An implementation model of the typed λ-calculus based on Linear Chemical Abstract Machine

Shinya Sato[1], Toru Sugimoto[2], and Shinichi Yamada[1]

[1] Department of Information Sciences, Science University of Tokyo,
2641 Yamazaki, Noda, Chiba 278-8510, JAPAN
shinya@sy.is.noda.sut.ac.jp, yamada@is.noda.sut.ac.jp
[2] RIKEN Brain Science Institute
2-1 Hirosawa, Wako, Saitama 351-0198, JAPAN
sugimoto@brain.riken.go.jp

Abstract. Abramsky's Linear Chemical Abstract Machine is a term calculus which corresponds to Linear Logic, via the Curry-Howard isomorphism. We show that the typed λ-calculus is embedded into Linear Chemical Abstract Machine by Girard's embedding of Intuitionistic Logic into Linear Logic. Then we extend our result to a simple functional programming language obtained from the typed λ-calculus by adding constants from PCF. We show that the call-by-value evaluation of terms of ground types (Booleans and Natural numbers) are preserved and reflected by this translation. Finally, we discuss an operational perspective of our result. We give a sequential execution model of Linear CHAM based on Abramsky's idea of a stack of coequations and a name queue, and then we consider a concurrent multi-thread implementation of the model.

1 Introduction

Girard's Classical Linear Logic (CLL) [6] is expected to give new theoretical foundations of parallel computation [7]. Abramsky [1] gave a computational interpretation of CLL using the framework of Berry and Boudol's Chemical Abstract Machine (CHAM) [3]. In this computational system which is called Linear Chemical Abstract Machine (Linear CHAM), proof expressions are assigned to proof trees in CLL, and they are reduced by rewriting rules which correspond to cut elimination procedures of CLL. Because all rewritings are local, any number of rewritings can be performed in parallel. Though he said that it offers the prospect of typed concurrent functional programming in which correctness is guaranteed by the typing, much work in this direction remains to be done. For this purpose, we investigate the relationship between Linear CHAM and the typed λ-calculus which is widely acknowledged to be the theoretical foundation of functional programming languages.

The typed λ-calculus is strongly connected to Intuitionistic Logic (IL) by the Curry-Howard Isomorphism. By using Girard's translation of proofs in IL into proofs in CLL, we give a translation of typable λ-terms into proof expressions. We show that for every reduction sequence in the call-by-value evaluation in the λ-calculus there is a corresponding reduction sequence in Linear CHAM. Since proof expressions in a normal form correspond to λ-terms in a normal form, Linear CHAM computes the same answer as the call-by-value evaluation in the λ-calculus.

Then, we extend our result to a simple functional programming language obtained from the typed λ-calculus by adding constants (Booleans, Natural numbers, arithmetic functions, conditional and recursion) from PCF [12]. One of the main contributions of this paper is that we extend proof expressions, Linear CHAM and Girard's translation so that the call-by-value evaluation of terms of ground types (Booleans and Natural numbers) is preserved and reflected. In this case, Linear CHAM computes the same answer as the evaluation of functional programs, and it terminates if and only if a given program terminates.

Finally, we discuss an operational perspective of our result. We give a sequential execution model of Linear CHAM based on Abramsky's idea of a stack of coequations and a name queue, and then we consider a concurrent multi-thread implementation of the model. Because Linear CHAM is considered as some sort of intermediate language into which typed functional programming languages are compiled, we can expect that Linear CHAM can be used as a foundation of concurrent functional languages with types.

Overview. The rest of this paper is structured as follows. In the next section we review the typed λ-calculus. Here we introduce a slightly modified version of the typed λ-calculus, the typed λ'-calculus in order to simplify our definitions of the translation. In Section 3 we review Linear CHAM, proof expressions and reaction rules. In Section 4 we investigate the relationship between the λ'-calculus and Linear CHAM. We give a translation from typable λ'-terms into proof expressions by using Girard's translation of proofs in IL into proofs in CLL, and we show that the typed λ-calculus is embedded into Linear CHAM. In Section 5 we extend this result to a functional programming languages obtained from the typed λ'-calculus by adding constants from PCF. In Section 6 we introduce sequential and concurrent execution models of Linear CHAM and discuss an operational perspective of our result. Section 7 concludes the paper.

Related Work. Interaction net [8] is a graphical rewriting system which can be regarded as a generalisation of proof nets for Linear Logic. Mackie [9] proposed a lambda evaluator based on interaction nets and showed that his translation from λ-terms into interaction nets preserves reduction in λ-calculus. He extended the translation to handle constants from PCF, but it is also shown that in this system we need some additional strategy for reducing nets so that results similar to Theorem 9 can hold when there is the possibility of non-terminating computation. There are similarities between interaction nets and Linear CHAM. In fact, using the method described in [5], we can obtain a textual calculus of interaction nets which is surprisingly similar to Linear CHAM. One of the differences between them is that Linear CHAM has a term of the form $\bar{x}(P)$ in which P is suspended from reduction, and it is only resumed when sufficient

$$\frac{}{x : A \vdash x : A}\,(\mathrm{Id}) \qquad \frac{\bar{x} : \Gamma, x : A, y : B, \bar{y} : \Delta \vdash M : C}{\bar{x} : \Gamma, y : B, x : A, \bar{y} : \Delta \vdash M : C}\,(\mathrm{Exchange})$$

$$\frac{\bar{x} : \Gamma, x : A \vdash M : B}{\bar{x} : \Gamma \vdash \lambda x.M : A \supset B}\,(\supset -\mathrm{I_{C1}}) \qquad \frac{\bar{x} : \Gamma, x_1 : A, x_2 : A, ..., x_n : A \vdash M : B}{\bar{x} : \Gamma \vdash \lambda x_1 @ x_2 @ ... @ x_n.M : A \supset B}\,(\supset -\mathrm{I_{C}}_n)_{(n \geq 2)}$$

$$\frac{\bar{x} : \Gamma \vdash M : B}{\bar{x} : \Gamma \vdash \lambda\_.M : A \supset B}\,(\supset -\mathrm{I_W}) \qquad \frac{\bar{x} : \Gamma \vdash M : A \supset B \quad \bar{y} : \Delta \vdash N : A}{\bar{x} : \Gamma, \bar{y} : \Delta \vdash MN : B}\,(\supset -\mathrm{E})$$

Distinct variables are introduced for each instance of (Id).

Fig. 1. The type assignment system for λ'-terms

demand has been generated. In this way, reduction in Linear CHAM is controlled so that it corresponds to the call-by-value evaluation, and this is the reason why Theorem 9 holds. On the other hand, interaction nets have no such mechanism in themselves.

The linear λ-calculus [1, 2, 13] is a resource-sensitive refinement of λ-calculus and is considered as a computational interpretation of Intuitionistic Linear Logic via Curry-Howard isomorphism. Translations from the typed λ-calculus into the linear λ-calculus are given in [4, 10].

An embedding of the linear λ-calculus into Linear CHAM is given by Mikami and Akama [11]. Composing Girard's translation from λ-terms into linear λ-terms and the translation given in [11], we can obtain another translation from λ-terms into proof expressions. For every λ-term M, this translation and our translation $(-)^\circ$ take the same value up to the equivalence relation $\cong$.

2 The typed λ-calculus

We use in this paper a slightly modified version of the typed λ-calculus, which we call the *typed λ'-calculus* in order to simplify our definition of the translation into Linear CHAM. As we can easily see, every typable λ-term is representable as a λ'-term and vice versa.

Definition 1 **(λ'-terms).**

- Patterns *have one of the forms:* $\quad x \quad \_ \quad x_1 @ x_2 @ ... @ x_n$
  where $x, x_1, ..., x_n (n \geq 2)$ are variables. We use p to range over patterns.
- λ'-terms *have one of the forms:* $\quad x \quad \lambda p.M \quad MN$
  where x is a variable, p is a pattern, and M, N are λ'-terms. We use M, N to range over λ'-terms.

In Figure 1, we give the type assignment system for λ'-terms; type assignments have the form of sequents $x_1 : A_1, ..., x_n : A_n \vdash M : B$ where $x_1, ..., x_n$ are variables and $A_1, ..., A_n, B$ are types. We use $\bar{x}, \bar{y}, ...$ to range over sequences of variables and $\Gamma, \Delta, ...$ to range over sequences of types. We sometimes write $\bar{x} : \Gamma$ for $x_1 : A_1, ..., x_n : A_n$ where $\bar{x} = x_1, ..., x_n$ and $\Gamma = A_1, ..., A_n$. For the rest of this paper, we assume that all λ'-terms mentioned are typable.

Definition 2. *The set of free variables, denoted* $\mathrm{FV}(M)$*, of a* λ'*-term* M *is defined inductively as follows:*

$$\mathrm{FV}(x) = \{x\} \qquad \mathrm{FV}(\lambda x.M) = \mathrm{FV}(M) - \{x\}$$
$$\mathrm{FV}(\lambda\_.M) = \mathrm{FV}(M) \qquad \mathrm{FV}(\lambda x_1@...@x_n.M) = \mathrm{FV}(M) - \{x_1, ..., x_n\}$$
$$\mathrm{FV}(MN) = \mathrm{FV}(M) \cup \mathrm{FV}(N)$$

M *is* closed *if* $\mathrm{FV}(M) = \emptyset$*. A closed* λ'*-term is called* a value *if it has the form* $\lambda p.N$*. We use* V *to range over values.*

We define one-step reduction of λ'-terms in the call-by-value evaluation using the notion of *context*.

Definition 3 (reduction in the call-by-value evaluation).

- Context $C[\,]$ *is defined by:* $\quad C[\,] ::= [\,] \mid VC[\,] \mid C[\,]M.$
  We call an atomic element of context a hole. *We write* $C[M]$ *to mean the result of substituting* M *for the hole in* $C[\,]$ *.*
- *We write*
  - $M[N/x]$ = *the result of substituting* N *for (the free variable)* x *in* M*;*
  - $M[N/\_] = M$*;*
  - $M[N/x_1@...@x_n]$ = *the result of substituting* $N^1, ..., N^n$ *for* $x_1, ..., x_n$ *in* M

  where $N^1, ..., N^n$ *are the results of replacing each variable in* N *by a fresh variable.*
- *We define* reduction in the call-by-value evaluation *by:* $\quad C[(\lambda p.M)V] \overset{\beta}{\Longrightarrow} C[M[V/p]]$.

Definition 4. *We define* $M \Downarrow N$ *by:* $M \overset{\beta}{\Longrightarrow}{}^{*} N$*, where* N *is a* $\overset{\beta}{\Longrightarrow}$*-normal form.*

3 Linear CHAM

3.1 Classical Linear Logic and proof expressions

Abramsky [1] introduced *proof expressions* which are assigned to proof trees in sequent calculus for Classical Linear Logic (CLL). For the purpose of this paper, we only need the part of it that corresponds to the Multiplicative Exponential fragment of CLL (MELL).

Definition 5 (proof expressions).

- *We assume a set* $\mathcal{N}$ *of* names, *ranged over by* $x, y, z, ..., x_1, x_2, ...$ *We use* $\bar{x}, \bar{y}, ...$ *to range over sequences of names.*
- Terms *have one of the forms:* $\quad x \quad t \otimes u \quad t \parr u \quad ?t \quad \_ \quad t@u \quad \bar{x}(P)$
  where t, u *are terms, and* P *is a proof expression.*
- Coequations *have the form* $t \perp u$ *where* t, u *are terms. We use* Θ, Ξ *to range over sequences of coequations.*
- Proof expressions *have the form* $\Theta; \bar{t}$ *where* Θ *is a sequence of coequations and* $\bar{t}$ *is a sequence of terms. We call* $\bar{t}$ the main body *of the proof expression* $\Theta; \bar{t}$*. We use* P, Q *to range over proof expressions.*

$$\frac{}{\vdash ; x : A^{\perp}, x : A}\ \text{Axiom} \qquad \frac{\vdash \Theta; \bar{t} : \Gamma, t : A, u : B, \Delta}{\vdash \Theta; \bar{t} : \Gamma, u : B, t : A, \Delta}\ \text{Exchange}$$

$$\frac{\vdash \Theta; \bar{t} : \Gamma, t : A \quad \vdash \Xi; \bar{s} : \Delta, u : A^{\perp}}{\vdash \Theta, \Xi, t \perp u; \bar{t} : \Gamma, \bar{s} : \Delta}\ \text{Cut}$$

$$\frac{\vdash \Theta; \bar{t} : \Gamma, t : A \quad \vdash \Xi; \bar{s} : \Delta, u : B}{\vdash \Theta, \Xi; \bar{t} : \Gamma, \bar{s} : \Delta, t \otimes u : A \otimes B}\ \text{Times} \qquad \frac{\vdash \Theta; \bar{t} : \Gamma, t : A, u : B}{\vdash \Theta; \bar{t} : \Gamma, t \parr u : A \parr B}\ \text{Par}$$

$$\frac{\vdash \Theta; \bar{t} : \Gamma, t :?A, u :?A}{\vdash \Theta; \bar{t} : \Gamma, t@u :?A}\ \text{Contraction} \qquad \frac{\vdash \Theta; \bar{t} : \Gamma}{\vdash \Theta; \bar{t} : \Gamma, \_ :?A}\ \text{Weakening}$$

$$\frac{\vdash \Theta; \bar{t} : \Gamma, t : A}{\vdash \Theta; \bar{t} : \Gamma, ?t :?A}\ \text{Dereliction} \qquad \frac{\vdash \Theta; \bar{t} :?\Gamma, t : A}{\vdash ; \bar{x} :?\Gamma, \bar{x}(\Theta; \bar{t}, t) :!A}\ \text{Ofcourse}$$

Distinct names are introduced for each instance of the Axiom, Ofcourse rules.

Fig. 2. The type assignment system for proof expressions

Proof expressions are assigned to proofs in MELL according to the rules shown in Figure 2; type assignments have the form of sequents $\vdash \Theta; t_1 : A_1, ..., t_n : A_n$ where Θ is a sequence of coequations, $t_1, ..., t_n$ are terms and $A_1, ..., A_n$ are types (formulas in MELL). We use $\Gamma, \Delta, ...$ to range over sequences of types. We sometimes write $\bar{t} : \Gamma$ for $t_1 : A_1, ..., t_n : A_n$ where $\bar{t} = t_1, ..., t_n$ and $\Gamma = A_1, ..., A_n$.

As we can see in Figure 2, coequations in a proof expression correspond to instances of Cut-rule in a proof in MELL. Coequations are rewritten by *reaction rules* of Linear CHAM, which correspond to the Cut elimination procedures in MELL.

For the rest of this paper, we assume that all proof expressions mentioned are typable.

Definition 6 (passive names and active names). *The occurrences of $x_1, ..., x_k$ in a term of the form $x_1, ..., x_k(P)$ are said to be* passive*; all other occurrences are* active*.*

3.2 Linear Chemical Abstract Machine

Linear CHAM [1] is a rewriting system for proof expressions. Rules of Linear CHAM are divided into three kinds: *structural rules* which describe the "magical mixing" [3], *reaction rules* which describe the actual computation steps and a *cleanup rule* which records the result of the computation in the main body. In Figure 3, we give structural rules, reaction rules and a cleanup rule that we use in this paper.

Definition 7 (canonical proof expressions). *A proof expression $P = \Theta; \bar{t}$ is* canonical *if it is in a $\longrightarrow$-normal form, and each coequation in Θ has the form $x \perp t$ or $t \perp x$ for some name x. We define $P \Downarrow Q$ by: $P \longrightarrow^* Q$, where Q is canonical.*

Notation:

(1) Given $\bar{x} = x_1, ..., x_k, \bar{t} = t_1, ..., t_k$, we write $\bar{x}\bot\bar{t}$ to denote the list $x_1\bot t_1, ..., x_k\bot t_k$.

(2) Given t, we write t^l, t^r to denote the results of replacing each occurrence of a name x in t by fresh names x^l and x^r, respectively; we also write P^l, P^r, and so on.

Structural Rules:

- Basic Structural Rules:
  $t\bot u \leftrightharpoons u\bot t$,
  $t\bot u, t'\bot u' \leftrightharpoons t'\bot u', t\bot u$
- Structural Context Rule:
  Basic structural rules can be applied in any context:

$$\frac{\Theta \leftrightharpoons \Xi}{C[\Theta] \leftrightharpoons C[\Xi]}$$

- Magical Mixing Rule:

$$\frac{P \leftrightharpoons^* P' \quad P' \longrightarrow Q' \quad Q' \leftrightharpoons^* Q}{P \longrightarrow Q}$$

Reaction Rules:

- Communication: $t\bot x, x\bot u \longrightarrow t\bot u$
- Read-Pair: $\bar{x}(\Theta; \bar{t}, t \parr u)\bot ?(t' \otimes u')$
  $\longrightarrow \Theta, \bar{x}\bot\bar{t}, t\bot t', u\bot u'$
- Discard: $x_1...x_k(P)\bot\_$
  $\longrightarrow x_1\bot\_, ..., x_k\bot\_$
- Copy: $\bar{x}(P)\bot u@v$
  $\longrightarrow \bar{x}\bot(\bar{x}^l@\bar{x}^r), \bar{x}(P)^l\bot u, \bar{x}(P)^r\bot v$
- Reaction Context Rule:

$$\frac{\Theta \longrightarrow \Xi}{\Theta_1, \Theta, \Theta_2; \bar{t} \longrightarrow \Theta_1, \Xi, \Theta_2; \bar{t}}$$

Cleanup Rule:

- Cleanup: $x\bot t, \Theta; \bar{t} \longrightarrow \Theta; \bar{t}[t/x]$
  (x is active in $\bar{t}$)

Fig. 3. Rules of Linear CHAM

Definition 8 (structural equivalence). *A* renaming *is a permutation $\rho : \mathcal{N} \cong \mathcal{N}$. This is extended to a substitution on terms, coequations, proof expressions etc. in the usual way. Now we define* structural equivalence *of proof expressions $P \equiv Q$ by: $\exists\rho.(\rho(P) \leftrightharpoons^* Q)$.*

This system has the following good properties [1].

Theorem 1. – **Weak Diamond Property:** *If $P \longrightarrow Q', P \longrightarrow Q''$ and $Q' \not\equiv Q''$, then for some R, $Q' \longrightarrow R$ and $Q'' \longrightarrow R$.*
- **Determinacy:** *If $P \Downarrow Q$ and $P \Downarrow R$, then $Q \equiv R$.*
- **Convergence:** *For every typable proof expression P, there is a proof expression Q such that $P \Downarrow Q$.*

4 Translation

4.1 Translation of λ'-terms into proof expressions

Formulas in IL are translated into formulas in CLL by Girard's translation [6] as follows: $A^\circ = !A$ when A is atom, $(A \supset B)^\circ = !(A^{\circ\bot} \parr B^\circ)$.

This translation is extended to the proof level, i.e., translation of a proof of $\Gamma \vdash A$ into a proof of $\vdash \Gamma^{\circ\bot}, A^\circ$. Using this idea, we define a translation $(-)^\circ$

$$\frac{}{(x \vdash x)^\circ = \;;x,x}\ (\mathrm{Id}) \qquad \frac{(\bar{x},x,y,\bar{x} \vdash M)^\circ = \ \Theta;\bar{t},t,s,\bar{s},u}{(\bar{x},y,x,\bar{x} \vdash M)^\circ = \ \Theta;\bar{t},s,t,\bar{s},u}\ (\mathrm{Exchange})$$

$$\frac{(\bar{x},x \vdash M)^\circ = \ \Theta;\bar{t},t_1,t}{(\bar{x} \vdash \lambda x.M)^\circ = \;;\bar{x'},\bar{x'}(\Theta;\bar{t},t_1 \parr t)}(\supset -\mathrm{I_{C1}})$$

$$\frac{(\bar{x},x_1,...,x_n \vdash M)^\circ = \ \Theta;\bar{t},t_1,...,t_n,t}{(\bar{x} \vdash \lambda x_1 @ ... @ x_n.M)^\circ = \;;\bar{x'},\bar{x'}(\Theta;\bar{t},(t_1 @ ... @ t_n) \parr t)}\ (\supset -\mathrm{I_{C}}_n)_{(n\geq 2)}$$

$$\frac{(\bar{x} \vdash M)^\circ = \ \Theta;\bar{t},t}{(\bar{x} \vdash \lambda\_.M)^\circ = \;;\bar{x'},\bar{x'}(\Theta;\bar{t},\_ \parr t)}\ (\supset -\mathrm{I_W})$$

$$\frac{(\bar{x} \vdash M)^\circ = \ \Theta;\bar{t},t \quad (\bar{y} \vdash N)^\circ = \ \Xi;\bar{s},s}{(\bar{x},\bar{y} \vdash MN)^\circ = \ \Theta,\Xi,t\bot?(s \otimes z);\bar{t},\bar{s},z}\ (\supset -\mathrm{E})$$

where $\bar{x'}$ and z are fresh names.

Fig. 4. Translation from λ'-terms into proof expressions

of a type assignment for a λ'-term $\bar{x} : \Gamma \vdash M : A$ into a proof expression $\Theta;\bar{t} : \Gamma^{\circ\bot}, t : A^\circ$ (Figure 4). For a closed λ'-term M, $(\vdash M)^\circ$ is uniquely determined independently of the application order of the typing rules, and we simply write M° to mean $(\vdash M)^\circ$.

4.2 An equivalence relation on proof expressions

We can translate closed λ'-terms into proof expressions using the translation $(-)^\circ$. If a closed λ'-term has a β-redex $(\lambda p.M)N$, we obtain a proof expression which has a coequation of the form $t\bot?(s \otimes z)$. Because only the Read-Pair rule can reduce this coequation, the Read-Pair rule is important when we consider the relationship between reductions in λ'-calculus and in Linear CHAM. Therefore, we distinguish the Read-Pair rule from other rules as follows:

Definition 9. – *We define $P \longrightarrow_{RP} Q$ if $P \longrightarrow Q$ can be derived using the Read-Pair rule.*
- *We define $P \longrightarrow_{rc} Q$ if $P \longrightarrow Q$ can be derived using rules other than the Read-Pair rule.*
- *We define $P \Rrightarrow Q$ if $P \longrightarrow^*_{rc} \longrightarrow_{RP} \longrightarrow^*_{rc} Q$.*

Since there seems to be no important relation between β-rule and rules other than the Read-Pair rule, we consider those proof expressions which are derived from one another using these rules equivalent. Taking other general Cut elimination procedures into account, we give the following definition of an equivalence relation on proof expressions.

Definition 10. *We define $\cong$ to be the smallest equivalence relation on proof expressions satisfying:*

(LinearCHAM) $P \ \cong \ Q$ *where* $P \longrightarrow_{rc} Q$ *or* $P \equiv Q$.

(AxiomCut) $\Theta, x \perp t; \bar{t} \cong \Theta[t/x]; \bar{t}[t/x]$

where the occurrence of x *in* Θ *or* $\bar{t}$ *is active.*

(CommutativeCut) $\Theta, x \perp \bar{x}(\Theta''; \bar{t''}, t''); \bar{t} \;\cong\; \Theta'; \bar{t'}$

where Θ' *and* $\bar{t'}$ *are the result of replacing occurrences of* $\bar{x_1} x \bar{x_2}(\Xi; \bar{t_1}, t, \bar{t_2}, u)$ *in* Θ *and* $\bar{t}$ *by* $\bar{x}\bar{x_1}\bar{x_2}(\Xi, t \perp \bar{x''}(\Theta''; \bar{t''}, t''); \bar{x''}, \bar{t_1}, \bar{t_2}, u)$.

(Inner) $\Theta; \bar{t_1}, t[P], \bar{t_2} \;\cong\; \Theta; \bar{t_1}, t[Q], \bar{t_2}, \quad \Theta, t[P] \perp s; \bar{t} \;\cong\; \Theta, t[Q] \perp s; \bar{t}$

where $P \cong Q$, *and* $t[\,]$ *is a context, which is defined as follows:*

$t[\,] ::= \bar{x}([\,]) \mid ?t[\,] \mid t \star t[\,] \mid t[\,] \star t$ $\quad$ ($\star$ *is a meta symbol denoting* $⅋, \otimes, @$)

and we write $t[P]$ *to mean the result of substituting* P *for the hole in* $t[\,]$.

4.3 Properties of the translation

First, we wish to show that for every one-step reduction in the call-by-value evaluation there is a corresponding reduction path in Linear CHAM.

Theorem 2. *If* $M \stackrel{\beta}{\Longrightarrow} N$, *then there is a proof expression* P *such that* $M^\circ \Rrightarrow P$ *and* $P \cong N^\circ$.

Next, we show that for every (n-step) reduction in the call-by-value evaluation there is a corresponding reduction path in Linear CHAM.

Theorem 3. *If* $M \stackrel{\beta}{\Longrightarrow}^{*} N$, *then there is a proof expression* P *such that* $M^\circ \Rrightarrow^{*} P$ *and* $P \cong N^\circ$.

Finally, we show that for every closed λ'-term M Linear CHAM computes the same answer as the call-by-value evaluation, up to the equivalence relation $\cong$.

Lemma 1. *Let* M *be a closed* λ'*-term which is a* $\stackrel{\beta}{\Longrightarrow}$ *-normal form. If* $M^\circ \cong P$, *then there is a proof expression* Q *such that* $P \Downarrow Q$ *and* $M^\circ \cong Q$.

Theorem 4. *If* M *is a closed* λ'*-term,* $M \Downarrow N$ *and* $M^\circ \Downarrow Q$, *then* $Q \cong N^\circ$.

4.4 An example

Here we consider a call-by-value evaluation $(\lambda x.(\lambda\_.x))(\lambda y.y) \Downarrow \lambda\_.(\lambda y.y)$. The following are translations of λ'-terms:

$$
\begin{aligned}
((\lambda x.(\lambda\_.x))(\lambda y.y))^\circ &= (; x_1 ⅋ x_1(; x, \_ ⅋ x)) \perp ?((; y ⅋ y) \otimes z); z \\
(\lambda\_.(\lambda y.y))^\circ &= ; (; \_ ⅋ (; y ⅋ y)).
\end{aligned}
$$

$((\lambda x.(\lambda\_.x))(\lambda y.y))^\circ$ is reduced to the canonical form $x_1 \perp (; y ⅋ y); x_1(; x, \_ ⅋ x)$ by the following sequence of reductions:

$$
\begin{aligned}
((\lambda x.(\lambda\_.x))(\lambda y.y))^\circ &\longrightarrow_{RP} x_1 \perp (; y ⅋ y), x_1(; x, \_ ⅋ x) \perp z; z \\
&\longrightarrow_{rc} x_1 \perp (; y ⅋ y); x_1(; x, \_ ⅋ x) \\
&\qquad (= (\lambda x.(\lambda\_.x))^\circ[\leftarrow (\lambda y.y)^\circ]).
\end{aligned}
$$

$$\frac{}{\vdash \overline{n} : \mathbf{Nat}}\ (\overline{n}) \qquad \frac{}{\vdash \mathbf{tt} : \mathbf{Bool}}\ (\mathbf{tt}) \qquad \frac{}{\vdash \mathbf{ff} : \mathbf{Bool}}\ (\mathbf{ff})$$

$$\frac{}{\vdash \mathbf{succ} : \mathbf{Nat} \supset \mathbf{Nat}}\ (\mathbf{succ}) \qquad \frac{}{\vdash \mathbf{pred} : \mathbf{Nat} \supset \mathbf{Nat}}\ (\mathbf{pred})$$

$$\frac{}{\vdash \mathbf{iszero} : \mathbf{Nat} \supset \mathbf{Bool}}\ (\mathbf{iszero}) \qquad \frac{}{\vdash \mathbf{cond}_A : \mathbf{Bool} \supset A \supset A \supset A}\ (\mathbf{cond}_A)$$

$$\frac{}{\vdash \mathbf{Y}_{A \supset B} : ((A \supset B) \supset (A \supset B)) \supset (A \supset B)}\ (\mathbf{Y}_{A \supset B})$$

Fig. 5. The typing rules for constants

And we have the following:

$$\begin{aligned} x_1 \bot (; y \parr y); x_1 (; x, \_ \parr x) &\cong ; (x \bot (; y \parr y); \_ \parr x) && \text{(CommutativeCut)} \\ &\cong ; (; \_ \parr (; y \parr y)) && \text{(Inner)} \\ &= (\lambda \_ .(\lambda y.y))^{\circ}. \end{aligned}$$

5 Extensions

In this section, we extend our result to a simple functional programming language obtained from the typed λ'-calculus by adding constants (Booleans, Natural numbers, arithmetic functions, conditional and recursion) from PCF [12]. We extend proof expressions, Linear CHAM and the translation $(-)^{\circ}$. Then we show that the call-by-value evaluation of terms of ground types (Booleans and Natural numbers) is preserved and reflected by this extended translation.

5.1 Adding constants to the typed λ'-calculus

In Figure 5, we give the syntax and the typing rules for constants; here A and B are well-formed types and $\overline{n}$ denotes a constant which corresponds to natural number n. We say that a λ'-term M is *a value* if M is either a closed term of the form $\lambda p.N$, or a constant.

For the rest of this paper, we assume that all λ'-term with constants mentioned are typable.

The reduction rules for constants, called δ-rules, are shown below:

$$\begin{array}{ll} \mathbf{pred}\ \mathbf{0} \to \mathbf{0} & \mathbf{pred}\ \overline{n+1} \to \overline{n} \\ \mathbf{iszero}\ \mathbf{0} \to \mathbf{tt} & \mathbf{iszero}\ \overline{n+1} \to \mathbf{ff} \\ \mathbf{cond}\ \mathbf{tt} \to (\lambda x.\lambda \_.x) & \mathbf{cond}\ \mathbf{ff} \to (\lambda \_.\lambda x.x) \\ \mathbf{Y}\ M \to M(\lambda y.(\mathbf{Y}\ M')y) & \mathbf{succ}\ \overline{n} \to \overline{n+1} \end{array}$$

where M' is the result of replacing all variables occurring in M by fresh variables.

Definition 11. *We define* one-step reduction of λ'-terms with constants in the call-by-value evaluation *by:* $C[M] \Longrightarrow C[N]$ *where either* M *is* $(\lambda p.M')N'$ *and* N *is* $M'[N'/p]$*, or* $M \to N$ *by using a* δ*-rule.*

$$\frac{}{\vdash; \mathrm{tt} :!\mathrm{Bool}}\ (\mathrm{tt}) \qquad \frac{}{\vdash; \mathrm{ff} :!\mathrm{Bool}}\ (\mathrm{ff}) \qquad \frac{}{\vdash; \overline{n} :!\mathrm{Nat}}\ (\overline{n})$$

$$\frac{}{\vdash; \mathrm{succ}(x) :?\mathrm{Nat}^{\perp}, x :!\mathrm{Nat}}\ (\mathrm{succ}) \qquad \frac{}{\vdash; \mathrm{pred}(x) :?\mathrm{Nat}^{\perp}, x :!\mathrm{Nat}}\ (\mathrm{pred})$$

$$\frac{}{\vdash; \mathrm{iszero}(x) :?\mathrm{Nat}^{\perp}, x :!\mathrm{Bool}}\ (\mathrm{iszero})$$

$$\frac{}{\vdash; \mathrm{cond}_{!A}(x) :?\mathrm{Bool}^{\perp}, x :!(?A^{\perp} \parr !(?A^{\perp} \parr !A))}\ (\mathrm{cond}_{!A})$$

$$\frac{}{\vdash; \mathrm{Y}_{A,B} :!(?(!(A^{\perp} \parr B) \otimes ?(A \otimes B^{\perp})) \parr !(A^{\perp} \parr B))}\ (\mathrm{Y}_{A,B})$$

Distinct names are introduced for each instance of the (succ), (pred), (iszero) and $(\mathrm{cond}_{!A})$ rules.

Fig. 6. Extensions of the type assignment system for proof expressions

δ-rules:

$$\begin{cases} c\perp\_ \longrightarrow \\ c\perp u@v \longrightarrow c\perp u, c\perp v \end{cases} \quad \text{(where } c \text{ is a meta symbol denoting tt, ff, Y}, \overline{\mathrm{n}})$$

$$\mathrm{succ}(x)\perp\overline{n} \longrightarrow \overline{n+1}\perp x$$

$$\begin{cases} \mathrm{pred}(x)\perp 0 \longrightarrow x\perp 0 \\ \mathrm{pred}(x)\perp\overline{\mathrm{n}+1} \longrightarrow x\perp\overline{n} \end{cases} \qquad \begin{cases} \mathrm{iszero}(x)\perp 0 \longrightarrow x\perp\mathrm{tt} \\ \mathrm{iszero}(x)\perp\overline{\mathrm{n}+1} \longrightarrow x\perp\mathrm{ff} \end{cases}$$

$$\begin{cases} \mathrm{cond}(x)\perp\mathrm{tt} \longrightarrow x\perp(; y' \parr y'(; y, \_ \parr y)) \\ \mathrm{cond}(x)\perp\mathrm{ff} \longrightarrow x\perp(; \_ \parr (; y \parr y)) \quad \text{(where } y \text{ and } y' \text{ are fresh names)} \end{cases}$$

$$\mathrm{Y}\perp ?(t \otimes z) \longrightarrow z\perp z''', t\perp ?((\mathrm{Y}\perp ?(t' \otimes z'), z'\perp ?(y \otimes z''); y \parr z'') \otimes z''')$$

(where t' is the result of replacing all names occurring in t by fresh names and z', z'', z'' are fresh names)

Fig. 7. Extensions of Reaction rules

Definition 12. *We define $M \Downarrow N$ by: $M \Longrightarrow^* N$, where N is a $\Longrightarrow$-normal form.*

5.2 Adding constants to Linear CHAM

We add the following syntax for terms in proof expressions :

$$\mathrm{succ}(x) \quad \mathrm{pred}(x) \quad \mathrm{iszero}(x) \quad \mathrm{cond}(x) \quad \mathrm{tt} \quad \mathrm{ff} \quad \mathrm{Y}_{A,B} \quad 0 \quad 1 \quad 2 \quad \ldots$$

where x is a name and A and B are well-formed types. Let $\overline{n}$ denote a term which corresponds to natural number n. In Figure 6 and 7, we give the typing rules and the reaction rules for these terms.

Theorem 5 (Weak Diamond Property). *If $P \longrightarrow Q'$,$P \longrightarrow Q''$ and $Q' \not\equiv Q''$, then for some R, $Q' \longrightarrow R$ and $Q'' \longrightarrow R$.*

As the corollary to this theorem, we have the following:

Theorem 6. *If $P \Downarrow Q$, then every reduction sequence starting from P finally terminates in a canonical proof expression Q' and $Q' \equiv Q$. Moreover, all reduction sequences starting from P to a canonical proof expression Q' have the same length.*

5.3 Extensions of the translation from λ-terms into proof expressions

We define translations of ground types (**Nat** and **Bool**) as follows:

$$\mathbf{Nat}^\circ = !\mathrm{Nat}, \ \ \mathbf{Bool}^\circ = !\mathrm{Bool}.$$

The following are translations of constants:

$$\begin{array}{c} (\vdash \overline{n})^\circ = \,;\overline{n} \qquad (\vdash \mathbf{tt})^\circ = \,;\mathrm{tt} \qquad (\vdash \mathbf{ff})^\circ = \,;\mathrm{ff} \\ (\vdash \mathbf{succ})^\circ =; (;\mathrm{succ}(x) \invamp x) \qquad (\vdash \mathbf{pred})^\circ =; (;\mathrm{pred}(x) \invamp x) \\ (\vdash \mathbf{iszero})^\circ =; (;\mathrm{iszero}(x) \invamp x) \qquad (\vdash \mathbf{cond})^\circ =; (;\mathrm{cond}(x) \invamp x) \\ (\vdash \mathbf{Y})^\circ = \,;\mathrm{Y} \end{array}$$

As before, for a closed λ'-term M we simply write M° to mean $(\vdash M)^\circ$.

5.4 Properties of the extended translation

We show that for every closed λ'-term M Linear CHAM computes the same answer as the call-by-value evaluation, up to the equivalence relation $\cong$.

Theorem 7. *If M is a closed λ'-term and $M \Longrightarrow^* N$, then there is a proof expression P such that $M^\circ \longrightarrow^* P$ and $P \cong N^\circ$.*

Proof. We examine only several cases here, but all the other cases can be checked straightforwardly in the similar way as the proof of Theorem 2 and 3.

- $(\mathbf{succ}\ \mathbf{0})^\circ = (;\mathrm{succ}(x) \invamp x)\perp ?(0 \otimes z); z$
  $\longrightarrow_{RP} \mathrm{succ}(x)\perp 0, x\perp z; z \longrightarrow_\delta 1\perp x, x\perp z; z \longrightarrow_{rc} 1\perp z; z \longrightarrow_{rc} \,;1 = \mathbf{1}^\circ$
- $(\mathbf{Y}\ M)^\circ = \mathrm{Y}\perp ?(t \otimes z); z \longrightarrow_\delta \longrightarrow_{rc}$
  $t\perp ?((\mathrm{Y}\perp ?(t' \otimes z'), z'\perp ?(y \otimes z''); y \invamp z'') \otimes z'''); z''' = (M(\lambda y.(\mathbf{Y}\ M')y))^\circ.$

Theorem 8. *If M is a closed λ'-term, $M \Downarrow N$ and $M^\circ \Downarrow Q$, then $Q \cong N^\circ$.*

We prove that evaluation of terms of ground types(Booleans and Natural numbers) is preserved and reflected by the extended translation.

Lemma 2. *Let M be a closed λ'-term of a ground type which is a $\Longrightarrow$ -normal form. If $M^\circ \cong P\,!\,\$$then $P \Downarrow M^\circ\,!\,\%$*

Proof. In the same way as the proof of Lemma 1, we can show that there is a proof expression Q such that $P \Downarrow Q$ and $Q \cong M^\circ$.

For every canonical proof expression P' such that $P' \cong M^\circ$, M° is equal to P' because M° contains neither occurrences of names nor occurrences of $\bar{x}(R)$. Therefore Q is equal to M°.

Theorem 9. *If M is a closed λ'-term of a ground type, then $M \Downarrow N$ iff $M^\circ \Downarrow N^\circ\,!\,\%$*

6 Execution model of Linear CHAM

In this section, we give an execution model of Linear CHAM.

6.1 $\longrightarrow_r$-normal form

First, we show that when we reduce a proof expression applications of Cleanup Rule can be postponed until all applications of Reaction Rules are done.

Definition 13. *We write $P \longrightarrow_r Q$ if $P \longrightarrow Q$ can be derived using Reaction Rules, and we write $P \longrightarrow_c Q$ if $P \longrightarrow Q$ can be derived using Cleanup Rule.*

Lemma 3. – *If $P \longrightarrow_c Q \longrightarrow_r R$, then for some S, $P \longrightarrow_r S \longrightarrow_c R$.*
– *If $P \longrightarrow^* Q$, then for some R, $P \longrightarrow_r^* R \longrightarrow_c^* Q$.*

So, we can concentrate our attention on reductions of coequations using Reaction Rules.

Lemma 4. *If a typable proof expression $\Theta; \bar{t}$ has a $\longrightarrow_r$-normal form $\Xi; \bar{t}$, then every reduction sequence starting from $\Theta; \bar{t}$ and only using Reaction Rules finally terminates in $\Xi'; \bar{t}$ where $\Xi' \leftrightarrows^* \Xi$.*

6.2 Execution model

Abramsky [1] sketched out an implementation technique for Reaction Rules using two data structures: a *stack of coequations* to be processed, and a *name queue*.

We consider these two data structures as lists of coequations, and formulate each reaction rule as an operation on pairs of them.

Definition 14 (An execution model of Linear CHAM). *We define an execution model of Linear CHAM as the following state transition machine:*

– *A state $(\Xi \mid \Theta)$ consists of two lists of coequations Ξ, Θ.*
– *Transition Rules $\longrightarrow_S$ are given as follows:*
  1. $(\Xi_1, x\bot s, \Xi_2 \mid x\bot t, \Theta) \longrightarrow_S (\Xi_1, \Xi_2 \mid s\bot t, \Theta)$
  2. $(\Xi_1, s\bot x, \Xi_2 \mid x\bot t, \Theta) \longrightarrow_S (\Xi_1, \Xi_2 \mid s\bot t, \Theta)$
  3. $(\Xi_1, x\bot s, \Xi_2 \mid t\bot x, \Theta) \longrightarrow_S (\Xi_1, \Xi_2 \mid t\bot s, \Theta)$
  4. $(\Xi_1, s\bot x, \Xi_2 \mid t\bot x, \Theta) \longrightarrow_S (\Xi_1, \Xi_2 \mid t\bot s, \Theta)$
  5. $(\Xi \mid x\bot t, \Theta) \longrightarrow_S (\Xi, x\bot t \mid \Theta)$
  6. $(\Xi \mid t\bot x, \Theta) \longrightarrow_S (\Xi, t\bot x \mid \Theta)$
  7. $(\Xi \mid c, \Theta) \longrightarrow_S (\Xi \mid c_1, ..., c_n, \Theta)$ *where* $c \longrightarrow_r c_1, ..., c_n$.

Rules 1,2,..,7 are checked in this order whether they are applicable to the current state, and if no rule is applicable then the machine stops.

First, we show that for every transition sequence in the machine there is a corresponding reduction path in Linear CHAM.

Lemma 5. *If* $(\varXi \mid \varTheta) \longrightarrow_S (\varXi' \mid \varTheta')$, *then* $\varXi, \varTheta; \bar{t} \longrightarrow_r^* \varXi', \varTheta'; \bar{t}$ *for any sequence of terms* $\bar{t}$.

Next, we show that the machine computes the same answer ($\longrightarrow_r$-normal form) as the evaluation of a proof expression in Linear CHAM, and it terminates if and only if the evaluation in Linear CHAM terminates.

Theorem 10. *Assume that* $\varTheta; \bar{t}$ *is a typable proof expression.*

(1) *If* $\varTheta; \bar{t}$ *has a* $\longrightarrow_r$*-normal form* $\varXi; \bar{t}$, *then a state* $(\mid \varTheta)$ *has a* $\longrightarrow_S$*-normal form* $(\varXi' \mid \,)$ *where* $\varXi \leftrightharpoons^* \varXi'$.
(2) *If* $(\mid \varTheta)$ *has a* $\longrightarrow_S$*-normal form* $(\varXi \mid \,)$, *then* $\varTheta; \bar{t}$ *has a* $\longrightarrow_r$*-normal form* $\varXi; \bar{t}$.

As a Corollary of Theorem 10, we can show that the state transition machine can be used as an evaluator of a simple functional language obtained from the typed λ'-calculus by adding constants from PCF.

Theorem 11. *If* M *is a closed* λ'*-term of a ground type,* $M \Downarrow c$ *and* $M^\circ = \varTheta; x$, *then* $(\mid \varTheta)$ *has a* $\longrightarrow_S$*-normal form* $(x \perp c^\circ \mid \,)$ *or* $(c^\circ \perp x \mid \,)$.

6.3 Concurrency

The state transition machine considered so far is sequential, in the sense that the transition rule applied at each stage is always uniquely determined.

When each name occurring in $(\varXi \mid \varTheta)$ occurs twice at most, we can see from the Weak Diamond Property of Linear CHAM that the order of coequations in $\varTheta$ is irrelevant to the answer computed by the machine.

From this observation, we can give another formulation of the state transition machine by using the following *concurrent* transition rules.

Definition 15 (Concurrent Transition rule). *Transition rules* $\longrightarrow_C$ *are given as follows:*

$$\frac{(\varXi \mid c_i) \longrightarrow_S (\varXi' \mid \varTheta')}{(\varXi \mid c_1, ..., c_i, ..., c_n) \longrightarrow_C (\varXi' \mid c_1, ..., \varTheta', ..., c_n)}$$

In the similar way to Theorem 10, we can show the correctness of the state transition machine which uses $\longrightarrow_C$ instead of $\longrightarrow_S$.

Now we can consider a multi-thread implementation of the machine, where $\varXi$ is implemented as a *nameserver* and a sequence $c_1, ..., c_i, ..., c_n$ is implemented as a *pool of threads*.

According to Transition Rules 1,2,3 and 4, processing a thread $x \perp t$ (or $t \perp x$) makes the nameserver forget the value s of a name x and creates a new thread $s \perp t$. According to Rules 5 and 6, processing a thread $x \perp t$ (or $t \perp x$) makes the nameserver store t as the value of a name x. According to Rule 7, processing a thread c creates new threads $c_1, ..., c_n$.

Note that as to Rules 1,2,3,4,5 and 6, some kind of synchronization mechanism is needed, while as to Rule 7 which does not refer to $\varXi$, threads can be processed in parallel.

7 Conclusions

We have shown that the typed λ-calculus is embedded into Linear CHAM by Girard's embedding of Intuitionistic Logic into Linear Logic. In addition, we have extended this result to a simple functional programming language with constants. Finally, we have introduced sequential and concurrent execution models of Linear CHAM. From these results, we can expect that Linear CHAM can be used as a foundation of concurrent functional languages with types.

One of the future applications of this work is to extend this result to more realistic functional programming languages which have data-types and control operator for continuations and so on. Another interesting research topic is to study the relationship between optimal reduction in λ-calculus and Linear CHAM. We can take sharing into account because the resources are manipulated explicitly in Linear Logic, and this is one of the main interests of using Linear Logic for implementation purpose. Mackie [9] investigated the efficiency of embeddings of λ-calculus into interaction nets. Because there are similarities between interaction nets and Linear CHAM, it will be interesting to examine the efficiency of embeddings of λ-calculus into Linear CHAM.

References

1. Samson Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 111:3–57, 1993.
2. P. N. Benton, G. M. Bierman, V. C. V. de Paiva, and J. M. E. Hyland. A term calculus for intuitionistic linear logic. In *the First International Conference on Typed Lambda Calculi and applications*, LNCS 664, Springer Verlag, 1993.
3. G. Berry and G. Boudol. The chemical abstract machine. In *POPL'90*, 1990.
4. Torben Braüner. *An Axiomatic Approach to Adequacy*. PhD thesis, BRICS, Aarhus, Danemark, 1996.
5. Maribel Fernández and Ian Mackie. A calculus for interaction nets. In *the International Conference on Principles and Practice of Declarative Programming (PPDP'99)*, LNCS 1702, Springer-Verlag, 1999.
6. Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
7. Jean-Yves Girard. Linear logic and parallelism. In *Mathematical Models for the Semantics of Parallelism*, LNCS 280, Springer Verlag, 1987.
8. Yves Lafont. Interaction nets. In *POPL'90*, 1990.
9. Ian Mackie. YALE: Yet another lambda evaluator based on interaction nets. In *the Third International Conference on Functional Programming*, ACM Press, 1998.
10. J. Maraist, M. Odersky, D. N. Turner, and P. Wadler. Call-by-name, call-by-value, call-by-need and the linear lambda calculus. *Theoretical Computer Science*, 228:175–210, 1999.
11. Seikoh Mikami and Yohji Akama. A study of Abramsky's linear chemical abstract machine. In *the Fourth International Conference on Typed Lambda Calculi and Applications*, LNCS 1581, Springer Verlag, 1999.
12. Gordon Plotokin. Lcf considered as a programming language. *Theoretical Computer Science*, 5:223–256, 1977.
13. P. Wadler. A syntax for linear logic. In *the Ninth International Conference on the Mathematical Foundations of Programming Semantics*, LNCS 802, Springer Verlag, 1993.