Enhancing a Hieralchical Graph Rewriting Language based on MELL Cut Elimination

PADL2025 @ Denver

January 20, 2025

Kento Takyu Kazunori Ueda

Waseda University, Tokyo, Japan

(Extended version at arXiv.org)

Overview

Hierarchical Graph Rewriting

- ✔ Expressive formalism that subsumes term rewriting, process calculus, etc.
- ! Designing a **practical high-level declarative language** based on hierarchical graph rewriting is still a challenge.
  - The “right” construct for **graph cloning and deletion** is highly non-trivial.

LMNtal[1]

- ✔ **Concrete PL** for hierarchical graphs
- ✔ We propose **well-motivated graph cloning and deletion constructs**

1. pronounced ”elemental”

\+

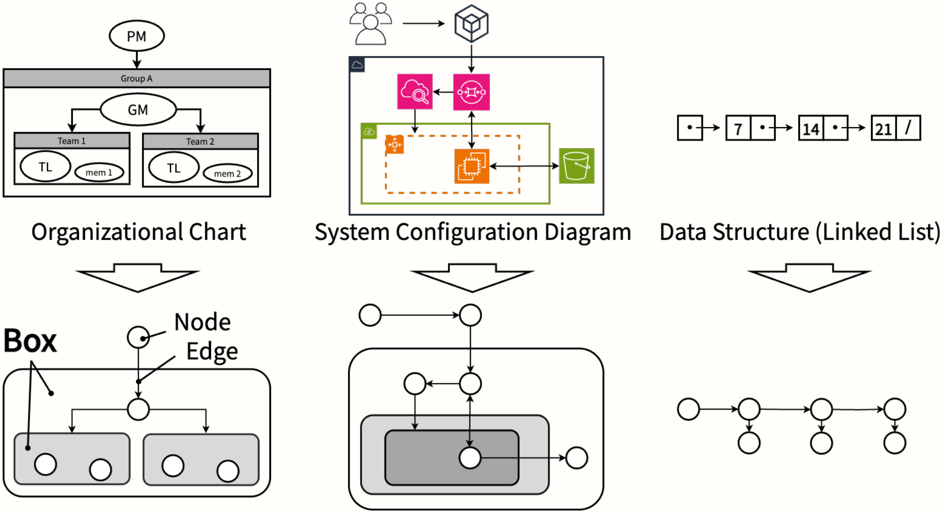
MELL[2] proof nets

- ✔ Hierarchical graph rewriting for Linear Logic proofs
- ✔ involving **cloning** and **deletion**

2. Multiplicative Exponential Linear Logic

Hierarchical graphs

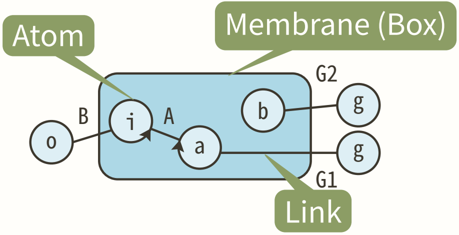
Supports two structuring mechanisms: **connectivity** and **hierarchy**



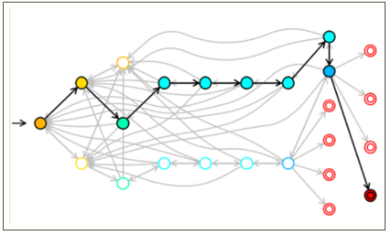
Hierarchical Graphs (Nodes + Edges + **Boxes**) can represent all these.

LMNtal (https://bit.ly/lmntal-portal)

- A hierarchical graph rewriting language inspired by **concurrent logic languages** and **Constraint Handling Rules** [Ued09]
  - graph nodes as **atom**(ic formula)**s** (no constants/functors)
  - links as (zero-assignment) **logical variables**
- Comes with a parallel **model checker** with **state space visualizer**



LMNtal graph



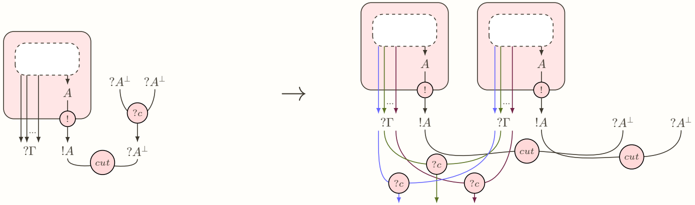
State space visualization of LTL model checking

MELL proof nets

Graph representation of MELL (Multiplicative Exponential LL) proofs [Gir87]

- Cut elimination expressed as hierarchical graph rewriting rules
- Some rules clone or delete hierarchical graphs

$$\dfrac{\dfrac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}\,! \qquad \dfrac{\vdash \Delta, ?A^{\perp}, ?A^{\perp}}{\vdash \Delta, ?A^{\perp}}\,?c}{\vdash ?\Gamma, \Delta}\,cut \quad \longrightarrow \quad \dfrac{\dfrac{\dfrac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}\,! \qquad \dfrac{\dfrac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}\,! \qquad \vdash \Delta, ?A^{\perp}, ?A^{\perp}}{\vdash ?\Gamma, \Delta, ?A^{\perp}}\,cut}{\vdash ?\Gamma, ?\Gamma, \Delta}\,cut}{\vdash ?\Gamma, \Delta}\,?c*$$



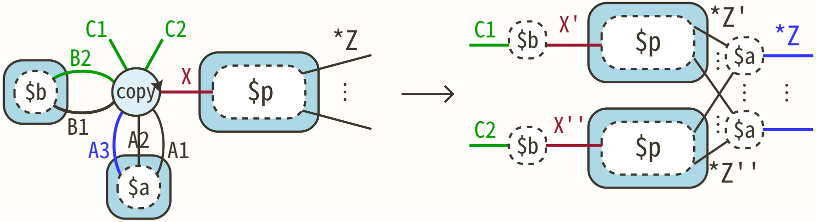
→ Cut elimination of MELL proof nets could provide a useful design

Contributions

1. We extended LMNtal with **process context aggregates**, and **designed and implemented the `mell` API** for graph cloning and deletion.

2. We showed that **MELL proof nets and their cut elimination rules can be directly encoded** into LMNtal.

3. We encoded several process calculi and demonstrated the **generality of LMNtal with the proposed constructs**.



Formulation of hierarchical graph rewriting systems

Two approaches:

(1) (traditional) **Algebraic approach** (e.g., using pushout in category theory)

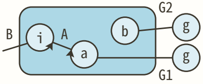
(2) **PL-style approach** with abstract syntax and small-step semantics, where graphs are represented by terms subject to structural congruence

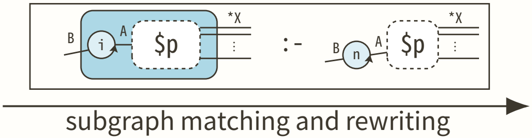
Hierarchical Graph Rewriting System	Model or Language?	**Definition Style**
CHAM(Chemical Abstract Machine) [BB92]	model	**(2)**
BRS(Bigraphical Reactive System) [Mil01]	model	(1)
AHP(Attributed Hierarchical Portgraph) [EFP18]	model	(1)
LMNtal [Ued09]	**language**	**(2)**

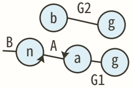
Syntax of LMNtal

$$
\begin{array}{rrcl}
\text{(process)} & P & ::= & \mathbf{0} \mid p(X_1,\dots,X_n) \mid P,P \mid m\{P\} \mid T\,\texttt{:-}\,T \\
\text{(process template)} & T & ::= & \mathbf{0} \mid p(X_1,\dots,X_n) \mid T,T \mid m\{T\} \mid T\,\texttt{:-}\,T \\
 & & & \mid \texttt{@}p \mid \texttt{\$}p[X_1,\dots,X_n \mid A] \mid p({}^{*}X_1,\dots,{}^{*}X_n) \\
\text{(residual)} & A & ::= & [\,] \mid {}^{*}X
\end{array}
$$

Example:








```
g(G1),g(G2),
{i(A,B),a(A,G1),b(G2)}.
```

```
{i(A,B),$p[A|*X]} :- n(A,B),$p[A|*X].
```

```
b(G2),g(G2),
n(A,B),a(A,G1),g(G1).
```

Syntax of LMNtal

Atom (Node) with ordered links (edges)

Membrane (Box)

$$
\begin{array}{rrcl}
\text{(process)} & P & ::= & \mathbf{0} \mid p(X_1,\ldots,X_n) \mid P,P \mid m\{P\} \mid T \text{ :- } T \\
\text{(process template)} & T & ::= & \mathbf{0} \mid p(X_1,\ldots,X_n) \mid T,T \mid m\{T\} \mid T \text{ :- } T \\
 & & & \mid @p \mid \$p[X_1,\ldots,X_n \mid A] \mid p(^*X_1,\ldots,^*X_n) \\
\text{(residual)} & A & ::= & [\,] \mid {}^*X
\end{array}
$$

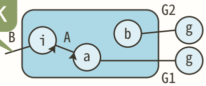
Process Context (Wildcard)

Rule (Subgraph rewriting rule)

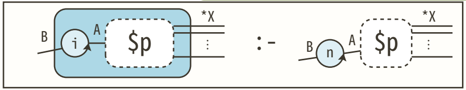
Bundle (Unspecified number of free links)

Example:

Free link

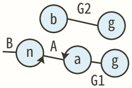



```
g(G1),g(G2),
{i(A,B),a(A,G1),b(G2)}.
```



subgraph matching and rewriting

```
{i(A,B),$p[A|*X]} :- n(A,B),$p[A|*X].
```



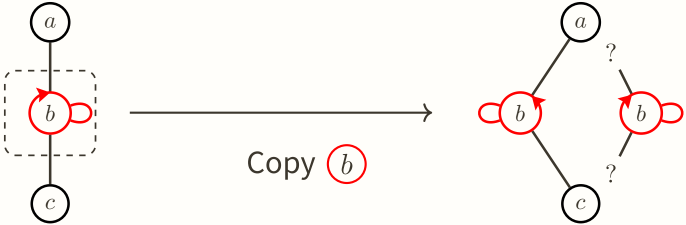

```
b(G2),g(G2),
n(A,B),a(A,G1),g(G1).
```

– Graphs with fixed-arity nodes are often called **port graphs**;

– **Free links** (half edges) play key roles in subgraph matching and rewriting

Subgraph cloning (1)

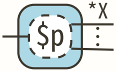
Cloning of subgraphs is a highly expected feature of high-level graph rewriting languages, but how to handle edges of the clones is not obvious. For example,



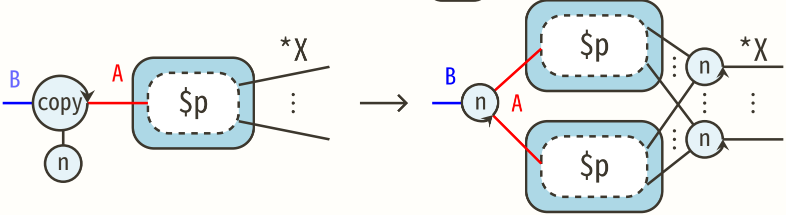
- In LMNtal which handles port graphs, a and c have fixed arities.

We must somehow splice two clones of the subraph into the context of the original subgraph.

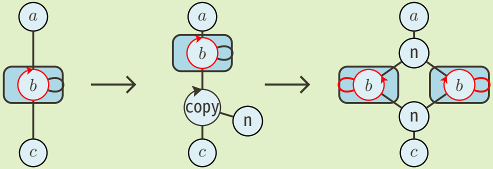
Subgraph cloning (2): nlmem

LMNtal's `nlmem` (nonlinear membrane) API [Inu+08] provides subgraph (box) cloning. For instance, `nlmem.copy` clones 

as:



example:



Can this design be supported by mathematical/logical background?

Subgraph cloning (3)

Design of graph cloning constructs is highly non-trivial even in (traditional) algebraic approaches.

Graph Rewriting System with cloning	Definition Style	**Background**
SePO [Cor+06], PBPO+[OER21]	Category Theory	✓
DLGRS [BES18]	Description Logic	✓
LMNtal with nlmem	PL standard	?
Present Work	PL standard	✓

Research Question:

Can we justify our language design based on **some mathematical/logical system** as a design guideline?

Multiplicative Exponential Linear Logic (MELL)

Definition (MELL Formula)

$$(\text{formula})\ F ::= X\ (\textit{atomic}) \mid X^{\perp} \mid F \otimes F \mid F \parr F \mid !F \mid ?F$$

(Binding strength: $\{^{\perp}\} > \{!, ?\} > \{\otimes, \parr\}$)

Negation $^{\perp}$ is moved/removed by:

$$A^{\perp\perp} := A \quad (A \otimes B)^{\perp} := A^{\perp} \parr B^{\perp} \quad (A \parr B)^{\perp} := A^{\perp} \otimes B^{\perp}$$

$$(!A)^{\perp} := ?(A^{\perp}) \quad (?A)^{\perp} := !(A^{\perp})$$

Linear implication $\multimap$ is defined by:

$$A \multimap B := A^{\perp} \parr B$$

Classical implication $\rightarrow$ can be translated by using linear implication $\multimap$:

$$A \rightarrow B \ \mapsto\ !A \multimap B$$

MELL inference rules (one-sided)

A, B: formulae, Γ, Δ: multisets of formulae

$$\frac{}{\vdash A, A^\perp}\, ax \qquad \frac{\vdash \Gamma, A \qquad \vdash \Delta, A^\perp}{\vdash \Gamma, \Delta}\, cut \qquad \frac{}{\vdash}\, mix_0$$

$$\frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}\, \parr \qquad \frac{\vdash \Gamma, A \qquad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B}\, \otimes \qquad \frac{\vdash \Gamma \qquad \vdash \Delta}{\vdash \Gamma, \Delta}\, mix_2$$

$$\frac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}\, ! \qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, ?A}\, ?d \qquad \frac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ?A}\, ?c \qquad \frac{\vdash \Gamma}{\vdash \Gamma, ?A}\, ?w$$

(promotion rule)
(Note: ‘?’ on the Γ)

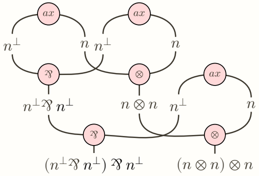
The cut elimination theorem holds [Gir87].

MELL proof nets

MELL proof nets are graph representations of proofs of MELL sequents

with higher level of abstraction.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash n^\perp, n}\,ax \quad \dfrac{}{\vdash n^\perp, n}\,ax}{\vdash n^\perp, n^\perp, (n \otimes n)}\,\otimes \quad \dfrac{}{\vdash n^\perp, n}\,ax}{\vdash n^\perp, n^\perp, n^\perp, (n \otimes n) \otimes n}\,\otimes}{\vdash (n^\perp \parr n^\perp), n^\perp, (n \otimes n) \otimes n}\,\parr}{\vdash (n^\perp \parr n^\perp) \parr n^\perp, (n \otimes n) \otimes n}\,\parr \quad \mapsto$$



$$\mapsfrom \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash n^\perp, n}\,ax \quad \dfrac{}{\vdash n^\perp, n}\,ax}{\vdash n^\perp, n^\perp, (n \otimes n)}\,\otimes}{\vdash (n^\perp \parr n^\perp), (n \otimes n)}\,\parr \quad \dfrac{}{\vdash n^\perp, n}\,ax}{\vdash (n^\perp \parr n^\perp), n^\perp, (n \otimes n) \otimes n}\,\otimes}{\vdash (n^\perp \parr n^\perp) \parr n^\perp, (n \otimes n) \otimes n}\,\parr$$

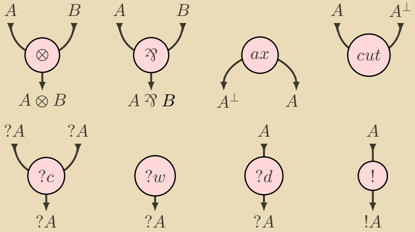
MELL proof nets are defined in two steps:

1. graph structures (**proof structures**)
2. geometric constraints on proof structures (using **switching graphs**).

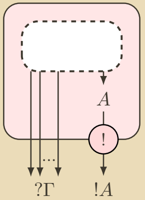
MELL proof structures

Definition (MELL proof structures)

An **MELL proof structure** is a directed acyclic multigraph that combines the cells and wires and the promotion box.



Cells and wires



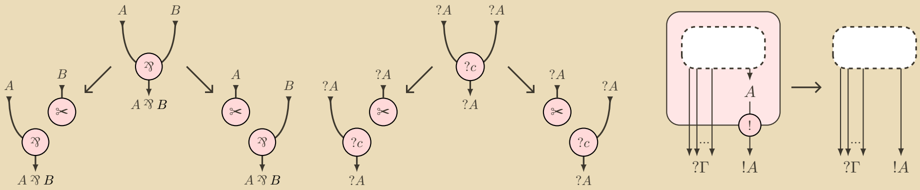
$$\frac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}\,!$$

Promotion box

MELL proof nets

Definition (MELL proof nets)

- A **switching** for a proof structure is a choice of left or right for $⅋$ and $?c$.
- A **switching graph** is obtained by rewriting all $⅋$ and $?c$ cells according to switching, and removing the outer frame of all promotion boxes.



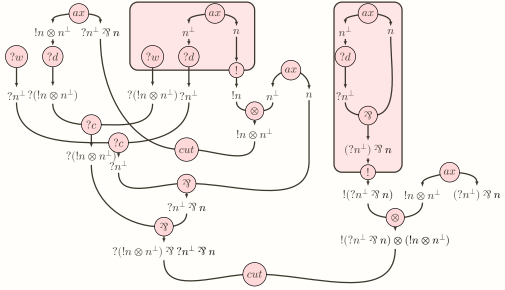
- A **MELL proof net** is a proof structure whose switching graphs have no undirected cycles and such that the content of each box is a proof net, inductively.

MELL proof nets can be converted to MELL (+ mix rule) sequents.

An example of MELL proof nets

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{ax}
\UnaryInfC{$\vdash !n \otimes n^\perp, ?n^\perp \parr n$}
\RightLabel{$?d$}
\UnaryInfC{$\vdash ?(!n \otimes n^\perp), ?n^\perp \parr n$}
\RightLabel{$?w$}
\UnaryInfC{$\vdash ?(!n \otimes n^\perp), ?n^\perp, ?n^\perp \parr n$}
\AxiomC{}
\RightLabel{ax}
\UnaryInfC{$\vdash n^\perp, n$}
\RightLabel{$?d$}
\UnaryInfC{$\vdash ?n^\perp, n$}
\RightLabel{$?w$}
\UnaryInfC{$\vdash ?(!n \otimes n^\perp), ?n^\perp, n$}
\RightLabel{$!$}
\UnaryInfC{$\vdash ?(!n \otimes n^\perp), ?n^\perp, !n$}
\AxiomC{}
\RightLabel{ax}
\UnaryInfC{$\vdash n, n^\perp$}
\RightLabel{$\otimes$}
\BinaryInfC{$\vdash ?(!n \otimes n^\perp), ?n^\perp, n, !n \otimes n^\perp$}
\RightLabel{cut}
\BinaryInfC{$\vdash ?(!n \otimes n^\perp), ?(!n \otimes n^\perp), ?n^\perp, ?n^\perp, n$}
\RightLabel{$?c$}
\UnaryInfC{$\vdash ?(!n \otimes n^\perp), ?(!n \otimes n^\perp), ?n^\perp, n$}
\RightLabel{$?c$}
\UnaryInfC{$\vdash ?(!n \otimes n^\perp), ?n^\perp, n$}
\RightLabel{$\parr$}
\UnaryInfC{$\vdash ?(!n \otimes n^\perp), ?n^\perp \parr n$}
\RightLabel{$\parr$}
\UnaryInfC{$\vdash ?(!n \otimes n^\perp) \parr ?n^\perp \parr n$}
\AxiomC{}
\RightLabel{ax}
\UnaryInfC{$\vdash n^\perp, n$}
\RightLabel{$?d$}
\UnaryInfC{$\vdash ?n^\perp, n$}
\RightLabel{$\parr$}
\UnaryInfC{$\vdash ?n^\perp \parr n$}
\RightLabel{$!$}
\UnaryInfC{$\vdash !(?n^\perp \parr n)$}
\AxiomC{}
\RightLabel{ax}
\UnaryInfC{$\vdash !n \otimes n^\perp, ?n^\perp \parr n$}
\RightLabel{$\otimes$}
\BinaryInfC{$\vdash !(?n^\perp \parr n) \otimes (!n \otimes n^\perp), ?n^\perp \parr n$}
\RightLabel{cut}
\BinaryInfC{$\vdash ?n^\perp \parr n$}
\end{prooftree}
$$

$$\mapsto$$



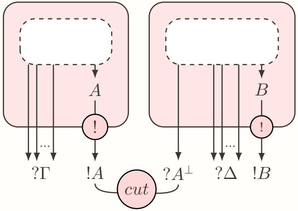
(Corresponds to the typing of $(\lambda f : n \to n \,.\, \lambda x : n \,.\, f\ x)(\lambda x : n \,.\, x)$, see later.)

Some examples of cut elimination rules (1) (!-!)

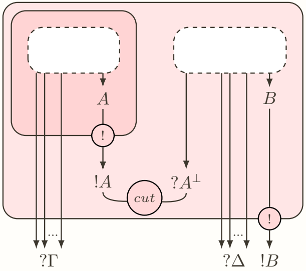
The cut elimination rules for MELL proof nets are hierarchical graph rewriting rules corresponding to those for MELL sequent calculus.

$$\dfrac{\dfrac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}\,! \qquad \dfrac{\vdash ?\Delta, B, ?A^{\perp}}{\vdash ?\Delta, !B, ?A^{\perp}}\,!}{\vdash ?\Gamma, ?\Delta, !B}\,cut \quad \longrightarrow_{(!\text{-}!)} \quad \dfrac{\dfrac{\dfrac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}\,! \qquad \vdash ?\Delta, B, ?A^{\perp}}{\vdash ?\Gamma, ?\Delta, B}\,cut}{\vdash ?\Gamma, ?\Delta, !B}\,!$$

$\downarrow \qquad\qquad \downarrow$



$\longrightarrow_{(!\text{-}!)}$

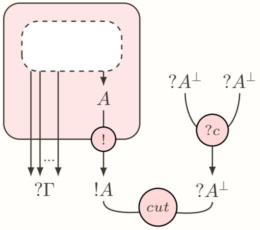


This rule involves box **migration**.

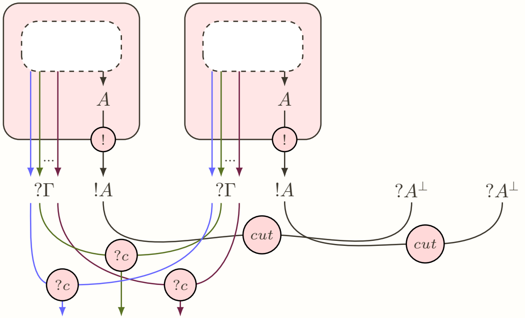
Some examples of cut elimination rules (2) $(!\text{-}?c)$

$$\dfrac{\dfrac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}\,! \qquad \dfrac{\vdash \Delta, ?A^\perp, ?A^\perp}{\vdash \Delta, ?A^\perp}\,?c}{\vdash ?\Gamma, \Delta}\,cut \quad \longrightarrow_{(!\text{-}?c)} \quad \dfrac{\dfrac{\dfrac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}\,! \qquad \dfrac{\dfrac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}\,! \qquad \vdash \Delta, ?A^\perp, ?A^\perp}{\vdash ?\Gamma, \Delta, ?A^\perp}\,cut}{\vdash ?\Gamma, ?\Gamma, \Delta}\,cut}{\vdash ?\Gamma, \Delta}\,?c*$$

$$\downarrow \qquad\qquad\qquad \downarrow$$

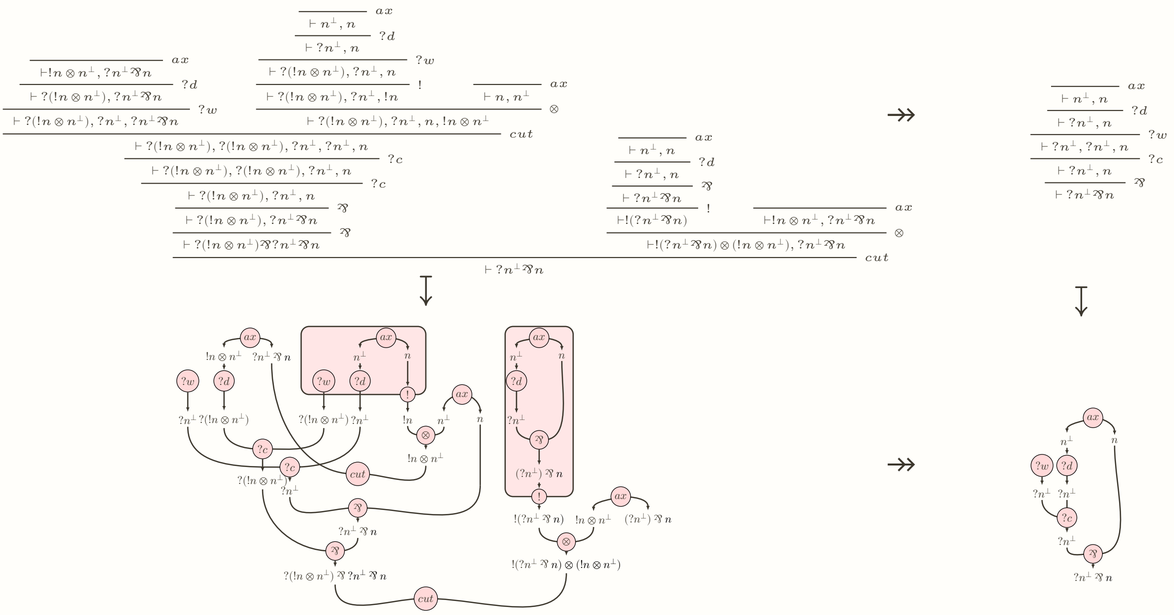


$\longrightarrow_{(!\text{-}?c)}$



This rule involves box **cloning**.

An example of cut elimination



Some properties of cut elimination

The following hold for cut elimination of MELL proof nets [Gir87; Gir93; PF10]:

1. **(Cut Elimination)** All cuts of an MELL proof net can be eliminated.
2. **(Stability)** An MELL proof net is still a proof net after cut elimination.
3. **(Confluence)** Cut elimination is confluent on MELL proof nets.
4. **(Strong Normalization)** Cut elimination is strongly normalizing.

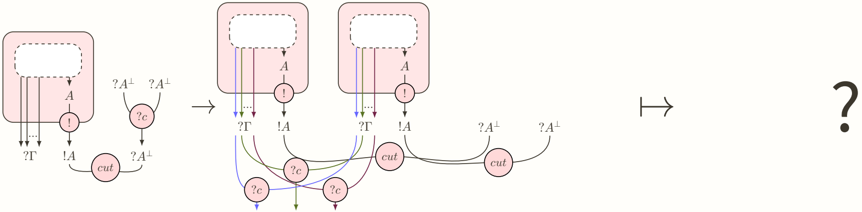
Research question, recap

We wish to establish a closer connection between

(1) LMNtal (concrete, general-purpose programming language based on hierarchical graph rewriting) and

(2) MELL proof nets (hierarchical graph rewriting system for a logical system)

by providing (1) with well-motivated **graph cloning and deletion constructs** inspired by (2).



Syntax extension for process contexts

We define a new syntactic construct, **process context aggregates**:

$$T \quad ::= \quad \cdots \quad | \quad \$p[{}^*X_1, {}^*X_2, \ldots, {}^*X_n] \quad (n > 0)$$

where each ${}^*X_i$ is a bundle which (i) appears in a process context with the same name and (ii) has the same number of links, i.e., $|{}^*X_1| = |{}^*X_2| = \cdots = |{}^*X_n|$

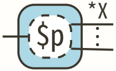
This construct represents (dynamically determined) $|{}^*X|$ copies of n-ary $\$p$'s.

→ **We can now represent an unspecified number of wildcards.**

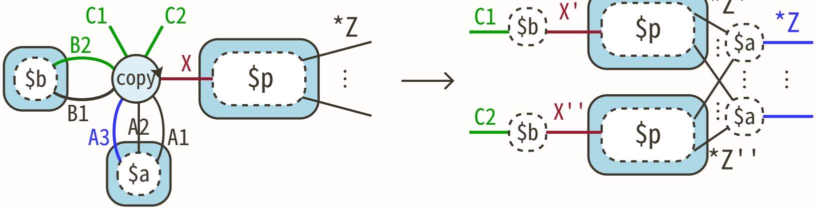
- An example usage on the next page.

API design and implementation

We can describe cloning and deletion using process context aggregates.

For instance, `mell.copy` clones 

as:



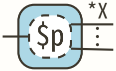

```
mell.copy(X,A1,A2,A3,B1,B2,C1,C2), {$p[X|*Z]}, {$a[A1,A2,A3]}, {$b[B1,B2]}
  → {$p[X'|*Z']}, {$p[X''|*Z'']}, $a[*Z',*Z'',*Z], $b[X',C1], $b[X'',C2].
```

We implemented it as API of SLIM runtime https://github.com/lmntal/slim/.

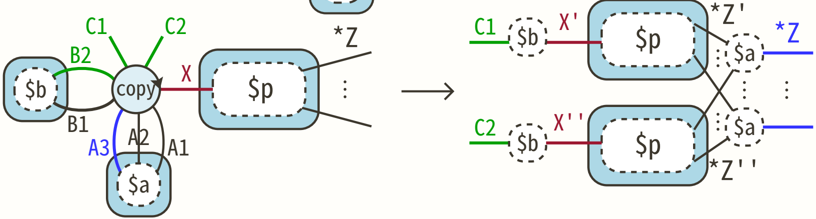
- Deletion is also defined in the similar way. Please see the paper for details.

API design and implementation

We can describe cloning and deletion using process context aggregates.

For instance, mell.copy clones 

as:



Represents an unspecified number (|*Z|) of $a

```
mell.copy(X,A1,A2,A3,B1,B2,C1,C2),{$p[X|*Z]},{$a[A1,A2,A3]},{$b[B1,B2]}
  → {$p[X'|*Z']},{$p[X''|*Z'']},$a[*Z',*Z'',*Z],$b[X',C1],$b[X'',C2].
```

We implemented it as API of SLIM runtime https://github.com/lmntal/slim/.

- deletion is also defined in the same way. Prease see the paper for details.

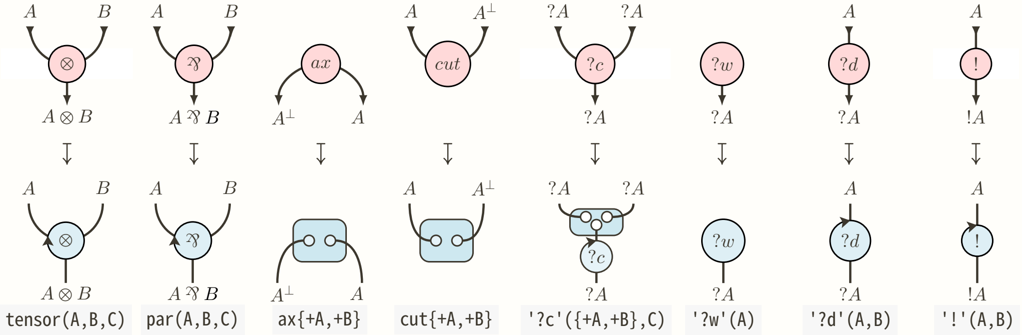
Encoding MELL proof nets in LMNtal, overview

We showed that **MELL proof nets and their cut elimination rules can be directly encoded** in LMNtal

(i.e., each cut elimination rule encoded into one LMNtal rule).

1. We encoded MELL proof structures.
2. We encoded cut elimination rules.
3. We constructed and visualized the state space of cut elimination.
4. We encoded some additional proposed rules and checked their consequences.

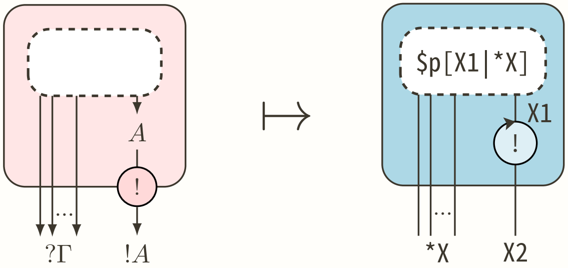
Encoding MELL proof structure (1): Cells and wires



- **Non-commutativity of inputs** to $\otimes$ and $⅋$ cells is represented by **atoms**,
- while the **commutativity of the arguments** of ax and cut is represented using **membranes**.
- A $?c$ cell with **commutative inputs and a single output** is represented by using both **an atom and a membrane**.

Encoding MELL proof structure (2): Promotion box

$$\frac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}\ !$$



`{'!'(X1,X2),$p[X1|*X]}.`

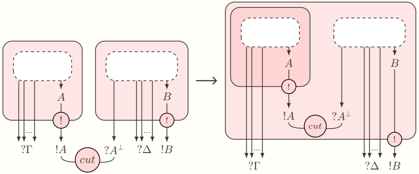
- The outer frame of the box is represented by a **membrane**.
- $?\Gamma$ is represented by a **bundle** `*X` .
- The blank space is represented by a **process context** `$p[X1|*X]` .

✔ **The MELL proof structure can be represented by a combination of cells, wires and boxes.**

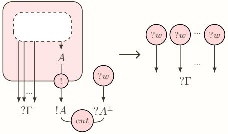
Encoding cut elimination rules

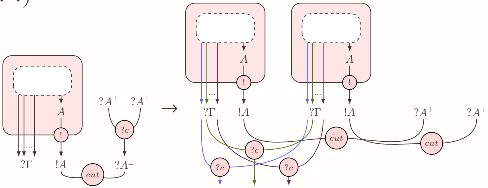
We showed that all six cut elimination rules of MELL proof nets can be directly encoded into LMNtal with the `mell` API.

The following three rules involve non-trivial box operations:



(!-!)

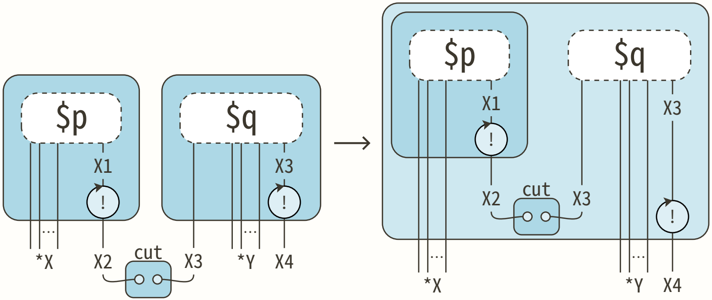




$(!\text{-}?w)$ (see the paper) $(!\text{-}?c)$

Encoding the cut elimination rule (!-!)

LMNtal encoding of (!-!):

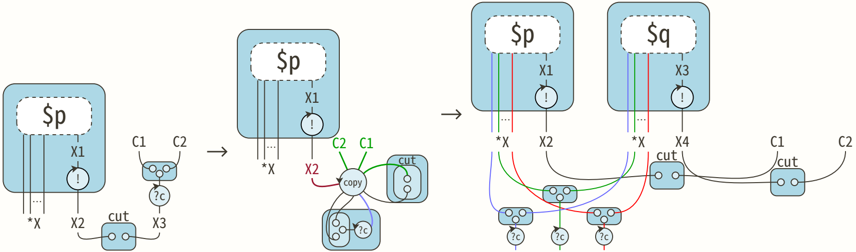



```
{'!'(X1,X2), $p[X1|*X]}, {$q[X3|*Y]}, cut{+X2,+X3}
  :- { {'!'(X1,X2), $p[X1|*X]}, $q[X3|*Y], cut{+X2,+X3}}.
```

The migration of the box can be represented by the movement of the braces.

Encoding the cut elimination rule $(!\text{-}?c)$

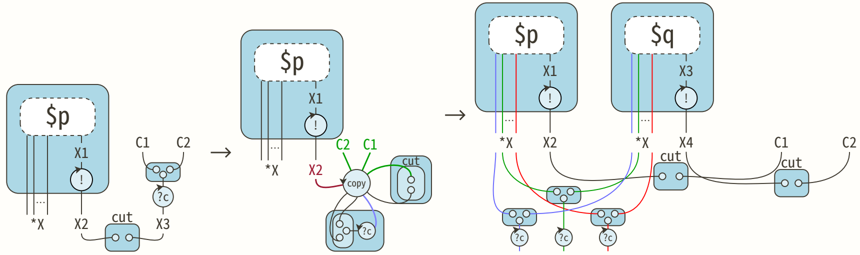
LMNtal encoding of $(!\text{-}?c)$:




```
{'!'(X1,X2), $p[X1|*X]}, '?c'({+C1,+C2},X3), cut{+X2,+X3},
  :- mell.copy(X2,A1,A2,A3,B1,B2,C1,C2), {'!'(X1,X2), $p[X1|*X]},
     {'?c'({+A1,+A2},A3)}, {cut{+B1,+B2}}.
```

Encoding the cut elimination rule (!-?c)

Encoding of (!-?c):




```
{'!'(X1,X2), $p[X1|*X]}, '?c'({+C1,+C2},X3), cut{+X2,+X3},
  :- mell.copy(X2,A1,A2,A3,B1,B2,C1,C2), {'!'(X1,X2), $p[X1|*X]},
     {'?c'({+A1,+A2},A3)}, {cut{+B1,+B2}}.
```

Three rules were necessary with nlmem, but now it can be encoded with one rule.

Encoding MELL proof nets: State space (1)

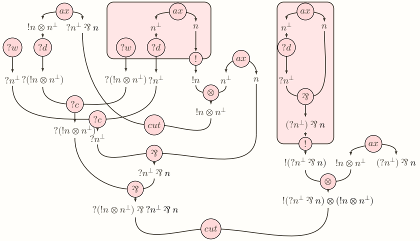
Example: β-reduction of the simply typed λ-calculus

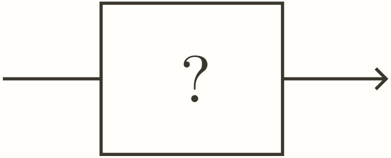
- By the CH isomorphism, the type derivation tree has a corresponding net, and its cut elimination corresponds to β-reduction [Gir87].

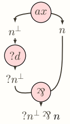
Initial State **Normal Form**

$$(\lambda f : n \to n \,.\, \lambda x : n \,.\, f\ x)(\lambda x : n \,.\, x) \longrightarrow \lambda x : n \,.\, (\lambda x : n \,.\, x)\ x \longrightarrow \lambda x : n \,.\, x$$

↧ **State Space** ↧

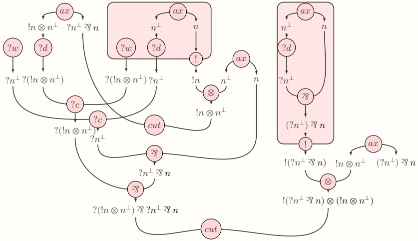




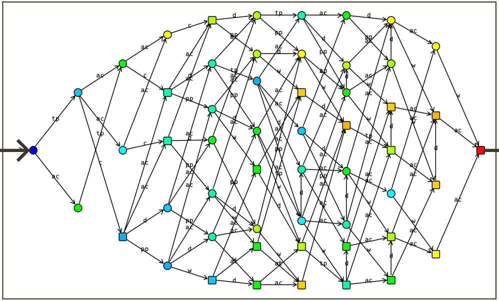


Encoding MELL proof nets: State space (2)

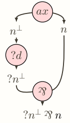
Initial State



State Space (Reduction Paths)



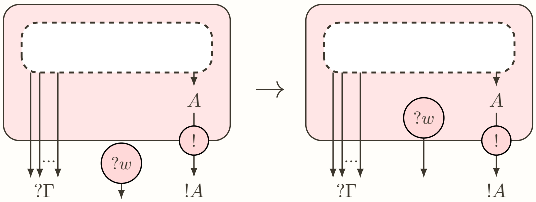
Normal Form



✔ We confirmed **well-known properties** such as CR and SN, as well as **non-trivial properties** such as the existence of redundant paths.

Encoding MELL proof nets: Addition of rules (1)

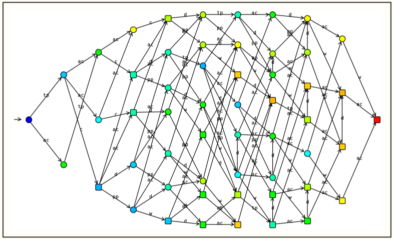
Additional rules have been proposed in the literature, and we can readily observe changes in the state space. For example, the rule



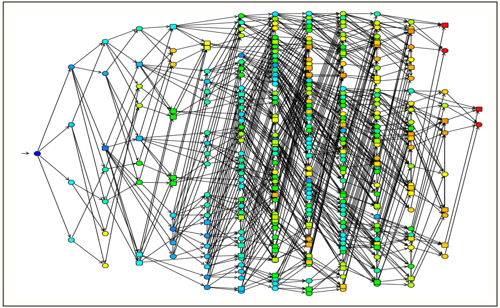

```
{'!'(X1,X2), $p[X1|*X]}, '?w'(X3) :- {'!'(X1,X2), '?w'(X3), $p[X1|*X]}.
```

is used in the expression of substitution [Vau07], but confluence is lost by this rule [Tra11] (next slide).

Encoding MELL proof nets: Addition of rules (2)



Before (1 end state)



After (4 end states)

- The number of final states increased to 4, losing confluence.

✓ Rules could be easily added.

✓ Consequences of rule addition could be visually confirmed.

→ **LMNtal with the `mell` API is useful as a workbench for proof nets.**

Encoding Process Calculi

We showed that several process calculi can be encoded in LMNtal.

Example: Replication rule of the **Ambient Calculus**, where **ambients are box structures that may be duplicated, migrated and deleted**

$$!(\mathrm{open}\ m.P) \mid m[Q] \to P \mid Q \mid !(\mathrm{open}\ m.P)$$

```
open_repl@@ open_repl(M,{$p}),{amb(M1),{id,+M1,-M2,$mm},$q,@q},{id,+M,+M2,$m}
  :- mell.copy({$p},A1,A2,A3,B1,B2,remove,P),{cp(A1,A2,A3)},{B1=B2},
     $q,{id,+M3,$m,$mm},open_repl(M3,P).
open_repl_aux@@ remove({$p}) :- $p.
```

➔ **Sufficiently general for modeling and computation.**

Other Related Work

- Some **hierarchical graph rewriting systems based on Proof Nets** [Mur20; AFM11] are useful as models of functional languages, whereas LMNtal is desighned as general-purpose modeling language.
- **Linear Logic Programming languages** [HM94; Hod+98] are based on LL, but our work is based on proof nets.
- **Quantifiers in graph rewriting** enables the handling of an indefinite number of atoms [Gha+12; MU24], whereas our work is concerned also with handling an unspecified number of free links in port graph rewriting.

Conclusion

We introduced `mell` API, which is based on the cut elimination rules of MELL proof nets, to perform cloning and deletion of hierarchical graphs in LMNtal.

Hierarchical graph rewriting language acquired:

- ✓ a **PL-style**, **validated**, and **general-purpose** **cloning and deletion constructs**

$\leftrightarrow$

Proof nets acquired:

- ✓ a **concrete PL** that enables concise encoding of proof nets
- ✓ and a **useful workbench**.

→ Beneficial bridge to both sides

Reference I

[Ued09] Kazunori Ueda. "LMNtal as a hierarchical logic programming language". In: *Theoretical Computer Science* 410.46 (2009), pp. 4784–4800. DOI: 10.1016/j.tcs.2009.07.043.

[Gir87] Jean-Yves Girard. "Linear logic". In: *Theoretical Computer Science* 50.1 (1987), pp. 1–101. DOI: 10.1016/0304-3975(87)90045-4.

[BB92] Gérard Berry and Gérard Boudol. "The chemical abstract machine". In: *Theoretical Computer Science* 96.1 (1992), pp. 217–248. DOI: 10.1016/0304-3975(92)90185-I.

[Mil01] Robin Milner. "Bigraphical Reactive Systems". In: *12th International Conference on Concurrency Theory (CONCUR '01)*. Vol. 2154. LNCS. Berlin, Heidelberg: Springer-Verlag, 2001, pp. 16–35. DOI: 10.1007/3-540-44685-0_2.

[EFP18] Nneka Chinelo Ene, Maribel Fernández, and Bruno Pinaud. "Attributed hierarchical port graphs and applications". English. In: *Electronic Proceedings in Theoretical Computer Science, EPTCS* 265 (Feb. 2018), pp. 2–19. DOI: 10.4204/eptcs.265.2.

Reference II

[Inu+08] Atsuyuki Inui et al. "LMNtal: The Unifying Programming Language Based on Hierarchical Graph Rewriting". In: *Computer Software* 25.1 (2008), 1_124–1_150. DOI: 10.11309/jssst.25.1_124.

[Cor+06] Andrea Corradini et al. "Sesqui-Pushout Rewriting". In: *Graph Transformations (ICGT 2006)*. Vol. 4178. LNCS. Berlin, Heidelberg, 2006, pp. 30–45. DOI: 10.1007/11841883_4.

[OER21] Roy Overbeek, Jörg Endrullis, and Aloïs Rosset. "Graph Rewriting and Relabeling with PBPO+". In: *Graph Transformation (ICGT 2021)*. Ed. by Fabio Gadducci and Timo Kehrer. Vol. 12741. LNCS. Cham: Springer, 2021, pp. 60–80. DOI: 10.1007/978-3-030-78946-6_4.

[BES18] Jon Haël Brenas, Rachid Echahed, and Martin Strecker. "Verifying Graph Transformation Systems with Description Logics". In: *Graph Transformation*. Ed. by Leen Lambers and Jens Weber. Vol. 10887. LNCS. Cham: Springer, 2018, pp. 155–170. DOI: 10.1007/978-3-319-92991-0_10.

[Gir93] Jean-Yves Girard. "Linear Logic: A Survey". In: *Logic and Algebra of Specification*. Ed. by F.L. Bauer, W. Brauer, and H. Schwichtenberg. Vol. 94. NATO ASI Series. Springer Berlin Heidelberg, 1993, pp. 63–112. DOI: 10.1007/978-3-642-58041-3_3.

Reference III

[PF10] Michele Pagani and Lorenzo Tortora de Falco. "Strong normalization property for second order linear logic". In: *Theoretical Computer Science* 411.2 (2010), pp. 410–444. DOI: `10.1016/j.tcs.2009.07.053`.

[Vau07] Lionel Vaux. "λ-calcul différentiel et logique classique : interactions calculatoires". Theses. Université de la Méditerranée - Aix-Marseille II, Jan. 2007.

[Tra11] Paolo Tranquilli. "Intuitionistic differential nets and lambda-calculus". In: *Theoretical Computer Science* 412.20 (2011), pp. 1979–1997. DOI: `https://doi.org/10.1016/j.tcs.2010.12.022`.

[Mur20] Koko Muroya. "Hypernet semantics of programming languages". Ph.D. thesis. University of Birmingham, 2020.

[AFM11] Sandra Alves, Maribel Fernández, and Ian Mackie. "A new graphical calculus of proofs". In: *Electronic Proceedings in Theoretical Computer Science* 48 (Feb. 2011). DOI: `10.4204/EPTCS.48.8`.

Reference IV

[HM94] J.S. Hodas and D. Miller. "Logic Programming in a Fragment of Intuitionistic Linear Logic". In: *Information and Computation* 110.2 (1994), pp. 327–365. DOI: https://doi.org/10.1006/inco.1994.1036.

[Hod+98] Joshua S. Hodas et al. "Efficient Implementation of a Linear Logic Programming Language". In: *IJCSLP*. 1998.

[Gha+12] Amir Hossein Ghamarian et al. "Modelling and analysis using GROOVE". In: *International Journal on Software Tools for Technology Transfer* 14.1 (Feb. 2012), pp. 15–40. DOI: 10.1007/s10009-011-0186-x.

[MU24] Haruto Mishina and Kazunori Ueda. "Introducing Quantification into a Hierarchical Graph Rewriting Language". In: *34th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR 2024)*. Vol. 14919. LNCS. 2024, pp. 220–239. DOI: 10.1007/978-3-031-71294-4_13.