A Formal Model of A New Linear Time Correctness Condition for Multiplicative Linear Logic

Satoshi Matsuoka

AIST

Motivation

- Can specify many computational problems by Linear Logic formulas
- Be able to specify many NP-complete problems using MLL formulas
- Can Use Proof Search as Problem Solving
- Proof Net Construction: a method of LL Proof Search
  - ✓ Construct a proof structure associated with the input specification formula,
    and then check whether it is a proof net using a *correctness condition*
  - ✓ Efficient (Linear Time) Correct Condition is a Key

Brief History of CCs of MLL proof nets

- 1987: Girard

  Introduction of Proof Nets

  Long Trip Condition for unit-free Multiplicative Linear Logic (MLL)
- 1989: Danos & Regnier

  Acyclic & Connected Condition over DR-graphs
- 1999: Guerrini

  The First Linear Time Correctness Condition based on Danos's Contractability Condition
- 2000: Murawski & Ong

  Another Linear Time Correctness Conditions based on Lamarche's Condition of IMLL proof nets
- 2007: de Naurois & Mogbil

  New Correctness Condition Establishing NL-completeness: we only need just one DR-graph

Our result

- A new linear time correctness condition based on de Naurois & Mogbil‘s work
- Introduction of deNM-trees
- Rewriting system over deNM-trees with three rewrite rules
- Use of union-find data structures in order to establish linear time termination
- Introduction of deadlock prevention mechanism in order to establish completeness of the algorithm
- Easy to implement (in Proof Net Calculator)
- A formal model (a finite transition system) based on the implementation

The system MLL

MLL formulas: $A ::= p \mid p^\perp \mid A \otimes B \mid A ⅋ B$

negation : $(p^\perp)^\perp := p, \qquad (A \otimes B)^\perp := B^\perp ⅋ A^\perp, \qquad (A ⅋ B)^\perp := B^\perp \otimes A^\perp$

Inference rules:

ID $\dfrac{}{\vdash A, A^\perp}$

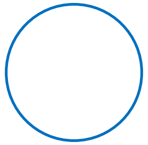
$\otimes$ $\dfrac{\vdash \Sigma_1, A \qquad \vdash B, \Sigma_2}{\vdash \Sigma_1, \Sigma_2, A \otimes B}$

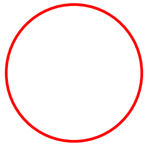
⅋ $\dfrac{\vdash \Sigma, A, B}{\vdash \Sigma, A ⅋ B}$

Cut $\dfrac{\vdash \Sigma_1, A \qquad \vdash A^\perp, \Sigma_2}{\vdash \Sigma_1, \Sigma_2}$ Cut is almost $\otimes$

$\Sigma, \Sigma_1, \Sigma_2$ are multisets of MLL formulas

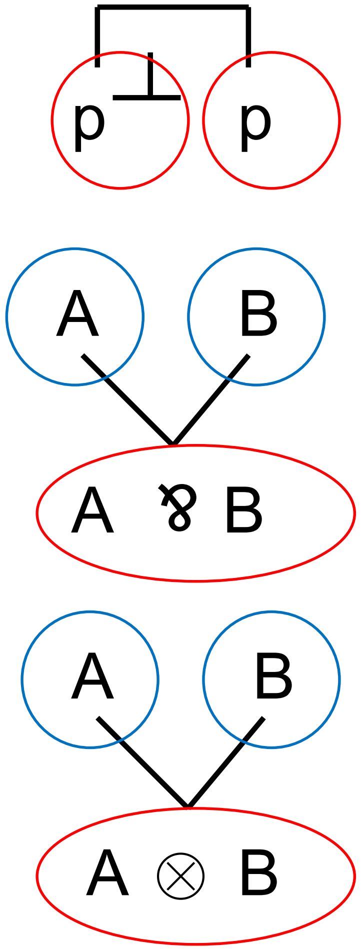
Links





premises

conclusions



ID-Links

Par-Links

Tensor-Links

Multiplicative-Links

MLL proof structures

- A set of links
- Satisfying two conditions:

1. any formula (occurrence) is a conclusion of exactly one link
2. any formula (occurrence) is at most one premise of a link

MLL Proof Nets

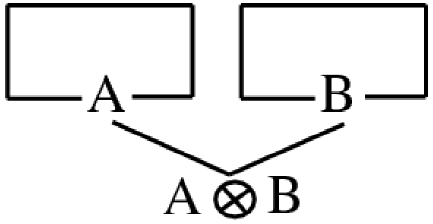
- Abstract formal proofs of Multiplicative Linear Logic
- Sequentializable MLL proof structures

Definition of Proof Nets: Sequentializability

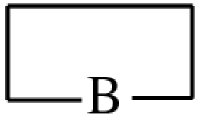
(1) ID–links are sequentializable

(2) 

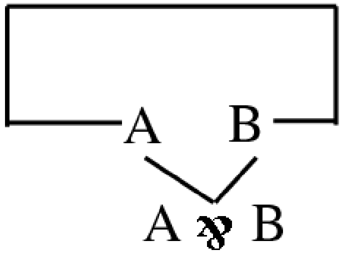
sequentializable if 

= 

and bofh 

and 

sequentializable

(3) 

sequentializable if 

sequentializable

A DR-switching S for a proof structure Θ

- A function from the set of par-links in Θ to {Left, Right}

The DR graph Θ_S induced by DR-switching S

(1) if $\overline{A \qquad A^{\perp}}$ occurs in Θ then A —— $A^{\perp}$ is an edge of Θ_S

(2) if $\dfrac{A \qquad B}{A \otimes B}$ occurs in Θ then $\dfrac{A}{A \otimes B}$ and $\dfrac{B}{A \otimes B}$ are two edges of Θ_S

(3) if $\dfrac{A \qquad B}{A \parr B}$ occurs in Θ and $S\left(\dfrac{A \qquad B}{A \parr B}\right) = L$ then $\dfrac{A}{A \parr B}$ is an edge of Θ_S

(4) if $\dfrac{A \qquad B}{A \parr B}$ occurs in Θ and $S\left(\dfrac{A \qquad B}{A \parr B}\right) = R$ then $\dfrac{B}{A \parr B}$ is an edge of Θ_S

A Graph-theretic characterization of proof nets

- Theorem (Girard, Danos-Regnier)
  A MLL proof structure Θ is a MLL proof net
  iff
  for any DR-switching S for Θ,
  the DR-graph ΘS is acyclic and connected

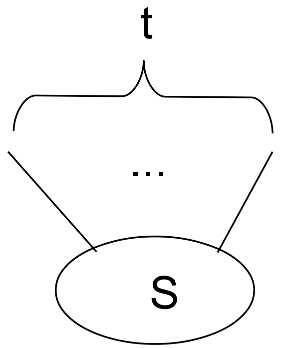
Our new linear time correctness condition

1. Given an MLL proof structure Θ
2. Select a DR-switching S for Θ
3. If the DR-graph S(Θ) is not acyclic and connected, then Θ is not a proof net
4. Otherwise, construct the *deNM*-tree T for Θ and S,
    1. check whether or not T can reduce to *exactly one* node deNM-tree using three rewrite rules
    2. If it succeeds, then Θ is a proof net
    3. Otherwise, not a proof net

deNM-trees

- Trees consisting of the following two types of nodes:

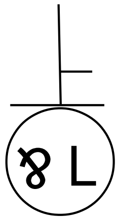
labeled-node



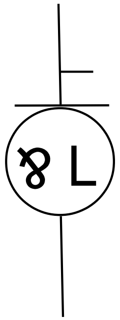
S is a set of labels l_L or r_L where L is a ⅋-link

$t \geq 0$

⅋ -node (degree 1)

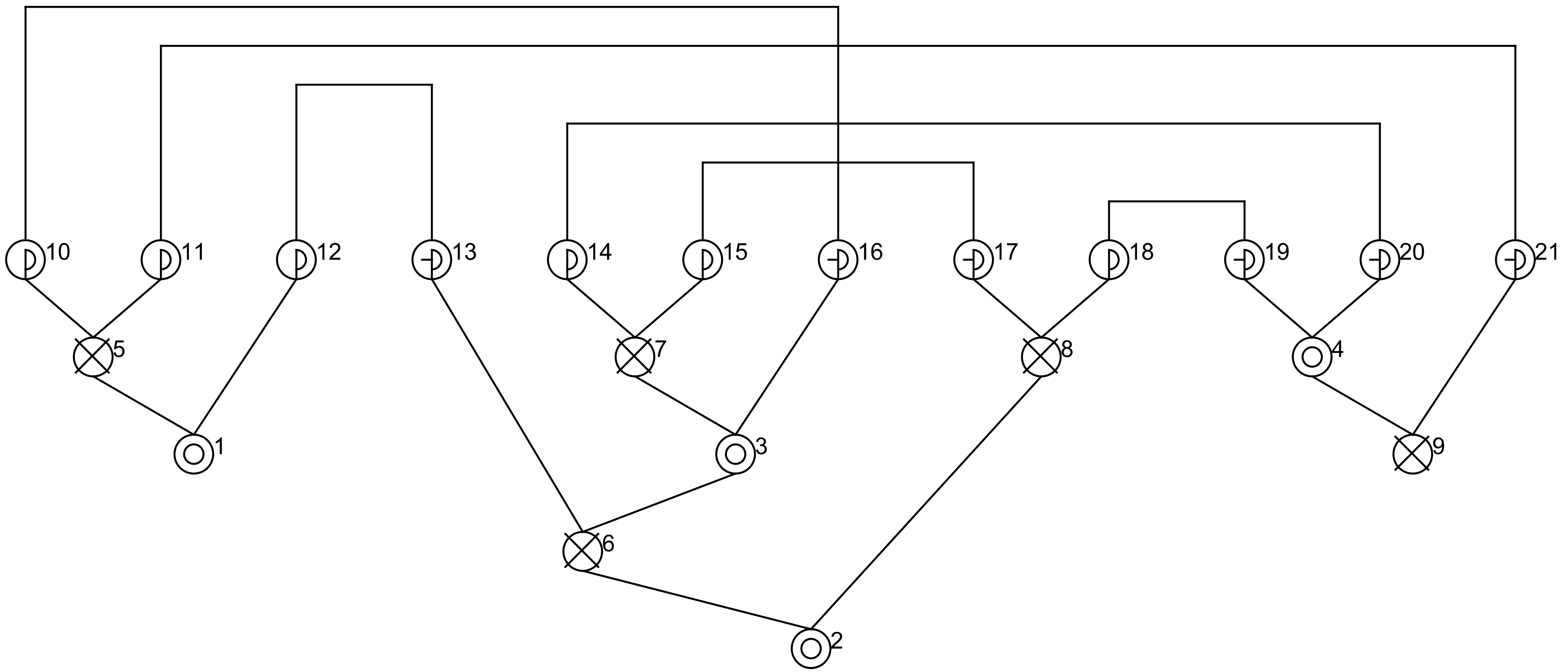


⅋ -node (degree 2)

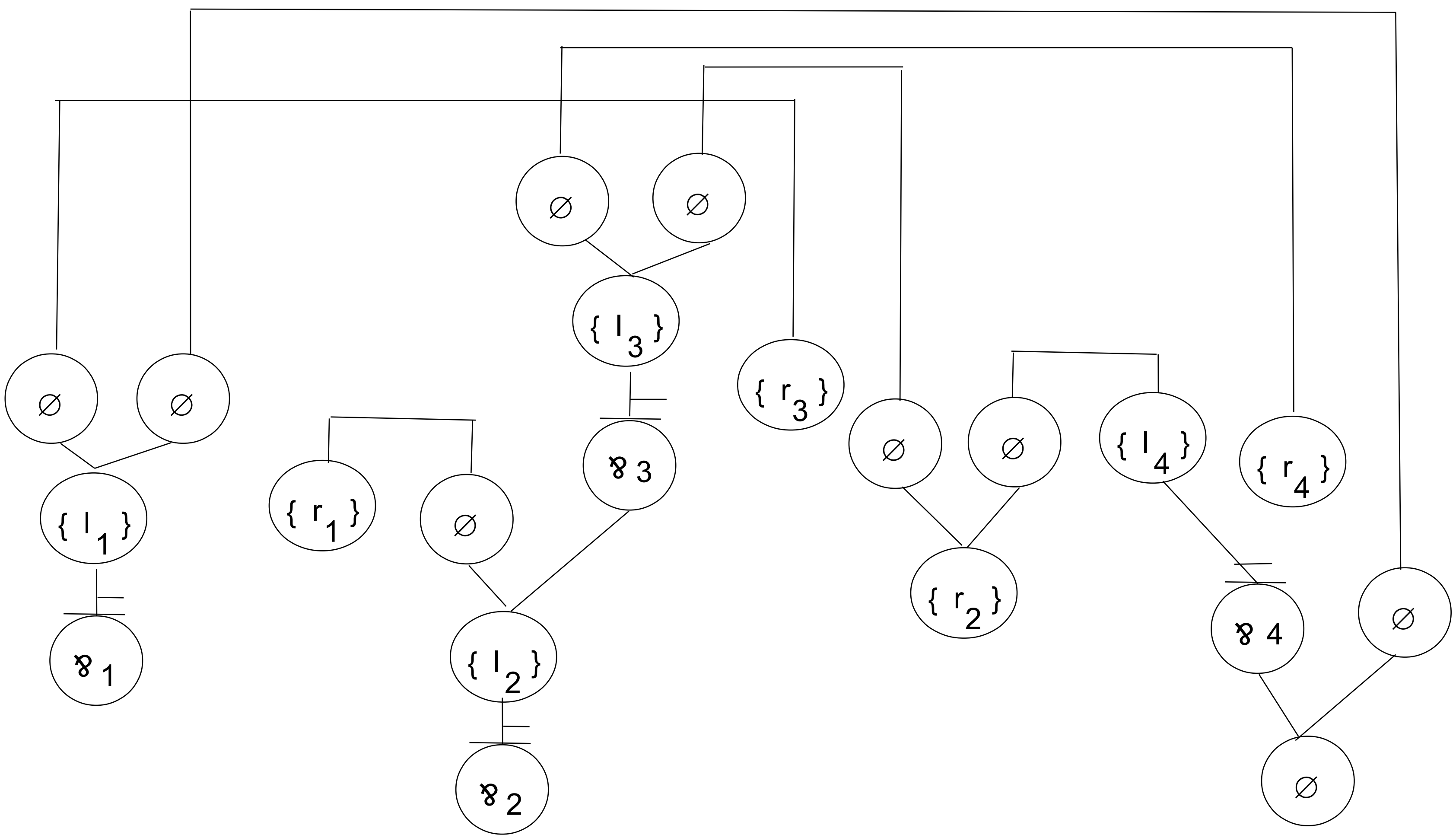


Example 1: An MLL proof net

PN-1.txt



Example 1: its deNM-tree

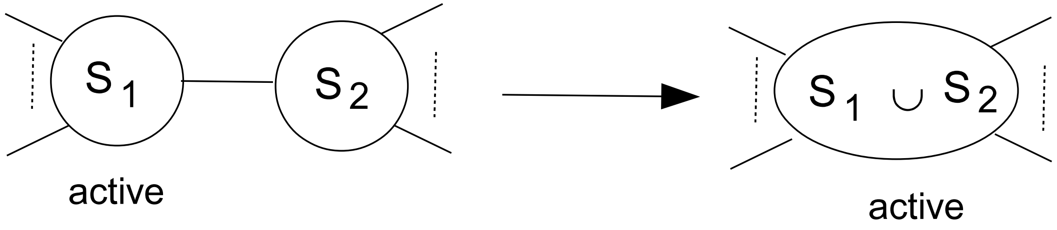


Rewriting System

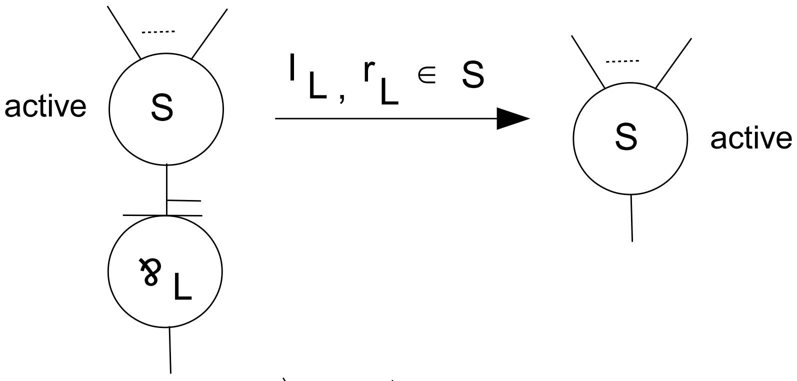
- Rewriting System over deNM-trees
- Only three rewrite rules
- Choose an active (labeled) node arbitrarily

Three rewrite rules

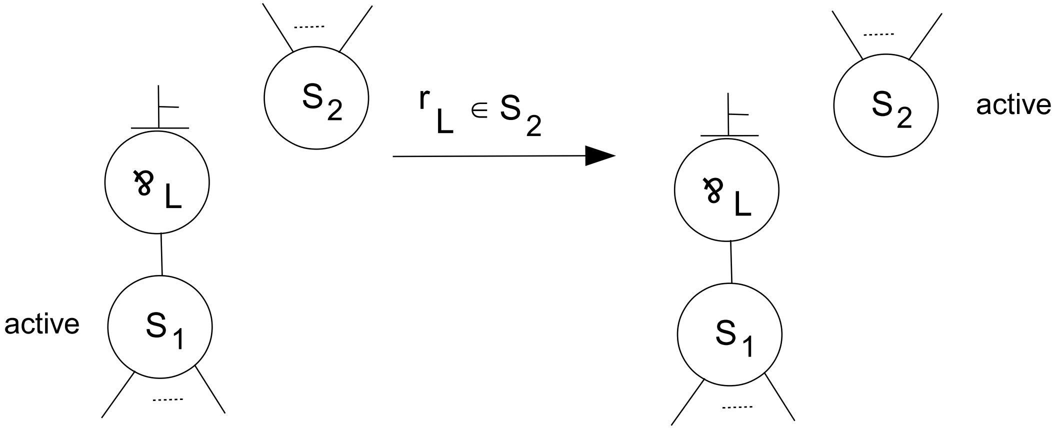
union



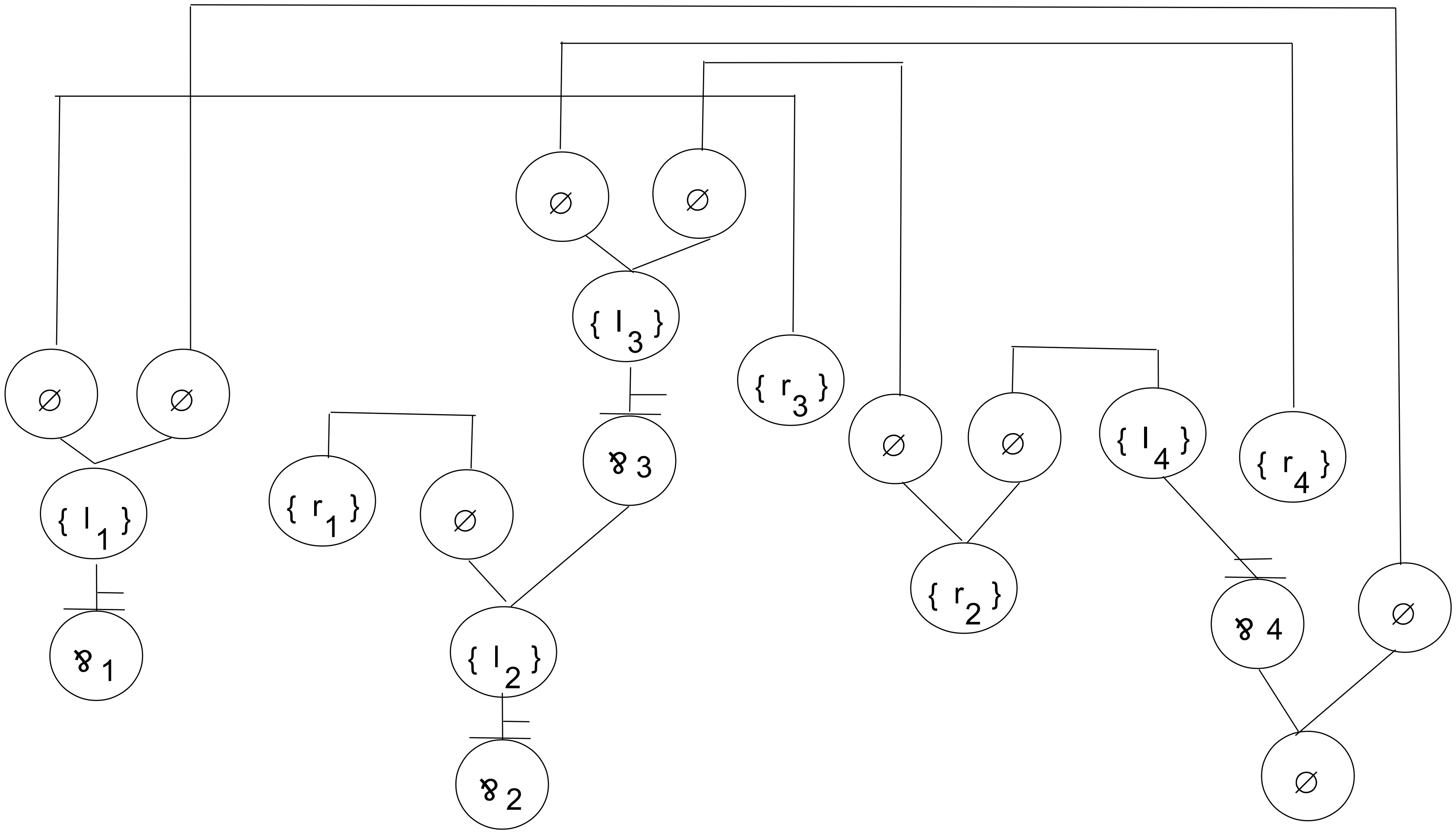
⅋ -elimination



local jump



Example 1: its deNM-tree

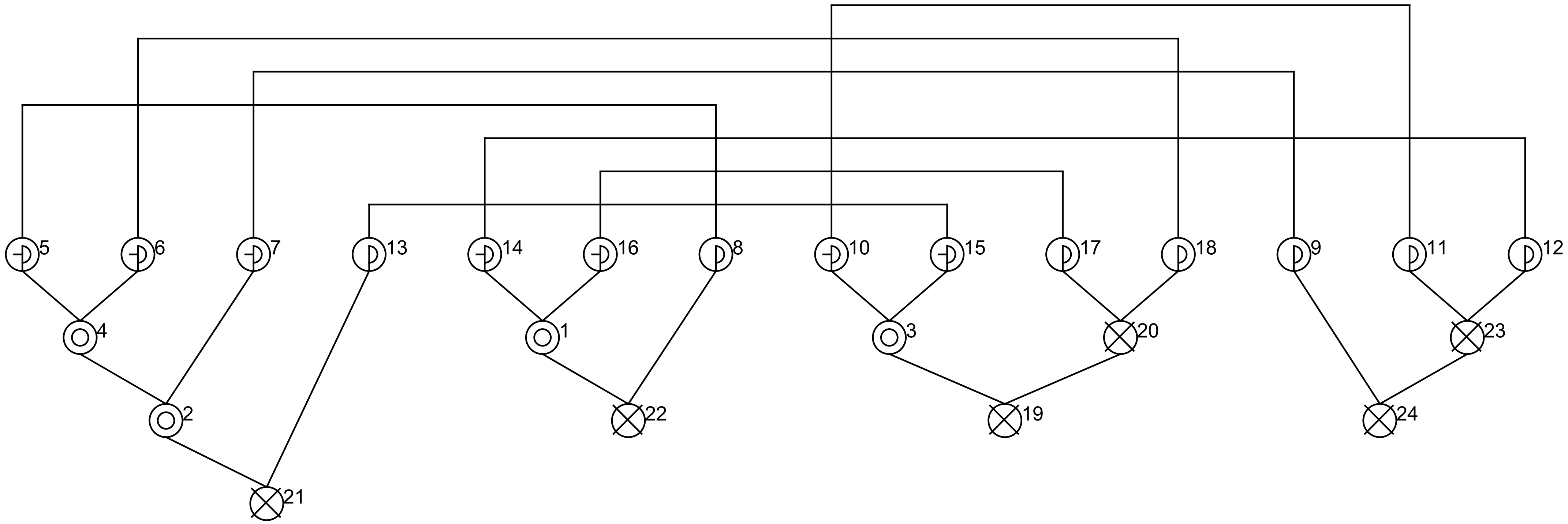


Example 1: the reduced one node deNM-tree

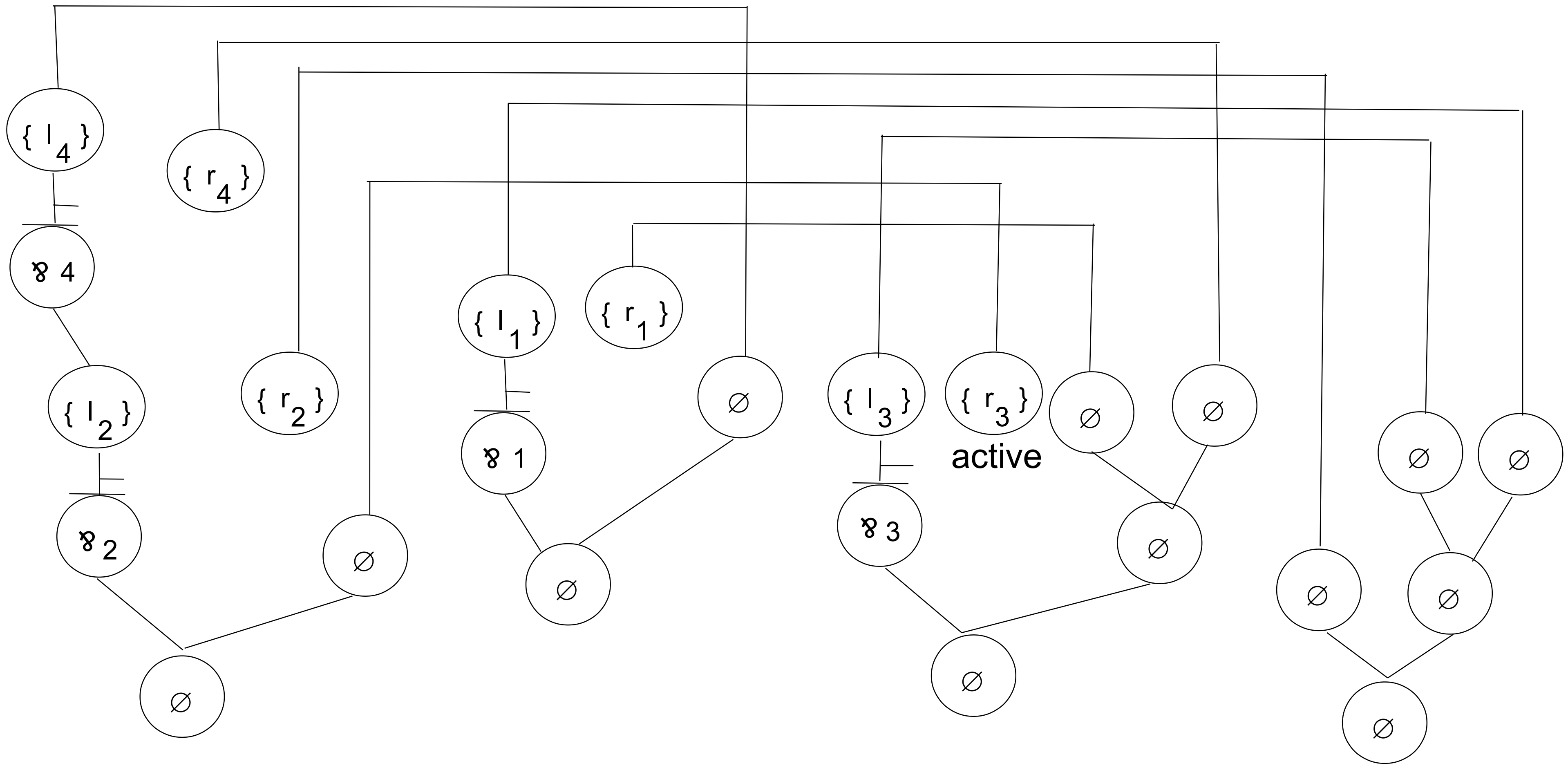
$$\{ l_1, r_1, l_2, r_2, l_3, r_3, l_4, r_4 \}$$

Example 2: An MLL proof structure, but not PN

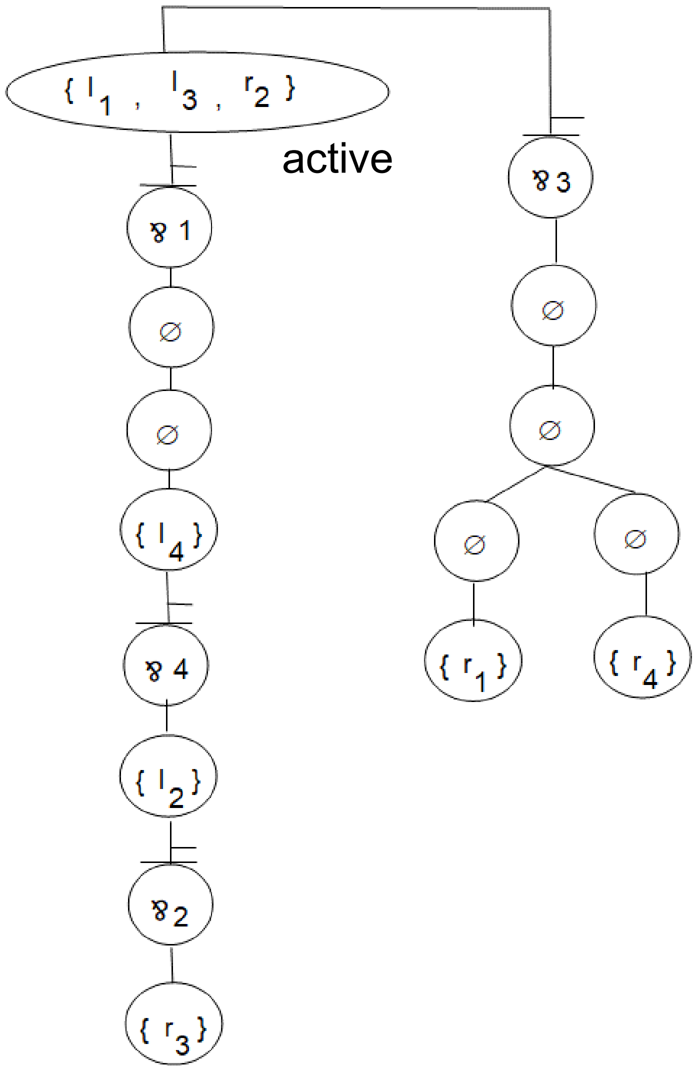
nonPN-1.txt



Example 2: its deNM-tree



Example 2: the reduced deNM-tree



Remark

- Any naïve implementation has worst cases terminating in quadratic time
- Need of more sophisticated data structures: union-find data structures

Union-Find Data Structures

- Maintenance of a partition on a finite set $S = \{a_1, a_2, \cdots, a_n\}$:
  - Each partition (subset) S_a of S has the representative element a
  - union(a, b): makes the union set $S_a \cup S_b$, where
    S_a (resp. S_b) is the current subset including a (resp. b)
  - find(a): return the representative element b of S_b that includes a
- Can check whether two elements a and b belong to the same partition
- Runs in almost linear time (in the sense of amortized cost)
- Runs in linear time over special cases, especially over *trees*

Our finite transition system

- States $\langle a, N, n, P, S_{\mathrm{elim}}, \mathrm{num}_{\mathrm{labeled}}, \mathrm{num}_{\infty} \rangle$
  - ✓ a denotes the current active node
  - ✓ N_i maintains the information about labeled node *i*
  - ✓ n_i denotes the representative element of the partition including *i*
  - ✓ P_j maintains the information about ∞ -node *j*
  - ✓ etc.

Our transition system (Cont.)

- Four transition rules:

  $\invamp$-elimination rewrite rule is divided into two transition rules
- Use of union-find data structures

✓union operation in union transition rule

✓find operation in two $\invamp$-elimination transition rules for the elimination condition

✓find operation in local jump transition rule in order to find the next active node

- Each transition rule only uses *constant number* of Union-Find operations

$\Rightarrow$ *Linear time termination*

Deadlock prevention mechanism

- Without it, we would judge that correct proof nets were not:

  can not establish completeness of the algorithm
- Realized by *queue data structures* and *union-find data structures*

✓Indices for not yet eliminated ⅋-nodes are maintained in queues

- Its correctness (deadlock freedom) can be stated by a *liveness property:*

  "Indices for not yet eliminated ⅋-nodes *must* be always in some queues"

Additional Results

- (Yet another) linear time sequentialization algorithm
  - Easy to implement
  - The basic idea is to repeat linear time CC twice
- Linear time algorithm for automated generation of planar proof nets
  - can be reduced to decremental graph connectivity problem

Our implementation

- Can be downloaded from

  https://staff.aist.go.jp/s-matsuoka/PNCalculator/index.html

  as a software package called *Proof Net Calculator*

Future Work

- Incorporation of backtracking mechanism
  Seeking elegant (verifiable) implementation
- Semi-persistent data structures?
- Extensions to other fragments and/or variants of LL

Thank you