M_2 \,:\, \tau_3}\ (\vee\mathrm{:L}) \quad \Delta_3}{\Gamma, x:\tau_1 \vee \tau_2 \rhd ((\mathbf{case}\ x\ \mathbf{of}\ y.M_1, z.M_2),\ M_3) \,:\, \tau_3 \wedge \tau_4}\ (\wedge\mathrm{:R}) \Longrightarrow$$

$$\dfrac{\dfrac{\Delta_1 \quad \Delta_3 + \{y:\tau_1\}}{\Gamma, x:\tau_1 \vee \tau_2, y:\tau_1 \rhd (M_1, M_3) \,:\, \tau_3 \wedge \tau_4}\ (\wedge\mathrm{:R}) \quad \dfrac{\Delta_2 \quad \Delta_3 + \{z:\tau_2\}}{\Gamma, x:\tau_1 \vee \tau_2, z:\tau_2 \rhd (M_2, M_3) \,:\, \tau_3 \wedge \tau_4}\ (\wedge\mathrm{:R})}{\Gamma, x:\tau_1 \vee \tau_2 \rhd \mathbf{case}\ x\ \mathbf{of}\ y.(M_1, M_3), z.(M_2, M_3) \,:\, \tau_3 \wedge \tau_4}\ (\vee\mathrm{:L})$$

In this rule, the derivation Δ_3 is duplicated. The same phenomenon occurs in the transformation of a conditional statement in [6]. It is not hard to modify the rule for $(\vee : L)$ to avoid copying a part of derivation by introducing additional assumption for holding the intermediate result of the case analysis.

We write

$$\mathcal{S} \vdash \Gamma \rhd M \overset{*}{\Longrightarrow} M' \,:\, \tau$$

if the proof of $\Gamma \rhd M \,:\, \tau$ can be transformed to that of $\Gamma \rhd M' \,:\, \tau$ by repeated application of some of the transformation rules $\mathcal{S}$. Since each rule in $\mathcal{S}$ is a valid proof transformation, it is immediate that if $\mathcal{GK} \vdash \Gamma \rhd M \,:\, \tau$ and $\mathcal{S} \vdash \Gamma \rhd M \overset{*}{\Longrightarrow} M' \,:\, \tau$ then $\mathcal{GK} \vdash \Gamma \rhd M' \,:\, \tau$. Moreover, we have the following.

Theorem 1. *If* $\mathcal{GK} \vdash \Gamma \rhd M : \tau$ *then there is some* M' *such that* $\mathcal{S} \vdash \Gamma \rhd M \stackrel{*}{\Longrightarrow} M' : \tau$ *and* $\mathcal{GKA} \vdash \Gamma \rhd M' : \tau$

This is proved by a routine induction on derivation of M. By these results, A-normal forms can be regarded as a form of normal proofs in $\mathcal{GK}$ identified by the subsystem $\mathcal{GKA}$.

This transformation is the first half of the proof normalization that corresponds to the computation of a program in a conventional implementation. A distinguishing property of this normalization process is that cuts are moved downwards in the compound proofs of products and sums, which is the opposite of the usual cut elimination procedure. As we shall show later, program execution does not corresponds to cut elimination either.

Our choice of the contraction-free variant of $\mathcal{GK}$ is suitable for the normalization transformation. If we adopted the original Gentzen's sequent calculus, then additional machinery would have been needed. To see the difficulty, consider the term (**proj** z **on** (x, y) **in** $(x, y), z)$. This is provable in the Gentzen's sequent calculus, but the corresponding proof of A-normal form **proj** z **on** (x, y) **in** $((x, y), z)$ is not directly provable in the subsystem corresponding to the Gentzen's sequent calculus.

4 A-Normal Compilation as Proof Transformation

Our first main result is that the compilation from the set of typed lambda terms into the set of A-normal forms is characterized as the combination of a proof transformation from $\mathcal{N}$ to $\mathcal{GK}$ and a proof normalization from $\mathcal{GK}$ to $\mathcal{GKA}$.

We first state the well known result in proof theory.

Theorem 2 ([8, 18, 21, 17]). *There is an algorithm, denoted here by* $\mathcal{NG}$*, that transforms any* $\mathcal{N}$ *proof to a* $\mathcal{GK}$ *proof.*

The main idea behind $\mathcal{NG}$ is to decompose an elimination rule into the combination of a left rule and a cut rule. Since this result will be used in the following development, we include some important cases of the algorithm $\mathcal{NG}$ in Fig. 5.

By combining Theorem 1 and Theorem 2, we have the following.

Corollary 3. *Every proof in* $\mathcal{N}$ *is transformed to a proof in* $\mathcal{GKA}$*.*

Moreover, compiling a lambda term to an A-normal form is exactly this transformation, which we prove below.

Flanagan et. al. [6] have given a linear time compilation algorithm from lambda terms to A-normal forms in Scheme using the two-level programming technique for CPS algorithms by Danvy and Fillinski [5]. To establish the desired result, it is essential to reason about the meta-level language as well. For this purpose, we re-state their algorithm using a simply typed first-order language for manipulating sequent proofs. To define the language, we extend the proof system with *proof variables* (ranged over by X) typed with a logical sequent $\Gamma \rhd \tau$. We also extend the set of terms with the same set of variables. We use σ

$$\mathcal{NG}\left(\frac{\begin{array}{cc}\Pi_1 & \Pi_2\\ (\Gamma \rhd M_1 \,:\, (\tau_1 \supset \tau_2)) & (\Gamma \rhd M_2 \,:\, \tau_1)\end{array}}{\Gamma \rhd M_1\ M_2 \,:\, \tau_2}\ \supset\!\text{:E}\right)$$

$$= \frac{\begin{array}{cc}\mathcal{NG}(\Pi_1) & \mathcal{NG}(\Pi_2) + \{x : \tau_1 \supset \tau_2\}\\ (\Gamma \rhd M_1' \,:\, \tau_1 \supset \tau_2) & \dfrac{(\Gamma, x : \tau_1 \supset \tau_2 \rhd M_2' \,:\, \tau_1) \quad \overline{\Gamma, x : \tau_1 \supset \tau_2, y : \tau_2 \rhd y \,:\, \tau_2}\ (\text{taut})}{\Gamma, x : \tau_1 \supset \tau_2 \rhd \mathbf{app}\ (x\ M_2')\ \mathbf{is}\ y\ \mathbf{in}\ y \,:\, \tau_2}\ (\supset\!\text{:L})\end{array}}{\Gamma \rhd \mathbf{let}\ x = M_1'\ \mathbf{in}\ \mathbf{app}\ (x\ M_2')\ \mathbf{is}\ y\ \mathbf{in}\ y \,:\, \tau_2}\ (\text{cut})$$

$$\mathcal{NG}\left(\frac{\begin{array}{c}\Pi_1\\ (\Gamma \rhd M \,:\, \tau_1 \wedge \tau_2)\end{array}}{\Gamma \rhd M.i \,:\, \tau_i}\ \wedge\!\text{:E}i\right)$$

$$= \frac{\begin{array}{cc}\mathcal{NG}(\Pi_1) & \overline{\Gamma, y : \tau_1 \wedge \tau_2, x_1 : \tau_1, x_2 : \tau_2 \rhd x_i \,:\, \tau_i}\ (\text{taut})\\ (\Gamma \rhd M' \,:\, \tau_1 \wedge \tau_2) & \dfrac{\Gamma, y : \tau_1 \wedge \tau_2 \rhd \mathbf{proj}\ y\ \mathbf{on}\ (x_1, x_2)\ \mathbf{in}\ x_i \,:\, \tau_i}{}\ (\wedge\!\text{:L})\ (\text{cut})\end{array}}{\Gamma \rhd \mathbf{let}\ y = M'\ \mathbf{in}\ \mathbf{proj}\ y\ \mathbf{on}\ (x_1, x_2)\ \mathbf{in}\ x_i \,:\, \tau_i} \quad (i \in \{1, 2\})$$

$$\mathcal{NG}\left(\frac{\begin{array}{ccc}\Pi_1 & \Pi_2 & \Pi_3\\ (\Gamma \rhd M_1 \,:\, \tau_1 \vee \tau_2) & (\Gamma, x_1 : \tau_1 \rhd M_2 \,:\, \tau_2) & (\Gamma, x_2 : \tau_2 \rhd M_3 \,:\, \tau_2)\end{array}}{\Gamma \rhd \mathbf{case}\ M_1\ \mathbf{of}\ x_1.M_1,\ x_2.M_2 \,:\, \tau_2}\ (\vee\!\text{:E})\right)$$

$$= \frac{\begin{array}{cc}\mathcal{NG}(\Pi_1) & \begin{array}{cc}\mathcal{NG}(\Pi_2) + \{y : \tau_1 \vee \tau_2\} & \mathcal{NG}(\Pi_3) + \{y : \tau_1 \vee \tau_2\}\end{array}\\ (\Gamma \rhd M_1' \,:\, \tau_1 \vee \tau_2) & \dfrac{\begin{array}{cc}(\Gamma, y : \tau_1 \vee \tau_2, x_1 : \tau_1 \rhd M_2' \,:\, \tau_2) & (\Gamma, y : \tau_1 \vee \tau_2, x_2 : \tau_2 \rhd M_3' \,:\, \tau_3)\end{array}}{\Gamma, y : \tau_1 \vee \tau_2 \rhd \mathbf{case}\ y\ \mathbf{of}\ x_1.M_2', x_2.M_3' \,:\, \tau_3}\ (\vee\!\text{:L})\ (\text{cut})\end{array}}{\Gamma \rhd \mathbf{let}\ y = M_1'\ \mathbf{in}\ \mathbf{case}\ y\ \mathbf{of}\ x_1.M_2', x_2.M_3' \,:\, \tau_3}\ (\text{cut})$$

Fig. 5. Some of Proof Translation Rules form $\mathcal{N}$ to $\mathcal{GK}$

as a meta variable ranging over logical sequent $\Gamma \rhd \tau$ regarded as a type. Let Ω be a set of type assignment for proof variables, which is a mapping from a finite set of proof variables to types (logical sequents), and write $\{X_1 : \sigma, \ldots, X_n : \sigma_n\}$ for a type assignment that assigns σ_i to X_i. Let $\mathcal{GK}(\Omega)$ be the proof system obtained from $\mathcal{GK}$ by adding $\Gamma \rhd X \,:\, \tau$ as an axiom for each $X : \Gamma \rhd \tau$ in Ω, and also by adding the set of variables appearing in Ω as new term variables. We write $\mathcal{GK}(\Omega) \vdash \Gamma \rhd M \,:\, \tau$ if $\Gamma \rhd M \,:\, \tau$ is provable in $\mathcal{GK}(\Omega)$. If Δ_1 is a proof containing an axiom for X of type σ and Δ_2 is a proof of type σ then we write $[\Delta_2/X]\Delta_1$ for the proof obtained from Δ_1 by replacing each occurrence of axiom for X with Δ_2 and the variable occurrences of X in the terms of Δ_1 by M_2. The following substitution property holds.

Proposition 4. *1. If Δ_1 is a proof of σ_1 in $\mathcal{GK}(\Omega, X : \sigma_2)$ and Δ_2 is a proof of σ_2 in $\mathcal{GK}(\Omega)$ then $[\Delta_2/X]\Delta_1$ is a proof of σ_1 in $\mathcal{GK}(\Omega)$.*

2. If $\mathcal{GK}(\Omega, X : (\Gamma_2 \rhd \tau_2)) \vdash \Gamma_1 \rhd M_1 \,:\, \tau_1$ and $\mathcal{GK}(\Omega) \vdash \Gamma_2 \rhd M_2 \,:\, \tau_2$ then $\mathcal{GK}(\Omega) \vdash \Gamma_1 \rhd [M_2/X]M_1 \,:\, \tau_1$.

The set of typings of the first-order language (whose terms are ranged over by D) is defined by the following rules to derive a typing of the form $\Omega \vdash D : \sigma$ denoting the fact that D is a well typed term under Ω.

– $\Omega \vdash D : (\Gamma \rhd \tau)$ if $\mathcal{GK}(\Omega) \vdash \Gamma \rhd D \,:\, \tau$.

– $\Omega \vdash \delta X : \sigma_1.D : \sigma_1 \to \sigma$ if $\Omega, X : \sigma_1 \vdash D : \sigma_2$.
– $\Omega \vdash D_1 \odot D_2 : \sigma$ if $\Omega \vdash D_1 : \sigma_1 \to \sigma$ and $\Omega \vdash D_2 : \sigma_1$.

The reduction relation on this language is defined by the following rule

$$(\delta X : \sigma_1.D_1) \odot D_2 \longrightarrow [D_2/X]D_1$$

In the following we omit the type annotation in $\delta X : \sigma_1.D$ if it does not cause any confusion.

It is easily seen that the reduction is confluent and terminating. We regard terms of this language as those modulo the equality induced by this reduction relation. Also, X in $\delta X.D$ is a bound variable, and we regard terms module bound variable renaming. From this and Proposition 4, the following properties are immediate.

Proposition 5. *1. If $\Omega \vdash D : \sigma$ then D determines a proof of σ in $\mathcal{GK}(\Omega)$.*
2. If $\Omega \vdash D : \sigma_1 \to \sigma_2$ then D is a term of the form $\delta X : \sigma_1.D'$ such that $\Omega, X : \sigma_1 \vdash D' : \sigma_2$.

We can therefore regard a typed term D such that $\Omega \vdash D : (\Gamma \rhd \tau_1) \to (\Gamma' \rhd \tau_2)$ as a "context," i.e. a proof of $\Gamma' \rhd \tau_2$ in $\mathcal{GK}(\Omega)$ containing a "hole" to be filled with a proof of $\Gamma \rhd \tau_1$ in $\mathcal{GK}(\Omega)$.

Suppose $\Omega \vdash D : \sigma$. We write $\mathcal{S}(\Omega) \vdash D \stackrel{*}{\Longrightarrow} D' : \sigma$ if the proof determined by D is reduced to the one determined by D' in $\mathcal{GK}(\Omega)$ using the set $\mathcal{S}$ of proof reduction rules defined earlier. Suppose $\Omega \vdash D : (\Gamma \rhd \tau_1) \to (\Gamma' \rhd \tau_2)$. We also write $\mathcal{S}(\Omega) \vdash D \stackrel{*}{\Longrightarrow} D' : \sigma_1 \to \sigma_2$ if $D = \delta X : \sigma_1.D_0$, $D' = \delta X : \sigma_1.D_1$ and $\mathcal{S}(\Omega, X : \sigma_1) \vdash D_0 \stackrel{*}{\Longrightarrow} D_1 : \sigma_2$.

The following two lemmas can then be shown by the properties of proofs in $\mathcal{GK}$ using Proposition 5.

Lemma 6. *If $\Omega \vdash D : (\Gamma_1 \rhd \tau_1) \to (\Gamma_2 \rhd \tau_2)$, $x \notin dom(\Gamma_1) \cup dom(\Gamma_2)$ and $\Omega \subseteq \Omega'$ then $\Omega' \vdash D : (\Gamma_1, x : \tau_3 \rhd \tau_1) \to (\Gamma_2, x : \tau_3 \rhd \tau_2)$.*

Lemma 7. *If $\mathcal{S}(\Omega) \vdash D_1 \stackrel{*}{\Longrightarrow} D_1' : \sigma \to \sigma'$ and $\Omega \vdash D_2 \stackrel{*}{\Longrightarrow} D_2' : \sigma$ then $\mathcal{S}(\Omega) \vdash D_1 \odot D_2 \stackrel{*}{\Longrightarrow} D_1' \odot D_2' : \sigma'$.*

Using this first-order language, A-normal translation algorithm is given as a function $[\![\_]\!]_\_$ that takes a terms D such that $\Omega \vdash D : \Gamma_1 \rhd \tau_1$ and a function term k such that $\Omega \vdash k : (\Gamma_1 \rhd \tau_1) \to (\Gamma_2 \rhd \tau_2)$, and return a term D' such that $\Omega \vdash D' : \Gamma_2 \rhd \tau_2$. For the notational reason, we give the algorithm as an algorithm to transformation untyped terms in Fig. 6. (Note that the first-order language does not contain variables of function type; k used in this definition is a meta variable denoting a term of the form $\delta X.D$.) It is straightforward to construct the complete algorithm from this description. This algorithm, when regarded as one on untyped lambda terms, is a generalization of the A-normalization algorithm given in [6].

Under these preparations, we can now establish the following desired result.

$$
\begin{aligned}
[\![c^b]\!]k &= k \odot c^b \\
[\![x]\!]k &= k \odot x \\
[\![\lambda x.M]\!]k &= k \odot (\lambda x.[\![M]\!](\delta X.X)) \\
[\![(M\ N)]\!]k &= [\![M]\!](\delta X.[\![N]\!](\delta Y.\mathbf{let}\ x = X\ \mathbf{in\ app}\ (x\ Y)\ \mathbf{is}\ z\ \mathbf{in}\ k \odot z)) \\
[\![(M,N)]\!]k &= [\![M]\!](\delta X.[\![N]\!](\delta Y.k \odot (X,Y))) \\
[\![M.i]\!]k &= [\![M]\!](\delta X.\mathbf{let}\ x = X\ \mathbf{in\ proj}\ x\ \mathbf{on}\ (x_1,x_2)\ \mathbf{in}\ k \odot x_i) \\
[\![\mathbf{in}i(M)]\!]k &= [\![M]\!](\delta X.k \odot \mathbf{in}i(X)) \\
[\![case\ M\ \lambda x.N, \lambda y.L]\!]k &= [\![M]\!](\delta X.(\mathbf{let}\ z = X\ \mathbf{in\ case}\ z\ \mathbf{of}\ x.[\![N]\!]k, y.[\![L]\!]k))
\end{aligned}
$$

Fig. 6. A-normal compilation algorithm $[\![\_]\!]\_$

Theorem 8. *If* $\mathcal{N} \vdash \Gamma \rhd M : \tau_1$ *and* $\Omega \vdash k : (\Gamma' \rhd \tau_1) \to (\Gamma \rhd \tau_2)$ *such that* $\Gamma \subseteq \Gamma'$ *then* $\Omega \vdash [\![M]\!]k : \Gamma \rhd \tau_2$ *and* $\mathcal{S}(\Omega) \vdash k \odot \mathcal{NG}(M) \stackrel{*}{\Longrightarrow} [\![M]\!]k : (\Gamma \rhd \tau_2)$.

As a special case of Theorem 8 where k is $\delta X.X$, we have the following.

Corollary 9. *If* $\mathcal{N} \vdash \Gamma \rhd M : \tau$ *then* $\mathcal{GKA} \vdash \Gamma \rhd [\![M]\!]\delta X.X : \tau$ *and* $\mathcal{S} \vdash \Gamma \rhd \mathcal{NG}(M) \stackrel{*}{\Longrightarrow} [\![M]\!]\delta X.X : \tau$.

This establishes that A-normal compilation corresponds to proof transformation.

5 Program Execution as Proof Reduction

We move to the second half of our Curry-Howard isomorphism and establish that the execution of the compiled program by an abstract machine corresponds to proof reduction process in $\mathcal{GKA}$. For the set of A-normal forms, Flanagan et.al. [6] have defined an abstract machine called C_aEK. Here we define an equivalent operational semantics in the style of natural semantics [12], which makes the correspondence to logic more evident. The set of *runtime values* (ranged over by r) is given by the following syntax:

$$r ::= c^b \mid cls(E, \lambda x.M) \mid (r, r) \mid \mathbf{in1}(r) \mid \mathbf{in2}(r)$$

$cls(E, \lambda x.M)$ is a *closure* representing a function, where E is a *runtime environment* which is a mapping form a finite set of variables to runtime values. Fig. 7 define the operational semantic as a set of rules to derive the relation $E \vdash M \Downarrow r$ indicating the fact that M is evaluated to r under E.

We show that this operational semantics corresponds to proof reduction in $\mathcal{GKA}$. We first define a restriction of $\mathcal{GKA}$ in Fig. 8 that corresponds to a set of runtime values defined above. In this system, a judgment of the form $\Gamma \rhd_v M : \tau$ corresponds to a runtime value, and one of the form $\Gamma \rhd_\lambda M : \tau$ is an auxiliary judgment used to derive a closure. If $\Gamma \rhd_v M : \tau$ is derivable, then one of M is of the form

$$\mathbf{let}\ x_1 = M_1\ \mathbf{in}\ \ldots \mathbf{let}\ x_n = M_n\ \mathbf{in}\ \lambda x.M$$

Computation Rules:

$$\frac{E(x) = cls(E_1, \lambda z.M_1) \quad \gamma(E,V) = r_1 \quad E_1, z : r_1 \vdash M_1 \Downarrow r_2 \quad E, y : r_2 \vdash M \Downarrow r}{E \vdash \mathbf{app}\ (x\ V)\ \mathbf{is}\ y\ \mathbf{in}\ M \Downarrow r}$$

$$\frac{E(x) = (r_1, r_2) \quad E, y : r_1, z : r_2 \vdash M \Downarrow r}{E \vdash \mathbf{proj}\ x\ \mathbf{on}\ (y,z)\ \mathbf{in}\ M \Downarrow r} \qquad \frac{E(x) = \mathbf{in1}(r_1) \quad E, y : r_1 \vdash M_1 \Downarrow r}{E \vdash \mathbf{case}\ x\ \mathbf{of}\ y.M_1, z.M_2 \Downarrow r}$$

$$\frac{E(x) = \mathbf{in2}(r_1) \quad E, z : r_1 \vdash M_2 \Downarrow r}{E \vdash \mathbf{case}\ x\ \mathbf{of}\ y.M_1, z.M_2 \Downarrow r} \qquad \frac{\gamma(E,V) = r_1 \quad E, x : r_1 \vdash M \Downarrow r}{E \vdash \mathbf{let}\ x = V\ \mathbf{in}\ M \Downarrow r}$$

Value Construction rules:

$$\gamma(E, c^b) = c^b \quad \gamma(E,x) = E(x) \quad \gamma(E, \lambda x.M) = cls(E, \lambda x.M)$$
$$\gamma(E,(V_1,V_2)) = (\gamma(E,V_1), \gamma(E,V_2)) \quad \gamma(E, \mathbf{in}i(V)) = \mathbf{in}i(\gamma(E,V))$$

Fig. 7. Operational semantics for A-normal forms

Lambda Derivations

$$(\supset\text{:R})\ \frac{\Gamma, x : \tau_1 \rhd M\ :\ \tau_1}{\Gamma \rhd_\lambda \lambda x.M\ :\ \tau_1 \supset \tau_2} \qquad (\text{cut})\ \frac{\Gamma \rhd_v M_1\ :\ \tau_1 \quad \Gamma, x : \tau_1 \rhd_\lambda M_2\ :\ \tau_2}{\Gamma \rhd_\lambda \mathbf{let}\ x = M_1\ \mathbf{in}\ r_2\ :\ \tau_2}$$

Value derivations

$$(\text{axiom})\ \ \Gamma \rhd_v c^b\ :\ b \qquad (\text{closure})\ \frac{\Gamma \rhd_\lambda M\ :\ \tau \quad FV(M) = \emptyset}{\Gamma \rhd_v M\ :\ \tau}$$

$$(\wedge\text{:R})\ \frac{\Gamma \rhd_v M_1\ :\ \tau_1 \quad \Gamma \rhd_v M_2\ :\ \tau_2}{\Gamma \rhd_v (M_1, M_2)\ :\ \tau_1 \wedge \tau_2} \qquad (\vee\text{:Ri})\ \frac{\Gamma \rhd_v M\ :\ \tau_i}{\Gamma \rhd_v \mathbf{in}i(M)\ :\ \tau_1 \vee \tau_2}$$

Fig. 8. Runtime Value Derivations

By the definition of the restricted proof system, each M_i is closed, and the order of the cuts is irrelevant. We can therefore consider the series of cuts as a mapping from variables to closed terms of the form $\{x_1 = M_1, \ldots, x_n = M_n\}$ and consider the term modulo the equivalence induced by reordering of cuts and write $\mathbf{let}\ \{x_1 = M_1, \ldots, x_n = M_n\}\ \mathbf{in}\ \lambda x.M$. Let E be a mapping from variables to closed terms. We write $E : \Gamma$ if $dom(E) = dom(\Gamma)$ and for each $x \in dom(E)$, $\emptyset \rhd_v E(x)\ :\ \Gamma(x)$ is provable. If $E : \Gamma$, then the sequence of cuts corresponding to E is abbreviated as follows.

$$\frac{E : \Gamma_1 \quad \Gamma_2; \Gamma_1 \rhd M\ :\ \tau_2}{\Gamma_2 \rhd \mathbf{let}\ E\ \mathbf{in}\ M\ :\ \tau_2}\ \text{cut}^*$$

Under this interpretation, if $\Gamma \rhd_v M\ :\ \tau$ is provable by the proof rules in Fig. 8 then M is isomorphic to some runtime value r defined above, and therefore the typing rules can be regarded as a type system of runtime values. In what follows, we identify runtime values with the corresponding terms and write $\Gamma \rhd_v r\ :\ \tau$ if a term corresponding to r is derivable.

Our plan now is to interpret the evaluation relation $E \vdash M \Downarrow r$ as a proof reduction that transforms the proof represented by $\mathbf{let}\ E\ \mathbf{in}\ M$ to the one represented by r. The second major result of this paper is to establish that this is indeed the case, as shown in the following.

Theorem 10. *There is an algorithm taking a proof of* $\emptyset \rhd \mathbf{let}\ E\ \mathbf{in}\ M\ :\ \tau$*, producing a runtime value* r *and a proof of* $\emptyset \rhd r\ :\ \tau$*.*

Proof (Outline). This is proved by defining a proof reduction algorithm, denoted as $\emptyset \rhd \mathbf{let}\ E\ \mathbf{in}\ M \Downarrow r\ :\ \tau$, and showing its correctness. Due to the space limitation, we can only explain the main idea behind the proof.

The proof reduction algorithm first transforms a proof (represented by a term) of the form $\mathbf{let}\ E\ \mathbf{in}\ M$ to a proof of the form $\mathbf{let}\ E'\ \mathbf{in}\ V$. This is done by inductively applying the algorithm to each argument proof to obtain a runtime value proof, and converting left rules in M to cuts of those runtime value proofs. The algorithm then converts $\mathbf{let}\ E'\ \mathbf{in}\ V$ to a runtime value proof. The correctness of the algorithm is shown using the idea of logical relation and reducibility [20]. We first define a family of predicates $P(\tau)$ indexed by types. $r \in P(\tau)$ if one of the following holds.

- if $\tau \equiv b$ then $r \equiv c^b$.
- if $\tau \equiv \tau_1 \supset \tau_2$ then $r \equiv \mathbf{let}\ E\ \mathbf{in}\ \lambda x.M$ such that $\forall r_1 \in P(\tau_1).\exists r_2.\emptyset \rhd \mathbf{let}\ E\{x : r_1\}M\ \mathbf{in}\ \Downarrow r_2\ :\ \tau_2$ and $r_2 \in P(\tau_2)$.
- if $\tau \equiv \tau_1 \wedge \tau_2$ then $r \equiv (r_1, r_2)$ such that $r_1 \in P(\tau_1)$ and $r_2 \in P(\tau_2)$.
- if $\tau \equiv \tau_1 \vee \tau_2$ then either $r \equiv \mathbf{in1}(r_1)$ such that $r_1 \in P(\tau_1)$, or $\mathbf{in2}(r_1)$ such that $r_1 \in P(\tau_2)$.

We then show the following property

> if $\Gamma \rhd_v M\ :\ \tau$ and for each $x \in dom(E)$, $E(x) \in P(\Gamma(x))$ then $\emptyset \rhd \mathbf{let}\ E\ \mathbf{in}\ M \Downarrow r\ :\ \tau$ for some r such that $r \in P(\tau)$

by induction on the derivation of M. □

This proof reduction algorithm, when projected on untyped terms, is the the operational semantics for A-normal forms given in Fig. 7. We have:

$$\emptyset \rhd \mathbf{let}\ E\ \mathbf{in}\ M \Downarrow r\ :\ \tau$$
$$\iff \text{there is some } \Gamma \text{ such that } E : \Gamma,\ \Gamma \rhd M\ :\ \tau, \text{ and } E \vdash M \Downarrow r$$

A distinguishing characteristic of the algorithm is that it is not based on the usual cut elimination procedure. Instead of inductively eliminating cuts, it converts left rules and cut rules to those cuts whose proofs correspond to runtime values and keeps them until the final result is obtained. This process reveals the correspondence: cut rule corresponds to building (extending) a runtime environment, and left rule corresponds to computation on a data constructor.

6 Conclusions

We have developed a logical foundation for compilation and program execution by showing that compilation of lambda terms to A-normal forms corresponds to a proof transformation from the natural deduction system to a Gentzen-style

sequent calculus followed by a proof normalization in the sequent calculus, and that evaluation of an A-normal form corresponds to a special proof reduction process in the sequent calculus. These results extend *Curry-Howard* isomorphism to compilation and program execution. There are a number of topics that merit further investigation. An interesting topic is to extend the formalism to second-order logic. Such extension wold provide a logical basis for type in compilation paradigm where a second-order type system is used to optimize programs. A-normal forms also appear to be related to various other computational interpretation of lambda calculi. In particular, it would be beneficial to compare the logical correspondence we have worked out with Moggi's [15] computational lambda calculus and Kobayashi's work on modal logic [14].

References

1. S. Abramsky. Computational interpretation of linear logic. *Theoretical Computer Science*, 3(57), 1993.
2. B. Breazu-Tannen, D. Kesner, and L. Puel. A typed pattern calculus. In *Proc. IEEE Symposium on Logic in Computer Science*, pages 262–274, 1993.
3. H. B. Curry and R. Feys. *Combinatory Logic*, volume 1. North-Holland, Amsterdam, 1968.
4. V. Danos, J-B. Jointe, and H. Schellinx. A new deconstructive logic: linear logic. *Journal of Symbolic Logic*, 63(3):755–807, 1997.
5. O. Danvy. Back to direct style. In *Proc. European Symposium on Programming*, volume 582 of *Lecture Notes in Computer Science*, pages 130–150, 1992.
6. C. Flanagan, A. Sabry, B.F. Duba, and M. Felleisen. The essence of compiling with continuation. In *Proc. ACM PLDI Conference*, pages 237–247, 1993.
7. J. Gallier. Constructive logics part I: A tutorial on proof systems and typed λ-calculi. *Theoretical Computer Science*, 110:249–339, 1993.
8. G. Gentzen. Investigation into logical deduction. In M.E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*. North-Holland, 1969.
9. T. Griffin. A formulae-as-types notion of control. In *Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages*, pages 47–58, 1990.
10. H. Herbelin. A λ-calculus structure isomorphic to Gentzen-style sequent calculus structure. In *Proc. European Association for Computer Science Logic*, Lecture Notes in Computer Science 933, pages 61–74, 1994.
11. W. Howard. The formulae-as-types notion of construction. In *To H. B. Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism*, pages 476–490. Academic Press, 1980.
12. G. Kahn. Natural semantics. In *Proc. Symposium on Theoretical Aspects of Computer Science*, pages 22–39. Springer Verlag, 1987.
13. S. Kleene. *Introduction to Metamathematics*. North-Holland, 1952. 7th edition.
14. S. Kobayashi. Monads as modality. *Theoretical Computer Science*, 175(1):29–74, 1997.
15. E. Moggi. Computational lambda-calculus and monads. In *Proceedings of the Symposium on Logic in Computer Science*, 1989.
16. I. Ogata. Cut elimination for classical proofs as continuation passing style computation. In *Proc. ASIAN Computing Science Conference, LNCS 1538*, 1998.

17. G. Pottinger. Normalization as a homomorphic image of cut-elimination. *Ann. Math. Logic*, 12:323–357, 1977.
18. D. Prawitz. *Natural Deduction.* Almqvist & Wiksell, 1965.
19. A. Sabry and M. Felleisen. Reasoning about programs in continuation-passing style. *J. Lisp and Symbolic Computation*, 6(3):287–358, 1993.
20. W. Tait. Intensional interpretations of functionals of finite type i. *Journal of Symbolic Logic*, 32(2), 1966.
21. J. Zucker. The correspondence between cut-elimination and normalization. *Ann. Math. Logic*, 7:1–112, 1974.