Encoding MELL Cut Elimination into a Hierarchical Graph Rewriting Language

Kento Takyu
Waseda University
Tokyo, Japan
takyu@ueda.info.waseda.ac.jp

Kazunori Ueda
Waseda University
Tokyo, Japan
ueda@ueda.info.waseda.ac.jp

Abstract

We show that several key language constructs of the hierarchical graph rewriting language LMNtal correspond directly to the operations required for the cut elimination of MELL (Multiplicative Exponential Linear Logic) Proof Nets, and that cut elimination of MELL can be succinctly encoded into LMNtal. We encoded and ran the cut elimination rules in LMNtal and demonstrated that the implementation serves as a workbench of MELL proof reduction.

Keywords: Linear Logic, Proof Nets, Cut Elimination, Hierarchical Graph Rewriting, LMNtal

1 Introduction

1.1 LMNtal: a hierarchical graph rewriting language

LMNtal [10], a hierarchical graph rewriting language, is a programming and modeling language that simultaneously supports two structuring mechanisms, connectivity and hierarchy. Its implementation, SLIM [4], provides model checking and ordinary program execution in a single framework. One of the key features of LMNtal is its ability to manipulate hierarchical structures using membranes and accompanying language constructs. This allows us to express what are difficult to achieve with ordinary graphs. For example, the ambient calculus, a model of concurrency with mobility and hierarchy, can be easily encoded into LMNtal [8].

1.2 Contribution

In this study, we show that Multiplicative Exponential Linear Logic (MELL) Proof Nets and proof reduction by cut elimination [2, 3] can be encoded in LMNtal in a concise manner. In addition to their general graph structure, MELL Proof Nets come with a hierarchical structure (called a *promotion box*) to properly handle exponential operators, and the latter turns out to allow straightforward encoding into LMNtal which can handle hierarchical structures along with general graph transformation.

The model checking functionality of the LMNtal system makes it possible to construct state spaces of cut elimination procedures. Since MELL Proof Nets can encode simply-typed λ-calculus, and cut elimination corresponds to β-reduction [6] (Curry-Howard isomorphism), LMNtal is expected to serve as a workbench for understanding and analyzing these concepts.

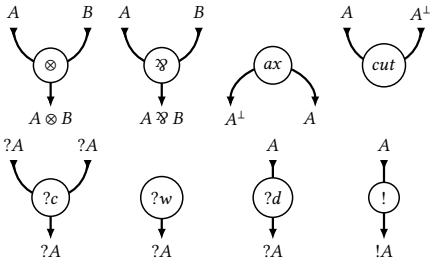

Figure 1. Components of MELL proof structure.

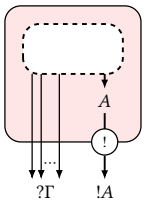

Figure 2. Promotion box.

```
tensor(A,B,C), par(A,B,C), ax{+A,+B}, cut{+A,+B}.
'?c'({+A,+B},C), '?w'(A), '?d'(A,B), '!'(A,B).
```

Figure 3. The LMNtal encoding of Fig. 1.

2 Proof Nets

2.1 Multiplicative Exponential Linear Logic

This study focuses on Multiplicative Exponential Linear Logic (MELL). MELL is an extension of Multiplicative Linear Logic (MLL), which is a fundamental fragment of linear logic. MELL adds two exponential operators '!' and '?' to MLL to allow non-linear, "classical" handling of resources.

Figure 1 shows the components of MELL Proof Structure, a directed, acyclic multigraph consisting of (i) cells (nodes) labelled with inference rules of MELL and (ii) wires (edges) labelled with MELL formulas. Only the cells ⊗ and ⅋ have two *ordered* inputs, while the inputs of other cells are unordered. MELL Proof Nets combine these components to form a proof structure.

The inference rule for the bottom-right component of Fig. 1, called *promotion*, is actually a contextual rule; the rest of the formulas of the sequent containing the $!A$ must come

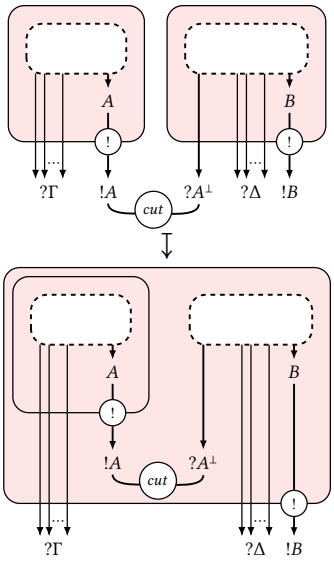

Figure 4. Cut elimination rule for the connection of two boxes.

```
box_cut_elimination_nested@@
{'!'(X1,X2),$g1[X1|*X]},{$g2[X3|*X]},cut{+X2,+X3}
  :- {
      {'!'(X1,X2),$g1[X1|*X]},$g2[X3|*X]
      cut{+X2,+X3}
     }.
```

Figure 5. The LMNtal encoding of Fig. 4.

with '?'s. In order to handle this constraint, a *(promotion) box* (Fig. 2) is used as a standard mechanism to protect its contents from transformation and to control the order of proof reductions [2]. How to express the nested structure of promotion boxes using only edges and nodes of ordinary graphs is not obvious, if not impossible, and has been a topic of active research.

2.2 Cut Elimination

Proof nets containing cuts can be reduced to simpler forms by applying the cut elimination rules. In addition, proof reduction by the cut elimination rules has confluence and strong normalization properties [2, 3].

3 Encoding Cut Elimination into LMNtal

3.1 Hierarchical constructs of LMNtal

LMNtal provides various constructs for handling hierarchical structures. The following is a list of constructs used in this study.

- **membrane**: Encloses the elements of a graph with `{}` to express hierarchical structure. This enables protection from transformation.

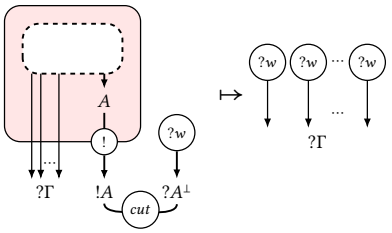

Figure 6. Cut elimination rule for weakening.

```
box_cut_elimination_weakening_step1@@
{'!'(X1,X2),$g[X1|*X]}, '?w'(X3),
cut{+X2,+X3}
  :- nlmem.kill({trash(X1),$g[X1|*X]},'?w').
```

Figure 7. The LMNtal encoding of Fig. 6.

- **process context**: Written in the form of `$p[...]` within a membrane and matches all non-explicit elements in the membrane. The arguments `[...]` specifies the free links (edges connected to the outside) of the context.
- **bundle**: Written in the form `*X` specified as the final argument of a process context. Matches all non-explicit free links of the process context; e.g., `$p[A|*X]` stands for a graph with the free link `A` and zero or more free links represented by `*X`.

Rewrite rules of LMNtal allow the copying and removing of membranes and process contexts. Membranes with process contexts that may be copied or removed are called *nonlinear membranes*.

3.2 Encoding into LMNtal

Using these functionalities, the cut elimination rules of MELL Proof Nets can be encoded on LMNtal. These features not only enable the representation of nested structures, as handled by promotion boxes, but also closely correspond to the cut elimination rules, making the encoding in LMNtal straightforward. We give a few examples below.

First, we show in Fig. 3 the LMNtal encoding of each of the components of Fig. 1. We need to represent two types of inputs: those with and without order. An atom `n(A,B)` represents a node named `n` with two ordered links, while a membrane `n{+A,+B}` represents a node named `n` (which is optional) with two unordered links `A` and `B` (terminated by unary atoms '+'). In Fig. 3, `'?c'({+A,+B},C)` is an abbreviation of `'?c'(D,C),{+A,+B,+D}`.

Figure 4 shows the cut-elimination rule for structures where the !-cell of a box is connected (via a cut-cell) to its dual *inside* another box, forming a nested structure of boxes as shown in the figure. Figure 5 shows an LMNtal program encoding Fig. 4. Such rules involving wire bundles and hierarchical structures can be straightforwardly expressed using

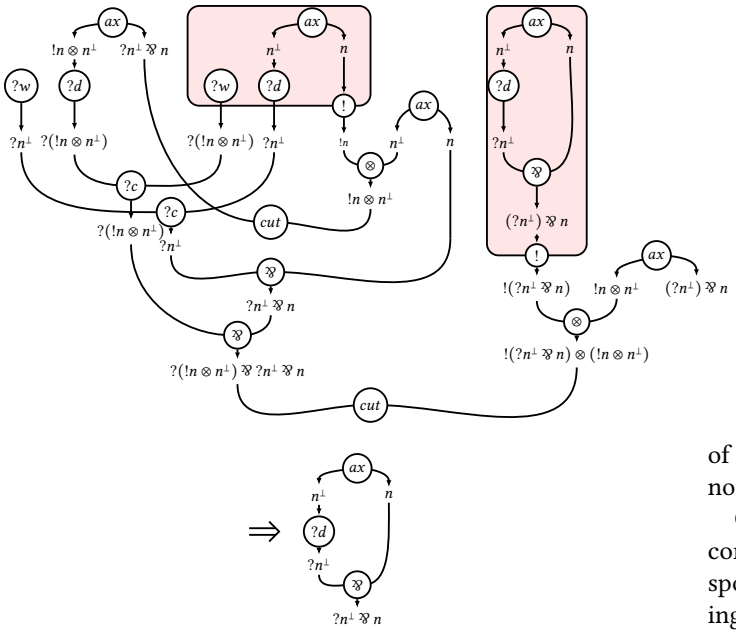

Figure 8. An example of proof net and reduction of a λ calculus.

```
ax{+A1,+A2},'?d'(A1,A3),'?w'(A4).
{
  ax{+B1,+B2},'?d'(B1,B3).
  '?w'(B4),'!'(B2,B5).
}.
'?c'({+A3,+B4}+C1),'?c'({+A4,+B3},C2).
tensor(B5,T1,D2),ax{+T1,+T2}.
cut{+A2,+D2}.
par(C2,T2,P1),par(C1,P1,F).
{
  ax{+E1,+E2},'?d'(E1,E3).
  par(E3,E2,E4),'!'(E4,E5).
}.
tensor(E5,T3,D4),ax{+T3,+T4}.
cut{+F,+D4}.
formula(T4).
```

Figure 9. The LMNtal encoding of Fig. 8.

LMNtal's bundles and membranes: the process contexts and bundles concisely express the matching, while the formation of nested structure is intuitively expressed with the braces.

Figure 6 shows the rule for a structure in which the !-cell of a box and a ?w-cell (standing for weakening) outside a box are connected by a cut-cell. A multiset $?\Gamma = \{?F_1, ?F_2, \ldots, ?F_n\}$, where F_i is a formula, is extracted from the box, and each member is connected to a ?w-cell. Figure 7 shows an LMNtal program encoding Fig. 6. The structure

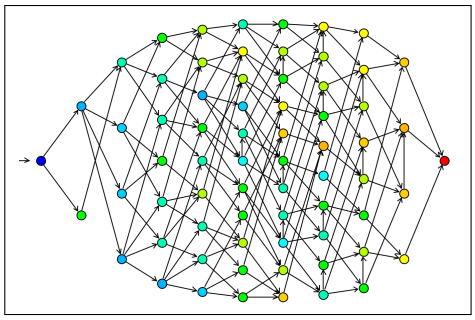

Figure 10. State Space of Fig 8.

of such a rule can be written in simple code by using the nonlinear membrane feature of LMNtal.

Other cut elimination rules can be encoded in a similarly concise manner. In particular, the rule for a ?c-cell (corresponding to contraction) can be encoded by using the copying of non-linear membranes.

We implemented all these rules in the LMNtal implementation SLIM and demonstrated that the reduction of actual proof structures could be performed. The six cut elimination rules are written in about 30 lines of code. SLIM also enables the construction of the state-space of cut elimination.

3.3 Example: Simply Typed Lambda Calculus

Figure 8 shows an encoding of simply-typed λ-calculus in Proof Nets. As is well known, the function type $A \to B$ can be mapped to $?A^{\perp} \parr B$, and cut elimination corresponds to β-reduction. Figure 8 shows the proof net representation of the λ-term $(\lambda f : n \to n \,.\, \lambda x : n \,.\, fx)(\lambda x : n \,.\, x)$ and the result of cut-elimination, and Fig. 9 shows an example of the LMNtal encoding of the initial proof net of Fig. 8. Figure 10 shows the state space of Fig. 8, visualized by LaViT (the LMNtal Visual Tool) [11]. It can be confirmed that the state transition always reaches a unique (red) state.

4 Conclusion and Future Work

We have demonstrated that the language constructs of LMNtal for representing hierarchies, i.e., membranes, process contexts and bundles, provide exactly what are needed for the concise encoding of cut elimination rules of MELL proof nets. There are many variations of proof-net representation of MELL, with and without boxes (e.g., [1, 5]), and LMNtal is expected to serve as a workbench for understanding and analyzing them. We note that several encodings of the λ-calculus in LMNtal have been proposed and implemented, including a fine-grained one [9] (inspired by [7]) that uses membranes in a different manner for scope management. It is expected that the encoding of cut elimination inspired by [9] provides another way towards the the connection of proof nets, λ-calculus, and graph rewriting.

References

[1] Beniamino Accattoli. 2015. Proof nets and the call-by-value λ-calculus. *Theoretical Computer Science* 606 (2015), 2–24. https://doi.org/10.1016/j.tcs.2015.08.006

[2] Jean-Yves Girard. 1987. Linear logic. *Theoretical Computer Science* 50, 1 (1987), 1–101. https://doi.org/10.1016/0304-3975(87)90045-4

[3] Jean-Yves Girard, Yves Lafont, and Paul Taylor. 1989. *Proofs and Types*. Number 7 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press.

[4] Masato Gocho, Taisuke Hori, and Kazunori Ueda. 2011. Evolution of the LMNtal runtime to a parallel model checker. *Computer Software* 28, 4 (2011), 137–157. https://doi.org/10.11309/jssst.28.4_137

[5] G. Gonthier, M. Abadi, and J.-J. Levy. 1992. Linear Logic Without Boxes. In *Proceedings of the Seventh Annual IEEE Symposium on Logic in Computer Science (LICS 1992)*. 223–234. https://doi.org/10.1109/LICS.1992.185535

[6] Stefano Guerrini, Simone Martini, and Andrea Masini. 2001. Proof nets, garbage, and computations. *Theoretical Computer Science* 253, 2 (2001), 185–237. https://doi.org/10.1016/S0304-3975(00)00094-3

[7] François-Régis Sinot. 2005. Call-by-Name and Call-by-Value as Token-Passing Interaction Nets. In *Typed Lambda Calculi and Applications - TLCA 2005*, Paweł Urzyczyn (Ed.). Springer-Verlag, Berlin, Heidelberg, 386–400. https://doi.org/10.1007/11417170_28

[8] Kazunori Ueda. 2008. Encoding Distributed Process Calculi into LMNtal. *Electronic Notes in Theoretical Computer Science* 209 (2008), 187–200. https://doi.org/10.1016/j.entcs.2008.04.012

[9] Kazunori Ueda. 2008. Encoding the Pure Lambda Calculus into Hierarchical Graph Rewriting. In *Rewriting Techniques and Applications - RTA 2008*, Andrei Voronkov (Ed.). Springer-Verlag, Berlin, Heidelberg, 392–408.

[10] Kazunori Ueda. 2009. LMNtal as a hierarchical logic programming language. *Theoretical Computer Science* 410, 46 (2009), 4784–4800. https://doi.org/10.1016/j.tcs.2009.07.043

[11] Kazunori Ueda, Takayuki Ayano, Taisuke Hori, Hiroki Iwasawa, and Seiji Ogawa. 2009. Hierarchical Graph Rewriting as a Unifying Tool for Analyzing and Understanding Nondeterministic Systems. In *Theoretical Aspects of Computing - ICTAC 2009*, Martin Leucker and Carroll Morgan (Eds.). Springer-Verlag, Berlin, Heidelberg, 349–355.